Crawling JavaScript websites using WebKit – with application to analysis of hate speech in online discussions

Hugo Hammer[1], Alfred Bratterud and Siri Fagernes
Oslo and Akershus University College of Applied Sciences
Institute of Information Technology

Abstract

JavaScript Client-side hidden web pages (CSHW) contain dynamic material created as a result of specific activities from the users. Websites relying on such technology are increasingly common. Crawling the so-called *Hidden Web* is challenging, particularly when JavaScript CSHW from an external website is seamlessly included as part of the web pages.

We have developed a prototype for a web crawler that efficiently extracts content from CSHW. The crawler uses *WebKit* to render web pages and to emulate human web page activities to reveal dynamic content. The WebKit crawler was used to collect text from 39 Norwegian online newspaper debate articles, where the online user discussions were included as JavaScript CSHW from other websites. The average speed to extract the main content and the JavaScript-generated discussions were 36.3 kB/sec and 8.8 kB/sec, respectively. Analyzing the collected text from the news paper debate articles using opinion mining, documents that the debate articles are more positive to Islam and Muslims than the following discussions. The results demonstrate the importance of being able to collect such JavaScript CSHW discussion content to get an overview of existing hate speech on the Internet.

1 Introduction

Over the past years there has been an alarming growth in hate against minorities like Muslims, Jews, gypsies, gays and women in Europe, driven by right wing populism parties and extremist organizations [12] [28]. A similar increase in hate speech is observed on the Internet [14] [4]. It has repeatedly been concerns that individuals influenced by such web content may resort to violence [24] [25]. Social media and online discussions on the Internet contains a wealth of information potentially relevant to improve the understanding of the extent of hate speech on the Internet and the risks of violence it may cause. However, it turns out that academia is lacking research on social media and online radicalization [26].

In addition to analysis of hate speech, automatic analysis of social media has recently received much attention in other areas. Companies can use social media information to better understand consumers' attitudes to their products or make online marketing targeted toward each costumer. Over the resent years such analysis has become big industry with large international actors like IBM and SAS Institute [8] [15]. Journalists can use social media analysis to quickly identify trends or rumors or see the public response to world events.

Social media analysis relies on collecting and analyzing text from online discussions. Software programs that traverse the Internet following hypertext links to collect web

[1]hugo.hammer@hioa.no

content is referred to as web *crawlers*, *spiders* or *ants* [9]. Social media platforms like *Facebook*, *Twitter* and *Disqus* increasingly rely on JavaScript to provide users with fast and smooth user experiences. Only a limited amount of content is available when the user first enters the website. More content is only revealed after specific activities from the user, like scrolling down the page or pressing buttons. Such websites is typically referred to as client-side hidden websites (CSHW) [18]. Text collection from such sites is challenging, but some crawlers like *Heritrix*, *Nutch* and *Googlebot* still document satisfying results [18].

Other websites make text collection even more challenging. A number of online newspapers and blogs include discussions on their websites by engaging external services like *Disqus*. The discussion website is seamlessly included through JavaScript as part of the news/blog website. This makes the discussion content unavailable in the newspaper/blog source file even after the content is revealed to the user. For most purposes it is the news articles and not the subsequent discussions that are of most importance, but related to social media analysis it is the other way around. The performance of existing crawling systems to such web pages is not known.

The work presented in this paper is part of a larger project where the overall goal is to contribute in the combat against the worrying trend with more hate speech on the Internet. We have developed a web crawler that efficiently collects text from JavaScript CSHW such as the ones described above. The crawler use WebKit [27] to render the web pages and to automatically emulate human web page activities to reveal dynamic content.

We use the WebKit crawler to investigate one piece in the understanding of hate speech on the Internet, to see if there is a difference in opinions towards immigration, Islam and Muslims for common newspaper debaters like politicians, journalists and academics and participants in related online discussions. In this paper we analyze the performance of the web crawler and argue that our solution is preferable compared to other solutions presented in the literature.

