TECHNISCHE
UNIVERSITÄT
DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair of Privacy and Data Security

CrowdFilter: Client-Side Classification of Web Content

Großer Beleg

Dominik Pataky

Tuesday 19th June, 2018

Author	Dominik Pataky dominik.pataky@tu-dresden.de Matrikel 3749148
Supervisor	M.Sc. Clemens Deußer
Professor	Prof. Dr. Thorsten Strufe
License	CC BY-SA 4.0

Contents

The widespread usage of online social networks like Facebook or Twitter introduce new challenges to regulators. Fake News and Hate Speech are subjects of mainstream media headlines and politicians are looking for ways to regulate online discussions. Providers of these platforms are in the need to react - but it is observable that these reactions are some times premature and can introduce additional problems like censorship.

In this project I examined multiple technical approaches with which undesirable content can be classified as such by the users, making it possible to collect crowd sourced classifications to further generate filter lists for automated content classification. Using these lists users could then decide what type of content they would like to filter out, very similar to common ad blocking techniques.

The approaches I chose include the use of a Firefox browser add-on with a back end server and a classification web interface based on independently crawled content from Reddit and 4chan. The exploration of these different implementations stemmed from the observation that creating a tool which reached a high participation rate is challenging. I present the three development iterations the project went through and can conclude that a mixture of all my approaches is the most promising composition for tools which aim to fulfil similar purposes.

1. Introduction and motivation

In 2018, the biggest social media websites – and therefore most of the platforms where interaction between users happen – are in control of very few companies. These companies, notably the US companies Facebook with WhatsApp and Instagram, Google with YouTube, Twitter, Reddit, LinkedIn, the Russian network VKontakte and the Chinese platforms Baidu, Weibo and QZone, have become very powerful exchange points for discussions. As displayed in the table in figure 1 Facebook announced over two billion monthly active people in its network, which would mean it currently reaches around 28% of the world population.

From a technical point of view these platforms are indeed great steps forward and their achievements in the deployment of distributed systems and Cloud interoperability should not be undervalued. Additionally tech companies with their own research laboratories often also publish their research results and thereby contribute to the global research community from inside privately held institutes, for example Facebook Research or Google Research.

Platform	Users
Facebook	2100
YouTube	1500
WhatsApp	1300
Instagram	800
QZone	568
Weibo	376
Twitter	330
LinkedIn	260
Reddit	250
VKontakte	97

Figure 1: Number of users on social media platforms (in millions). Source: statista.com/statistics/272014

But there is another side to all the advancements in creating global social platforms: the use and misuse of the provided communication channels. There is currently an increasingly politicised debate around regulation of user generated content on these platforms and operators will have to expand their software stacks with tools to not only fulfil law enforcement information requests but also to remove content which collides with national laws. The German Netzwerkdurchsetzungsgesetz [Jus] (NetzDG), effective in Germany since October 2017, is one example of such a law. It forces platforms to cooperate and regulate content within a certain time span or the company faces heavy fines. These regulations introduce new challenges to be solved by platform providers. Currently no established regulation infrastructure exists and the development and implementation of such tools carry risks like misuse and censorship.

This debate not only needs proper technical tools but also answers to political and philosophical questions. We will come back to this in the discussion in section 5, followed by a review of the whole project.

To examine possible solutions CrowdFilter consists of multiple concepts with which the control over content stays with the users of platforms. I believe this initial shift away from holding the platforms responsible is a usable way to prevent central authorities from becoming the policy-makers over content – preventing abuse as well as excessive regulatory demands. In the first chapter in section 3 we will have a look at the first approach, a Firefox browser add-on and its back end. The evaluation at the end of this chapter gives a review of this first implementation. The second chapter beginning in section 4 covers the collection of *Comments* and the implementation of a classification web tool. After the technical details I will again evaluate the results and look into statistics generated from the collected data.

1.1. Related projects

In Fall 2016, four students developed the browser extension **„FiB"** [De+] for Google Chrome during a hackathon at Princeton University to combat Fake News on Facebook. The tool identified content on the timeline and used external sources to validate the extracted headlines, displaying the classifications „verified" or „not verified".