The paper is organized as follows. In Section 2 different approaches to crawling the hidden web are reviewed. In Sections 3 and 4 we describe and evaluate our WebKit crawler. In Section 5 we analyze the collected text to see if there are differences in opinions in debate articles and the subsequent online discussions, and finally Section 6 provides some conclusions.

2 Crawling the Hidden Web

The so-called *hidden web* or *deep web* defines the part of the web that is not accessible directly through simply following hyperlinks. Hidden web content includes a range of dynamic material, for instance pop-up menus, content behind web forms, which is only reachable through queries, or documents that are generated as a result of user behavior, such as comments to published news articles or blogs. Monitoring web activity, for instance discussions in web forums, involves extracting text from a huge amount of pages, as the size of the hidden web is estimated to be a lot larger than the visible web [6].

Searching hidden web involves finding web documents that are dynamically generated [3]. Allocating these documents requires web crawlers that can handle crawling both client-side and server-side hidden web. Client-side hidden web techniques involve accessing web content dynamically generated in the client web browser, while server-side solutions focus on the content hidden behind web search forms [22].

Extracting text from the hidden web has proven challenging, as it involves handling a range of technologies and implementing techniques that deals with these technologies

in a fully automatic way [19]. Also, crawling the hidden web is a very complex and effort-demanding task as the page hierarchies grow deeper.

A common approach is to access content and links using interpreters that can execute scripts [19]. Other authors have developed a crawling algorithm based on script analysis [29]. Another approach is to use a *mini-browser* [3] [1]. A mini-browser can interpret source code for specifying a user's navigation sequences, which makes it able to execute scripts and managing redirections etc.

Crawling JavaScript generated content has been identified as the most acute challenge to overcome in order to get full access to the publics free expressions on the Internet. Various approaches have been taken to tackle this:

1. Ignore JavaScript-generated content.

2. Extract clear-text links from inside the scripts.

3. Execute the JavaScript and crawl the generated content.

In projects where only the main content on a web page is required, such as the headlines and articles of a newspaper, alternative 1 might be justified, as this content is often served directly as HTML, in order for it to reach the users' attention quickly. However, we are often more interested in the commentary, such as user-submitted responses to the newspaper article, more than the article in itself. One prominent example of a provider of such commentary is Disqus[10], a commenting service, common to several large newspapers worldwide and also Norwegian papers, such as *dagbladet.no*. One commonly used method for integrating Disqus into existing websites, is to have the Disqus comments loaded into a JavaScript-generated *iframe*.

Alternative 2 has been considered, but is currently not our most promising alternative as it do not work for e.g. Disqus comments included through iframe. The links presented in clear text within the scripts generating the Disqus frame is often not independently browsable. This is believed to be made so intentionally, in an attempt to tightly couple the Disqus comments to the corresponding article. Although there are most likely workarounds to this particular case, it points to a deeper issue; in the general case, predicting the outcome of JavaScript without executing it is an instance of the halting problem, meaning that we do not know the outcome of the JavaScript before we actually run it. A lot of simple scripts can easily be predicted, or proven not to generate new content, but since we have reason to believe that it is the explicit intention of the provider of commentary to make it hard to independently extract the JavaScript-generated content, we have to assume that the scripts used to generate the links, or the content, are intentionally complex. Attempting to predict their behavior would then be considered optimistic.

For these reasons we find that alternative 3 (the actual execution of JavaScript) would be the only route available, in order to make sure we get the full contents of the commentary. This presents a new set of choices;

1. *Bottom-up*: set up a custom-made JavaScript execution environment, such as the cobra toolkit used in [11].

2. *Top-down*: use a full-fledged browser object, such as in [2].

We find that the second alternative is most likely to provide us with the web content that bears the closest resemblance to the content presented to the real users. While Internet

Explorer has been the leading browser for over a decade, according to statcounter[23] it has recently been surpassed by the WebKit-based Google chrome, which now has a market share of over 40%. Together with the 7.9% market share of Safari, WebKit-based browsers now generates close to 50% of the worlds Internet requests, making WebKit the most used browser engine in the world. The fact that WebKit is an open source library, with strong emphasis on JavaScript execution efficiency, makes a WebKit-based solution seem ideal for the purposes of retrieving true representations of Internet content.

3 Web crawling using WebKit and Qt

Qt[20] is a leading cross platform development framework for GUI-applications. The project has close ties with the KDE project[2], from which KHTML originated, of which WebKit is a fork. The Qt project maintains a port of WebKit, wrapped in a convenience class *QWebView*, designed to integrate seamlessly with other Qt components. Creating a WebKit-based browser in Qt can be done with very few lines of code; it is simply a matter of instantiating the QWebView class and adding an address bar and a few buttons. All kinds of browsing activities can then be mechanized by interfacing with the QWebView instance using C++.

A key feature of Qt is the addition of *slots* and *signals*, which enable asynchronous and distributed communication between objects. The slots and signals are introduced as additional keywords atop standard C++, together with facilities for *emitting* a signal, which in turn will trigger an asynchronous call to each *connected* slot. This makes Qt well equipped to deal with the asynchronous nature of AJAX page handling. Also, prior experience with these facilities[7] indicate that they scale very well across many CPU cores, while yielding excellent performance.

A crawler prototype based on QWebView

The developed web crawler is written in C++, using the Qt framework and the QWebView version of WebKit. The aim is not to develop a full fledged crawler, but rather the spearhead of one, proving the frameworks ability to extract web content as plain text, before and after triggering JavaScript through emulating various user events. Figure 1 shows the overall design of the prototype, the main functionality being the extraction of web content as plain text, before and after triggering and running JavaScript. In order to yield an as truthful rendition of the web content as possible, very few changes were made to the default WebKit object. The loading of images was disabled to increase load time, and the visibility of the generated web content was turned off by default, so as to not spend time and resources on rendering.

JavaScript triggered by scrolling

Preliminary manual experimentation showed that a lot of discussion threads attached to newspaper articles are generated by JavaScript, and the content is only revealed after the user has scrolled down to the bottom of the article. It is straightforward to make QWebView scroll to any desired position by calling the `void scroll(int x, int y)` method of the frame object generated by QWebView. However, simply scrolling to the bottom of the page, turned out to often not generate the content, as the scrolling then happened too fast for the script to trigger. A reliable solution was to scroll incrementally,