In December 2017, the Boston Globe reported about the project **Open Mind** which originated from a hackathon at Yale University [Glo]. Open Mind is also an extension for the Chrome browser and aims to detect Fake News articles and display a warning to the reader, too. According to the article it also suggests alternative articles which talk about the same topic but were deemed a good balance (on the political left-right-spectrum) for the current publication.

At the Chaos Communication Congress 2017, the 34C3, German author Michael Kreil presented his research results **„Social Bots, Fake News und Filterblasen"** [Kre]. His evaluation of an analysis of accounts and user networks on the Twitter platform resulted in the realisation that influential „social bots" are in fact mostly highly active Twitter users. He also criticised methods and research approaches of related works on the topic, calling upon a re-evaluation of published articles.

A collaboration project involving the Universities of **Antwerp and Hildesheim** uses real time analysis algorithms of Tweets to recognise hate speech [Ant]. Their approach is based on classified keywords and resulted in 80% detection accuracy.

Both **Twitter** [Pos] and **Facebook** [Tri] [Exa] rolled out UI features to handle reporting of unwanted content. In general most forum software releases have some kind of „report" functionality, enabling the users to report illegal or unacceptable content from other users of the community.

In April 2018, **Github** announced a new feature called „Minimized comments" which in theory has similar intentions as the CrowdFilter add-on: „While maintainers can edit or delete disruptive comments, they may not feel comfortable doing this, and it doesn't allow the comment author to learn from their mistake. [...] maintainers can now click in the top-right corner to minimize and hide comments [...]".

1.2. Terms and definitions

For reference and to further improve the understanding of terms created during the development of this project I use the following definitions when using the described terms.

WebExtension The add-on API specification which Firefox enforces since version 57.

Collector Back end tool for the browser add-on. Receives the data which was sent by user interaction through the add-on.

Simulation The second stage of this project started after the evaluation of the add-on. Implemented as a website form, the *Simulation* tool offered an easy to use interface with 10 *Comments* to be classified. The *Comments* were aggregated from a collection of crawled user *Content* from 4chan and Reddit.

Content *Content* on websites can be text (e.g. forum contributions, posts on social media) or any multimedia (images, videos, audio files).

Comment Used in the context of chapter 4. A text from social websites like 4chan or Reddit, written by a user of the corresponding platform. Can contain meta data like a timestamp and identifier.

Submission Input a CrowdFilter user sent to the CrowdFilter web endpoints. Either JSON sent via the browser add-on or the submission of the *Simulation* web form.

Evaluation Labeled *Comments*, a combination of a *Comment* together with either a *Classification* or the label „No classification".

Classification *Classifications* are *Evaluations* which used any of the available classification labels except the label „No classification".

2. Project overview, designs and evaluations

Due to multiple alterations of the original concept for this project I will give a short introductory overview to have a look at all parts of this project. In total, the project includes three iterations of design, implementation and evaluation. Figure 2 contains the iteration details.