[2]KDE is a desktop environment for Linux, developed in Qt

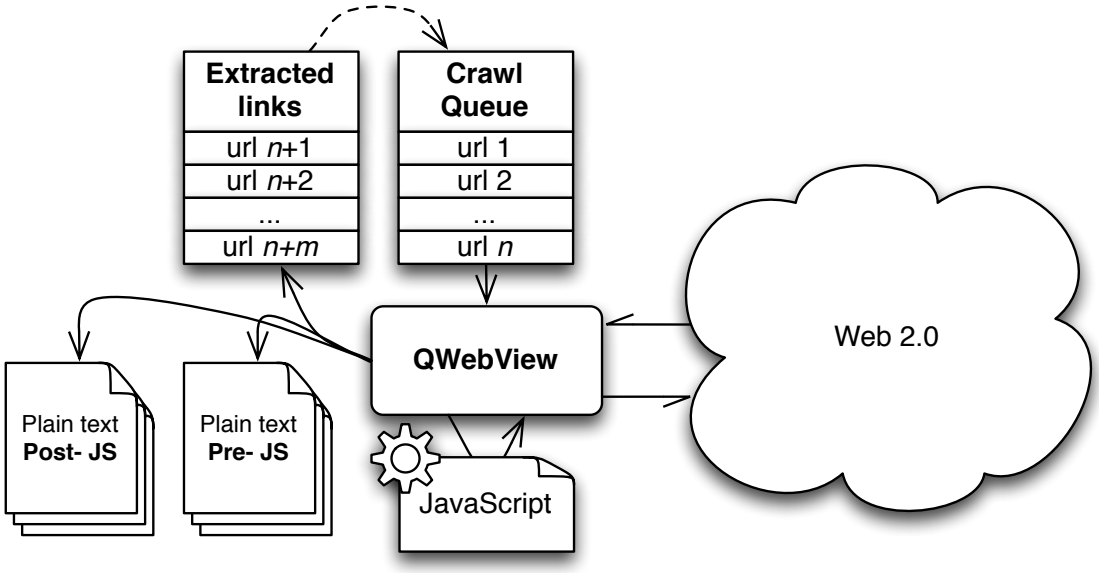

Figure 1: *The design of the QWebView based crawler prototype. The seed URL's are loaded to the crawling queue from file. Each URL is first fetched directly, and plain text and links are extracted. In step 2, scrolling is executed to trigger JavaScript, before plain text content is extracted again.*

in four steps, with a pause of 5 ms. between each step. This had the effect of generating between 2 and 17 new HTML-frames for the URL's in the test set, each containing a fairly large amount of information, including discussion threads and in some cases advertisements.

4 Performance measures for the WebKit-based web crawler

In this Section we evaluate the performance of the WebKit crawler for webpages where discussions is included through JavaScript from the external website Disqus. As described in Section 2 such pages are particularly challenging since the web content is only revealed after user activities (scroll down) and the text content never will appear in the source file.

An experiment was executed on a test set of 39 URL's from the online newspapers *NRK Ytring*, *Dagbladet*, *Aftenposten* and *Bergens Tidene*, each pointing to an article with an attached Disqus JavaScript discussion thread. The experiment was done on a MacBook Air, with a 2GHz Intel Core i7 processor, running OSX version 10.8.4, on a home WiFi network with a ADSL broadband connection. The extraction was fully automated, with a timeout after 5 seconds of inactivity for each URL.

To evaluate the reliability of the web crawler, we collected the text from each URL three times. The end results were similar after each text extraction. Next we compared the collection speeds to extract plain text content both before and after running the scroll-triggered JavaScript. Since we ran all URL's three times we have a total of $39 \cdot 3 = 117$ collections of content before and after the triggered JavaScript. Figure 2 shows the distribution of collection speeds in kB per second for each of these collections. Table 1 summarizes the main properties of the distributions in Figure 2. We see that collection speed is on average about 4 times faster for the non-JavaScript content compared to the JavaScript content. We also observe that to be able to automatically collect the JavaScript content in a robust way, following the scrolling strategy described in the previous Section,

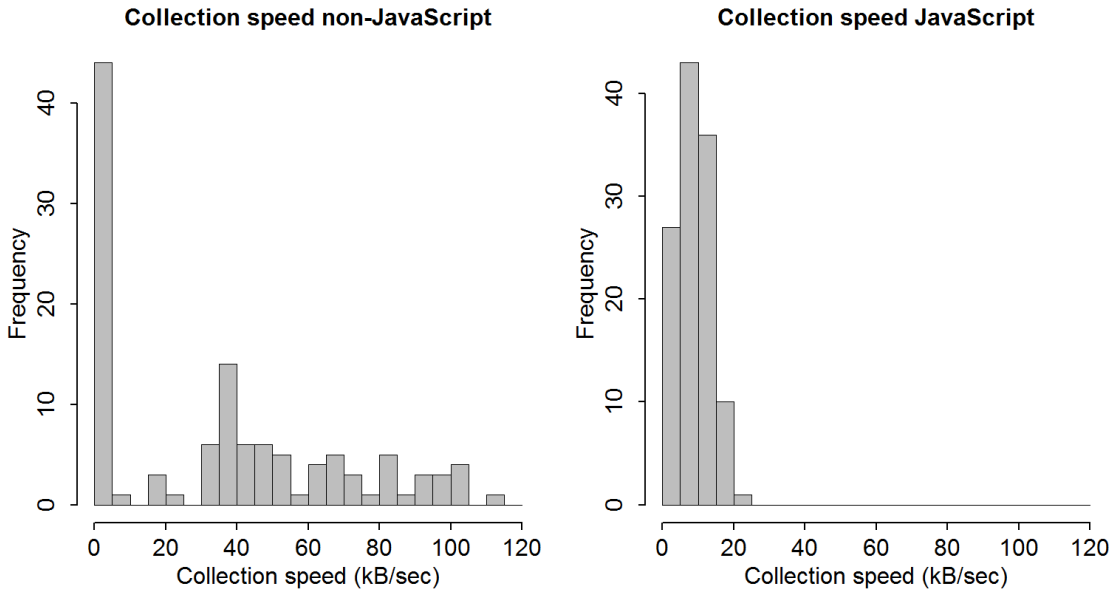

Figure 2: Collection speeds (kB/second) for the non–JavaScript and JavaScript content.

Table 1: Mean, median and quantiles for the collection speed distributions (kB/sec).

	average	median	10% quantile	90% quantile
non-JavaScript	36.3	36.5	1.8	83.7
JavaScript	8.8	8.7	2.1	14.8

sets an upper limit for the collection speed of about 20 kB/sec. One might say that this is not very fast, but if higher collection efficiency is needed the problem is easily parallelizable.

5 Comparison of opinions in debate articles and the subsequent discussions

The 39 articles from the previous Section are all related to some of the themes “Immigration”, “Muslims” and “Islam”. In this Section we want to compare the opinions in the debate articles and in the subsequent discussion for these themes. Our hypothesis is that the debate articles are more positive to these themes than the subsequent discussion.

Opinion mining is performed following the method described in [16], Chapters 5.1 – 5.3, which has proven to work quite well. The method requires a list of words with sentiment and for each word a score is given for the amount of sentiment. Words like ’success’, ’improvement’ and ’happy’ would have positive sentiment scores while words like ’catastrophe’, ’smelly’ and ’motherfucker’ would have negative sentiment scores. There exists a few sentiment lists in English with AFINN being the most used [17]. Unfortunately, no sentiment list exists for the Norwegian language and therefore we developed our own by manually translating the AFINN list. We also added a few words important to the themes in this study like ’deport’ and ’expel’, ending up with a list of 3939 Norwegian sentiment words.

Suppose we want to measure the opinions towards ’immigration’ in a document. The method is then as follows