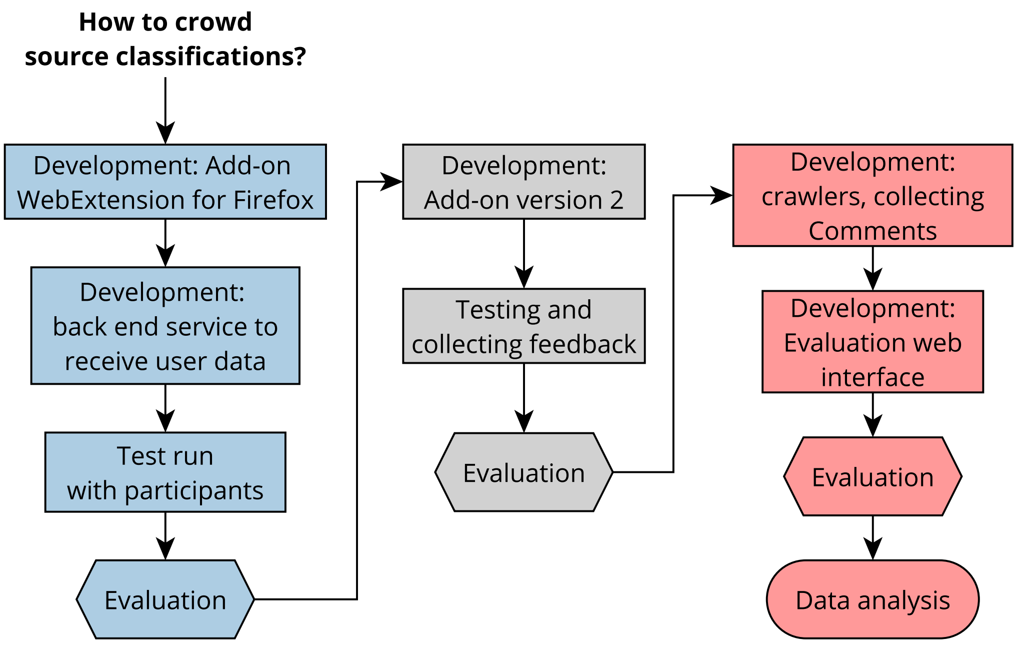

How to crowd source classifications?
Development: Add-on WebExtension for Firefox
Development: back end service to receive user data
Test run with participants
Evaluation
Development: Add-on version 2
Testing and collecting feedback
Evaluation
Development: crawlers, collecting Comments
Development: Evaluation web interface
Evaluation
Data analysis

Figure 2: Overview of the project design, implementation and evaluation flow

The initial question I based the implementation of this project on resulted in the first approach, the add-on. I opted to develop a tool which could be easily integrated into the every day modern browsing experience. For this, I chose Firefox with its WebExtension API. It opens the possibility to integrate all features I aimed for, namely website manipulation, integration into the user interface and submission of user-generated data to an external collector server.

After the first round of testing and participant feedback the evaluation showed that the approach had a chance to succeed but the implementation carried some restraints. I therefore stayed with the browser add-on approach but switched to a more generic implementation, lowering the usage barrier and improving extensibility.

With a second round of evaluation and feedback collection I ended the add-on development and concluded that this approach does provide a useful tool but would need more advertisement and preparation for participation.

Since the central task was to crowd source the classification of text content I decided to design a third implementation, this time not based on the add-on. For this, I used crawlers to populate a database with a collection of *Comments* from the Reddit and 4chan social platforms and combined this pool of *Comments* with a web interface, which was also developed specifically for this project. This third stage resulted in more participation and data, which could be used for a final analysis to have a look into its informative value.

The project review in section 6 continues this section and provides a conclusion of all approaches. We will now continue with section 3 in which I explain the implementation details of the add-on.

3. Firefox browser add-on and API back end

3.1. The WebExtension add-on

The add-on prototype is implemented for Firefox. Since version 57 add-ons in Firefox use the WebExtension Application Programming Interface (API) exclusively, documented in the Mozilla Developer Network (MDN). This API allows finer permissions, better isolation of components and is also implemented in Chrome which allows cross-browser development. The implementation of prototypes of this add-on produced two different approaches. The first version injected a button into the Document Object Model (DOM) of specific sites (`github.com`, `twitter.com` and `heise.de`) which offered a drop down list with classification labels. A refactoring of this version towards a more generic implementation for an unknown set of websites resulted in the second version of the add-on, beginning with v2[1].

A WebExtension consists of multiple files packaged as a ZIP archive. Firefox unpacks this archive, reads the content of the `manifest.json` file and executes the add-on if the manifest contained no errors. The manifest contains meta data consisting of the name of the add-on, version specification, project URL and name of the authors. It also contains a list of permissions and scripts to be executed.

Scripts are either background scripts or content scripts. They differ in their permission set and API features. A content script is injected as an additional script into a page which is running in a browser tab, like every other JavaScript script which is included by the website. Background scripts are executed in their own context, the add-on context, in the background (as scripts in background pages).

Background scripts have access to the storage and can make their own requests to remote resources with Asynchronous JavaScript and XML (AJAX). Since they are not associated with a tab and run as a single instance during the whole browser session, independent of any tab state, they do not have access to any DOM elements loaded in tabs inside the browser.

Content scripts are injected on websites the browser recognises by a pattern, e.g. `//example.com/*`. The content script is running along side all other page scripts in the context of the tab and can manipulate the DOM of the page.

Both script types combined offer the full feature set that is needed to do complex tasks. Connecting content with background scripts is therefore an essential feature, which is also offered. There are two possible ways to implement channels:

1. `runtime.sendMessage` and corresponding handlers when a message appears,
2. `runtime.connect` to open a dedicated channel („port“) between content and background scripts.