1. For each sentence containing the word ’immigration’, locate all words in the

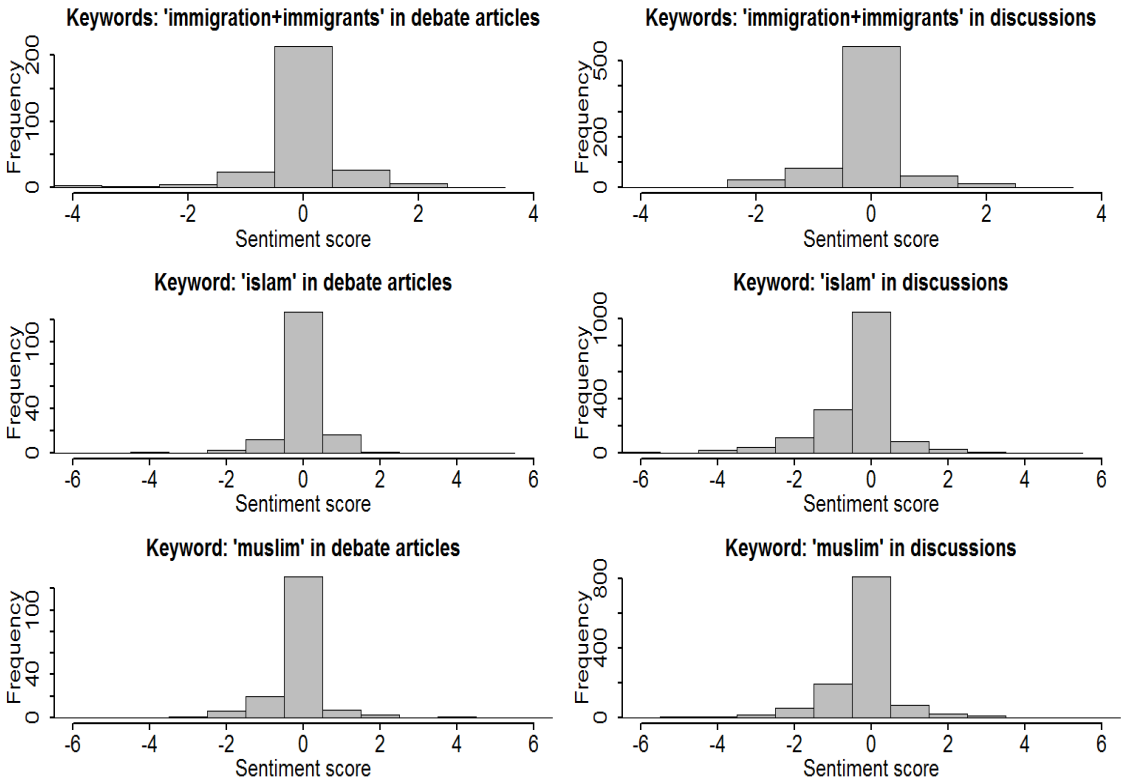

Figure 3: The figures shows the distribution of opinions towards immigration, Islam and Muslim in debate articles and subsequent discussions.

sentence with sentiment (words that is in the sentiment list).

2. Compute the sentiment towards immigration in the sentence using the formula

$$\text{sentence sentiment score} = \sum_{i=1}^{n} \frac{s(w_i)}{\text{dist}(w_i, k)} \tag{1}$$

where k refers to the keyword 'immigration' and w_i the i'th word in the sentence with sentiment. The functions $s(w_i)$ is the sentiment score of word w_i and $\text{dist}(w_i, k)$ the number of words between w_i and k in the sentence. The sentiment score function $s(w_i)$ is equal to the sentiment score of w_i in the sentiment list, except if it is next to a so called sentiment shift word like not, never, none, nobody, nowhere, neither and cannot. Such words may change the sentiment, like for example that 'happy' have a positive sentiment while 'not happy' have a negative sentiment.

The computed sentence sentiment scores for all the sentences gives a representation of all the sentiments toward 'immigration' in the document. The opinion mining method is now run through all the debate articles and subsequent discussions. Figure 3 shows the distribution of opinions toward immigration, Islam and Muslims in debate articles and the subsequent discussions. We see that most sentences have a sentiment close to zero which means no strong sentiment, either positive or negative. We also observe more sentences with negative sentiment towards Islam and Muslims in the discussions compared to in the debate articles.

Next we do a statistical analysis to investigate if the observation above, that the discussions are more negative to Islam and Muslims, are in fact statistically significant. It

Theme	Parameter	Estimate	Std. error	t-value	p-value	
Immigration	deb/disc	0.058	0.050	1.166	0.2426	
Islam	deb/disc	0.342	0.073	4.632	$< 10^{-4}$	***
Muslim	deb/disc	0.142	0.072	1.969	0.048	*

Table 2: Estimates of the parameter in 'deb/disc' for the three regression analyzes related to the themes immigration, Islam and Muslims. Significance codes: * p-value < 0.05, ** p-value < 0.01, *** p-value < 0.001.