If a content script sends a message to a background script, a listener receives this event and executes a function called e.g. `handleMessage(message)`. The content of the passed object `message` can be freely chosen. This implementation for example used a `src` attribute to differentiate between multiple scripts that send messages. In case the message is from an injection script, the handler would look for the `cmd` attribute and decide what to do. In this case, a content script requested a list of currently configured classifiers. Using the `respond` method the response object with a type and content are sent back by the background script.

WebExtensions in Firefox are also able to include a custom options page (accessible from the add-ons list), a toolbar button (with either an action or a popup) and a bundled page that can be opened in a tab. CrowdFilter used all of these features:

1. The customised options page let the user toggle the usage of the TOR hidden service instead of the public internet *Collector* endpoint and also offered a feedback form.
2. A toolbar button with the add-on's icon was displayed on tabs in which the plugin was activated.
3. Clicking on the toolbar button opened a bundled page that showed additional information about the add-on and displayed a list of recently sent JSON data for review by the user (for increased transparency, stored in the browser storage).

The feedback form mentioned in point 1 enabled the users to directly tell me about usage problems and improvement suggestions. The integration of the form inside the options page was not supported by the native API, since the internal configuration pages (add-ons, preferences etc.) in Firefox are not rendered like other DOMs and cannot communicate with the background script. They do however support a very limited interface for styling.

[1] https://github.com/CrowdFilter/crowdfilter-webextension/releases/tag/v2.0.0

The solution was to use the available JavaScript API which listened for the submission of the feedback text form. The triggered event then cleared the submission form and put the entered text into the storage. The background script had a second listener which was triggered by a change in the storage, notifying the background script about a new submission which it should then send to the *Collector* API. Most of the content from the options page was later migrated into the info page, which also resolved the need for this workaround.

Another aspect which produced multiple solutions was the update of URL filter regular expressions and the list of classifications. This was solved with two approaches:

1. Pulling of a remote configuration file hosted on the web server, which was always up-to-date and the add-on would use the new version.
2. Using the Firefox auto-update feature to roll out new patch-level versions (according to SemVer) regularly.

The first approach began with a list of classifications formatted in the JSON, which can be natively used with JavaScript. The CrowdFilter API offered an endpoint which generated the JSON from the internal database. Another endpoint supplied a JSON with the list of URL filter regex strings. Both endpoints were later merged together so the add-on only had to pull one configuration file. I then added a timer which polled the endpoint every 10 minutes (for testing purposes, the timer should wait longer in production environments).

But I observed two crucial disadvantages of this solution: with the small timer value the requests against the configuration API endpoint could very well be used to track the user. Given a 10 minute polling interval one could easily log the user's IP changes and e.g. standby times in which Firefox was not running. The fact that this „feature" is so easy to activate was shocking. The second risk this method introduced was the injection of code parts. Especially the payload consisting of regular expression patterns which were later executed in JavaScript looked like a risk factor. I did not try to exploit this feature.

Instead the remote configuration was replaced with the native browser auto-update for add-ons. Since I used the self-hosted add-on style which does not distribute the add-on XPIs over addons.mozilla.org (AMO) the downloadable files have to be provided on a web server. This can be done by using the `update_url` key inside the manifest. The URL must point to a JSON formatted file which provides URLs to all signed versions of the add-on. If the user has activated the automatic update feature, Firefox polls the URL once a day for updates and installs the latest version if a newer one has been found.

I now had the ability to put the configuration with classifications and URL filters inside the add-on package and reduced the needed communication between the add-ons and the web server drastically.

3.1.1. Version 1: Injecting a drop down button

The first implemented version of the add-on used multiple scripts and site-specific code to add a button into the pages Document Object Model (DOM). It is best to explain the procedure with an example: Tweets.

First, the permission to execute code on `twitter.com` is needed, an example is illustrated in figure 4. Twitter is also a good example to explain the combination of executed content and background scripts. Both are needed to handle AJAX requests on pages which do not reload the whole page when browsing through it but only replace changed elements on pages.

The problem is: even though the address bar changes the displayed URL, the WebExtension filter for content scripts is not matched if you do not make a direct page request by putting the URL into the address bar and pressing enter. To solve this, another background script is needed which intercepts requests on certain websites to look for AJAX requests that need code injection.

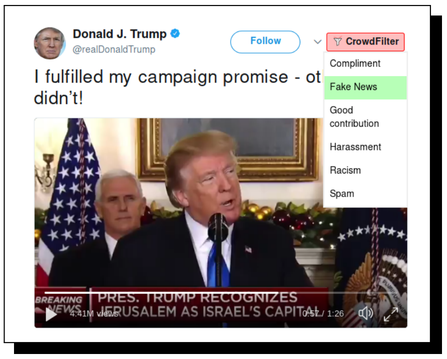

Donald J. Trump
@realDonaldTrump
I fulfilled my campaign promise - ot
didn't!

Figure 3: Injection of the drop down button on Twitter

As displayed in the example in figure 4, if the user visits any page on `twitter.com` the content script is injected by the add-on, after executing all scripts that were requested by `twitter.com` itself. This script is needed for later execution of code. While the `cf-injection.js` script is loaded as a content script inside the tab, the background scripts `background.js` and `ajax-detector.js` are loaded when the add-on is loaded into the browser. Triggering the background script is done by adding a listener with `browser.webRequest.onCompleted.addListener`. If the URL which was loaded matches a pattern given as parameter to the function, a callback function is executed when the resource is finished loading. See figure 5 for an example.

To explain the code three steps need to be considered:

1. A request for a new resource is fired, probably as an XHR. After this request is done, the browser notifies the script which listens on `webRequest.onCompleted`.
2. If the URL of the AJAX request is matched in the listeners filter list `urls`, `url_catcher` is executed with the full request object.
3. This request object is again examined, but in more detail - it is matched against regular expressions for very specific cases (here: loading a Tweet via its permalink).

If all these three steps succeed we know for sure that there was a request we want to inject code into. This is done with the `inject` function, which receives the key of the identified regex. Inside the add-on package are multiple injection scripts in `/js/injectors/`, e.g. `twitter.js` as illustrated in figure 6. Coming back to the injected content script, executing `injectButton(...);` uses the injected JavaScript function to insert the button element, specifically positioned inside the DOM of this page.

If you wish to have a further look into the implementation, I wrote about it in detail in [Pat].

3.1.2. Version 2: Using context menu and generic text selection

The first version of the add-on had one significant disadvantage: one had to bundle the element extraction identifiers for every website the add-on should be able to handle. And those identifiers were also subject to change when the website provider rolled out changes on his side. The result was a collection of snippets with identifiers which would have to be updated and extended by publishing new versions of the add-on or would have to be pulled from external sources.

Version 2 of the CrowdFilter add-on introduced a new approach to solve this problem. Since the key feature of the add-on was to send text content from websites paired with a classification, I looked into browser capabilities for an alternative implementation and chose the context menu API[2].

[2] https://developer.mozilla.org/en-US/Add-ons/WebExtensions/user_interface/Context_menu_items

```
"permissions": [
  "activeTab",
  "https://twitter.com/*"
],
"content_scripts": [{
    "matches": [
        "https://twitter.com/*"
    ],
    "js": ["js/cf-injection.js"],
    "run_at": "document_end"
}]
```

Figure 4: Permissions to load the injection script on pages under the domain twitter.com in the Firefox WebExtension manifest

```
const filters = {
    "github": "issues/[0-9]{1,10}\\??",
    "twitter": "/status/[0-9]*(\\?conversation.*)?"
};

function url_catcher(details) {
    let url = details.url;
    let regexp;
    for (const key of Object.keys(filters)) {
        regexp = new RegExp(filters[key], "i");
        if (url.match(regexp) != null) {
            inject(key);
            break;
        }
    };
}

browser.webRequest.onCompleted.addListener(
    url_catcher,
    {  // Filter
        urls: [
            "https://github.com/*/*",
            "https://twitter.com/*"
        ]
    }
);
```

Figure 5: WebRequest event listener used for detecting AJAX page loads and trigger the injection on certain URL patterns, identified by regular expressions

```
var comment_element_id_prefix = null;
var comment_element_classes = ["permalink-tweet", "tweet"];
var injection_element_identifier = ".permalink-header";
var clicked_source = "twitter";

injectButton(injection_element_identifier);
```

Figure 6: Injection script for Twitter, identifying specific DOM elements used in the injection script for element insertion

The two crucial features which made me choose the context menu interface were

1. the native integration into the browser UI, only using the logo icon to enhance recognisability,
2. the availability of the `menus.ContextType` which only displays the context menu when text is selected. This reduces the visibility of the add-on to situations where interaction is possible, avoiding pollution of UI elements.