is natural to expect that the sentiment score of a sentence depends on who was the author the sentence and from which debate article with discussion the sentence originated. Said in another way, some authors are more negative than others and some debate articles create more reactions than others. Thus we model the sentence sentiment score with the following regression model

$$\text{sentence sentiment score} = \mu + \text{deb/disc} + \text{art.nr} + \text{author} + \varepsilon \tag{2}$$

where 'deb/disc' refers to whether a sentence is from a debate article or a discussion. The variable takes the value 1 for sentences from debate articles and 0 for sentences from discussions. The variable 'art.nr' refers to the different debate articles (with discussions) and 'author' refers to the different authors. Finally ε is random error independent for each sentence. Our main interest is on the variable 'deb/disc' but the two other variables must be included in the model so that the uncertainty in the data is not underestimated. Since we have as much as 39 different articles (with discussions) and 158 different authors, it is natural to model the two variables, 'art.nr' and 'author', as random effects, resulting in a linear mixed regression model [13]. The parameters in the regression model are estimated using the function `lmer` in the statistical software R [21] [5]. We fit one regression model to each of the three themes. Table 2 shows the estimates of the parameter 'deb/disc' for the three regression models. We see that the parameter 'deb/disc' is positive for all the three themes indicating that the debate articles are more positive to these themes than to the discussions, but note that the result is only significant for the themes Islam and Muslims.

In the collected discussions 9% of the discussion posts where removed by moderators. It is reason to believe that the differences between the debate articles and the discussions would be even greater if it would be possible to also include these posts in the analysis.

The analysis documents more negative opinions toward Islam and Muslims in online discussions compared to the related debate articles. With regard to the concerns about hate speech, the analysis shows the discussions are unique sources of information, showing the importance of being able to crawl and analyze the content of such JavaScript discussions.

6 Closing remarks

The most common social platforms like Facebook and Disqus are so called JavaScript based client-side hidden websites (CSHW) which makes it difficult to automatically extract web page text content. Web developers do not regard this information as the most important and therefore is not designed to be easy accessible to web crawlers and search engines. However the interest for such web discussions is increasing because of its potential to get thorough information about people's feelings and opinions.

We expect that the usage of JacaScript CSHW will increase rapidly in the coming years, promoting an increased demand to efficiently crawl and analyze such web pages. We have demonstrated how such pages can be crawled using WebKit to render the web

pages and emulate human web page activity. The crawler also works for pages where the JavaScript hidden content is included from another website. The approach works well for most sites we tested, but we believe that more robust crawling can be achieved by further emulation of human web page behavior.

Opinion mining document differences in opinions in debate articles and in the subsequent discussion. The debate articles are mostly written by politicians, journalists and academics while the discussions represent the opinions to ordinary people. In our further work we want to use our developed WebKit crawler to collect larger amounts of online discussions to investigate how opinions are spread and if peoples become more hateful or change opinions after long time participation in online discussions.

References

[1] Manuel Álvarez, Alberto Pan, Juan Raposo, and Justo Hidalgo. Crawling web pages with support for client-side dynamism. In *Advances in Web-Age Information Management*, pages 252–262. Springer, 2006.

[2] Manuel Álvarez, Alberto Pan, Juan Raposo, and Justo Hidalgo. Crawling Web Pages with Support for Client-Side Dynamism. In Jeffrey Yu, Masaru Kitsuregawa, and Hong Leong, editors, *Advances in Web-Age Information Management*, volume 4016 of *Lecture Notes in Computer Science*, chapter 22, pages 252–262. Springer Berlin / Heidelberg, 2006.

[3] Manuel Alvarez, Alberto Pan, Juan Raposo, and Angel Vina. Client-side deep web data extraction. In *E-Commerce Technology for Dynamic E-Business, 2004. IEEE International Conference on*, pages 158–161. IEEE, 2004.

[4] Jamie Bartlett, Jonathan Birdwell, and Mark Littler. The rise of populism in Europe can be traced through online behaviour... Demos, `http://www.demos.co.uk/files/Demos_OSIPOP_Book-web_03.pdf?1320601634`, 2013. [Online; accessed 21-May-2013].

[5] Douglas Bates, Martin Maechler, and Ben Bolker. *lme4: Linear mixed-effects models using S4 classes*, 2013. R package version 0.999999-2.

[6] Michael K Bergman. The deep web: Surfacing hidden value. *journal of electronic publishing*, 7(1):07–01, 2001.