This way the user simply highlights a text part on the current website, opens the context menu and can classify content with one additional click. Additionally, using this technique to classify content is not only easier it also solves the problem of implementing the classification button for specific websites, it is a website agnostic implementation.

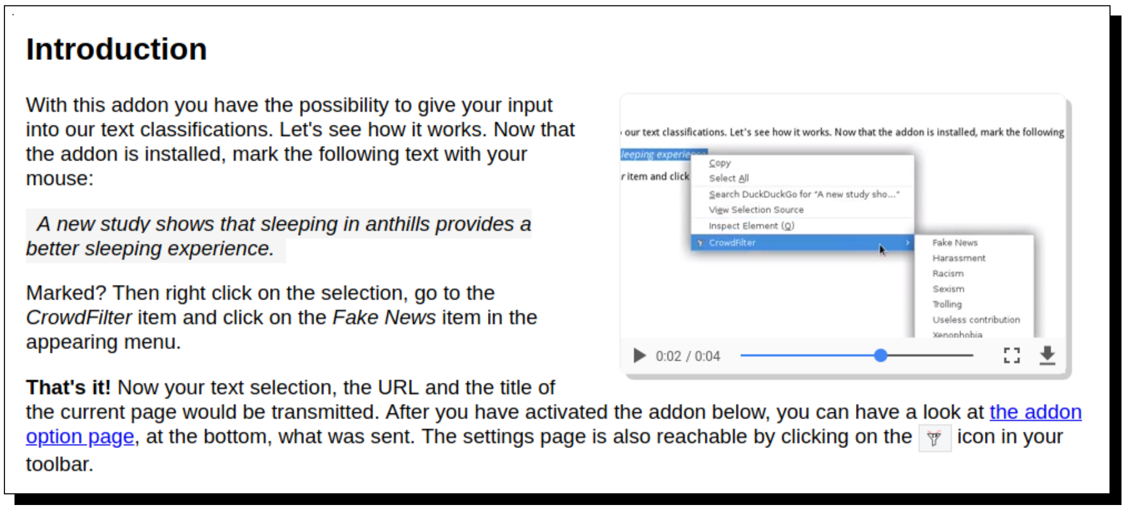
As soon as the user selects a text and clicks on the context menu classification item, the add-on handles the `browser.contextMenus.onClicked` event. The handler receives the event and the ID of the context menu item (set up at the time of creation to uniquely identify the classifications) and sends a JSON with the `original_url`, the `page_title`, the `selection` and of course the `classification` to the *Collector* server.

Another introduced feature which came with this version was the reduced permission set. The add-on did not have to ask for any site permissions except the two server endpoints (HTTPS and TOR) because it as not necessary to inject any scripts into tabs anymore. Furthermore even those two permissions were later removed – leaving to site permissions at all – by properly configuring the Cross-origin resource sharing (CORS) headers in nginx.

Lastly, I decided to finish the development of version 2 with a small usage tutorial. For this a small website was integrated into the add-on package and was loaded when either the add-on was installed the first time or an update of the installed add-on occurred. A screenshot of this page is displayed in figure 7. For privacy reasons I also added an activation button to this page, only enabling the transmission of data to the server when the user clicked on the „*we would like to ask you for your permission to transmit data to our server*“ button. If the user did not click the button and tried to use the context menu on another website, the string `[not activated]` was appended to the CrowdFilter entry.

3.1.3. Features used in both versions

Both versions generated a unique identifier which was stored in the browser's storage. In case the add-on is executed the first time, the background script creates a new ID and stores it. Further executions checked the value inside the storage and remembered the ID. The ID was used as a payload field in all *Submissions* users sent. My original intention was to detect possible misuse, making it possible to correlate *Submissions* and participants with each other. But given the small user base this was not necessary.



Introduction

With this addon you have the possibility to give your input into our text classifications. Let's see how it works. Now that the addon is installed, mark the following text with your mouse:

A new study shows that sleeping in anthills provides a better sleeping experience.

Marked? Then right click on the selection, go to the *CrowdFilter* item and click on the *Fake News* item in the appearing menu.

That's it! Now your text selection, the URL and the title of the current page would be transmitted. After you have activated the addon below, you can have a look at the addon option page, at the bottom, what was sent. The settings page is also reachable by clicking on the icon in your toolbar.

Figure 7: Add-on version 2 showing the introduction page which appears after installation and updates of the add-on

3.2. Collector

The *Collector* is the back end tool which the add-on sends data to. It is a Python application proxied behind an nginx web server listening on ports HTTP (80) and HTTPS (443) for incoming connections. During the run time of the experiment the server was hosted as a virtual machine at the TU Dresden.

Due to data privacy concerns the remote IP address of clients connecting to the web service was removed from any logs. This way the stored Personally Identifiable Information (PII) was reduced on the server. To counter abuse of the API (e.g. denial of service due to flooding with data) the virtual host which handled incoming requests was configured with a rate-limit, guarding the proxied application.

The application itself, written in Python 3 with the Flask web framework and connecting to a PostgreSQL 9 database, was started as a uwsgi instance. The decision to use the combination of Flask, uwsgi and nginx was based on experience from earlier projects, the stack proved to work very well.

3.2.1. HTTPS with LetsEncrypt

LetsEncrypt is a new Certificate Authority (CA) (launched in 2016) which offers HTTPS certificates at no charge. Until the release of LetsEncrypt and the newly developed Automated Certificate Management Environment (ACME) protocol for automated issuing of certificates to clients one had to buy a validated certificate from one of multiple CAs. It was also possible to use the CAcert Web of Trust, with the limitation that the CAcert root certificate was installed in nearly none operating system distribution trust store by default. This leads to verification errors inside the browser when one requests a secure connection to a web resource.

Both problems are solved by LetsEncrypt. The service is free of charge, automated and the root certificate is anchored on most platforms. In February 2018, LetsEncrypt has issued more than 100 million certificates and played a big part in the percentage growth of encrypted websites from around 50% to 70%, according to Firefox Telemetry measurements. New services like wildcard certificates and the improvement of automated issuing will probably push this share even higher. Browsers like Mozilla Firefox[3][4] and Google Chrome[5] integrate methods to not only display insecure connections but also block unencrypted resources completely - resulting in an increased interest in migrating websites to HTTPS.

3.2.2. TOR hidden service

In the context of privacy I offered the feature to send data over the internal Tor network. Users which wished to use the anonymous *Collector* endpoint, a Hidden Service inside the Tor network, could opt-in on the add-ons options page. As soon as the add-on notices a change in the setting, it switched the configured endpoint URL to either the Hidden Service or back to the publicly accessible HTTPS endpoint.

Connecting via a Hidden Service ensures an encrypted, authenticated channel. Every connection is logged from `127.0.0.1` and no other identifiable connection meta data is available to the administrator of the web server (except that one has chosen to use Tor).

[3]https://blog.mozilla.org/security/2017/01/20/communicating-the-dangers-of-non-secure-http/
[4]https://www.ghacks.net/2018/02/14/firefox-60-new-not-secure-indicator-preferences/
[5]https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html

3.3. Evaluation

The implementations presented in this chapter showed to be a less viable solution than anticipated at the beginning of the project.

Version 1 proved to be a helpful tool to collect not only text snippets but also to extract meta data like authors, timestamps of posts and other website-specific elements. The downside was the need for updates when website DOM structures were changed or new websites should have been included as sources into the project.

Version 2 offered the ability to select and classify any text selection on any website and came with a reduced the code size. It was easier to use and was integrated into the native browser UI without interfering into the DOM generation. But it introduced the loss of much of the meta data.

After gathering feedback from a handful of people which were asked about their usage experience the following two improvements were identified:

1. Users need an incentive to participate in the collection of classifications (e.g. they need to experience the „worth" of their own inputs),
2. an effect after submitting their inputs would increase the engagement rate (e.g. the user receives a newly generated filter list after submission and experiences an improvement of filtered content).