[7] Alfred Bratterud. Micromanage: A testing and simulation framework for large populations of virtual machines. *ACM Symposium of Cloud Computing 2013 (Notification pending)*, June 2013.

[8] IBM Corporate. `http://www-01.ibm.com/software/analytics/solutions/customer-analytics/social-media-analytics/`. [Online; accessed 25-June-2013].

[9] S. Dhenakaran and K. Thirugnana Sambanthan. Web crawler – An overview. *International Journal of Computer Science and Communication*, 2(1):265–267, 2011.

[10] Disqus. `www.disqus.com`, 2013.

[11] Cristian Duda, Gianni Frey, Donald Kossmann, and Chong Zhou. Ajaxsearch: crawling, indexing and searching web 2.0 applications. *Proc. VLDB Endow.*, 1(2):1440–1443, August 2008.

[12] Liz Fekete. Pedlars of hate: The violent impact of the European far Right. Institute of Race Relations, `http://www.irr.org.uk/wp-content/uploads/2012/06/PedlarsofHate.pdf`, 2013. [Online; accessed 21-May-2013].

[13] Andrzej Galecki and Tomasz Burzykowski. *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*. Springer, 2013.

[14] Matthew Goodwin, Vidhya Ramalingam, and Rachel Briggs. The New Radical Right: Violent and Non-Violent Movements in Europe. Institute for Strategic Dialogue, `http://www.strategicdialogue.org/ISD%20Far%20Right%20Feb2012.pdf`, 2013. [Online; accessed 21-May-2013].

[15] SAS Institute. `http://www.sas.com/software/customer-intelligence/social-media-analytics.html`. [Online; accessed 25-June-2013].

[16] B. Liu. *Sentiment Analysis and Opinion Mining*. Morgan and Claypool Publishers, 2012.

[17] Finn Årup Nielsen. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. *CoRR*, abs/1103.2903, 2011.

[18] Víctor M. Prieto, Manuel Álvarez, Rafael López-García, and Fidel Cacheda. A scale for crawler effectiveness on the client-side hidden web. *Comput. Sci. Inf. Syst.*, 9(2):561–583, 2012.

[19] Víctor M Prieto, Manuel Alvarez, Rafael López-García, and Fidel Cacheda. A scale for crawler effectiveness on the client-side hidden web. *Computer Science and Information Systems*, 9(2):561–583, 2012.

[20] Qt Project. `http://qt-project.org/`, 2013. [Online; accessed 24-July-2013].

[21] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[22] DK Sharma and AK Sharma. A qiiiep based domain specific hidden web crawler. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pages 224–227. ACM, 2011.

[23] StatCounter Global Stats. `http://gs.statcounter.com`, 2013. [Online; accessed 23-July-2013].

[24] Øyvind Strømmen. *The Dark Net. On Right-Wing Extremism, Counter-Jihadism and Terror in Europe*. Cappelen Damm, 2012.

[25] Inger Marie Sunde. Preventing radicalization and violent extremism on the Internet (Norwegian). The Norwegian Police University College 2013:1.

[26] Hannah Taylor. Social Media for Social Change. Using the Internet to Tackle Intolerance. Institute for Strategic Dialogue, http://tsforum.event123.no/UD/rehc2013/pop.cfm?FuseAction=Doc&pAction=View&pDocumentId=46414, 2013. [Online; accessed 21-May-2013].

[27] WebKit. https://www.webkit.org/. [Online; accessed 25-June-2013].

[28] Robin Wilson and Paul Hainsworth. Far-right Parties and discourse in Europe: A challenge for our times. European network against racism, http://cms.horus.be/files/99935/MediaArchive/publications/20060_Publication_Far_right_EN_LR.pdf, 2013. [Online; accessed 21-May-2013].

[29] Zhang Yao, Wang Daling, Feng Shi, Zhang Yifei, and Leng Fangling. An approach for crawling dynamic webpages based on script language analysis. In *Web Information Systems and Applications Conference (WISA), 2012 Ninth*, pages 35–38. IEEE, 2012.