Since I was starting with an empty database, there was a very high barrier of collected data to be reached before one would have been able to begin with the creation of filter lists which could have been pushed back to the user.

This brought another difficulty to light: it could be anticipated that asking specific persons – friends, colleagues, people in online forums – to participate would lead to a biased user base, since the chance that people we have around us in our daily life have a very similar set of opinions is very high. To test the option of gathering more participants, I posted the project's website on Reddit and Hacker News and talked about the project in a Lightning Talk at the 34C3. The response was sparse. For testing purposes the small circle of participants showed to be very helpful input, but I concluded that the attractiveness of the add-on must be further improved before any significant variation of inputs can be generated. We will again talk about the project continuation in section 6.

In conclusion the approach of using a browser add-on is a useful tool to provide a submission channel for user-generated classifications, and could possibly also be used as a bi-directional channel to push back filter/classification lists into the browser. But this experiment showed – after implementing two concepts – that such a tool must be designed approachable and that some kind of reward system, e.g. a directly visible gain for the user, is necessary to elevate the participation rate. Research projects which focus on similar crowd sourcing approaches are very welcome to use and extend my implementations.

Please note that the data collected in this part of the project did not suffice for an extended data analysis and evaluation. Have a look in section 4.4 if you are interested in this topic.

4. Classification of collected comments

This chapter picks up the conclusion from the previous approach. In the following I will demonstrate a different procedure implemented to make classification of text snippets more approachable.

For this I first crawled a collection of *Comments* from Reddit and 4chan. Then, having a data set of *Comments* available for classification, I built a web front end which presents a fixed set of *Comments* to participants.

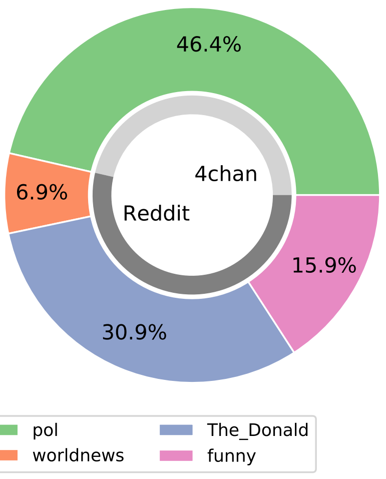
4.1. Crawling Reddit and 4chan

In detail, I chose to crawl the Subreddits */r/The_Donald*, */r/funny* and */r/worldnews* on Reddit and the */pol/*-Board on 4chan. I chose *The_Donald* and */pol/* for their high probability of containing controversial content, and *funny* and *worldnews* to extend the collection with neutral content which was anticipated to balance out the data set.

Both websites offer APIs to access their site's content for external developers.

For the Reddit API, one can use the Python library PRAW (Python Reddit API Wrapper). It uses a registered account to authenticate via OAuth and offers read-only querying of Subreddits to fetch their submissions. The query requested the `hot` category which returns a list of submissions which according to the ranking algorithm have a high chance of being new and voted up at the same time [Sal]. I decided against using the `controversial` sorting because it does not take the posting time into account.

The board contents from 4chan are also offered by a read-only API. The limits are much less strict, there is no need for authentication. As written in the *4chan API Rules* a rate-limiting function was added into the crawler to limit the amount of requests to a maximum of one request per second. By design, 4chan offers very little metadata with its content, since using and posting to the site is possible without any form of registration or authentication.



Source	Comments	Share
4chan /pol/	105.427	46,4%
Reddit	121.707	53,6%
▷ funny	36.008	(15,9%)
▷ the_donald	70.074	(30,9%)
▷ worldnews	15.625	(6,9%)
Total	227.134	

Figure 8: Shares of Comment sources from Reddit and 4chan

Results from queries to the APIs were examined by the crawler script. Only a part of the delivered metadata was stored, together with the content `body` and `source_id`, as displayed in figure 9. Both platforms offer unique identifiers and the creation timestamp of submissions. Additionally Reddit's permalink URL and the current score was stored together with the submission, and the board and country name with 4chan posts. The Reddit permalink enabled finding the submission's Subreddit without having to store this value individually.

Reddit	4chan
identifier	post number
timestamp	timestamp
permalink	board name
score	country

Figure 9: Stored metadata of Comments crawled from Reddit and 4chan

The first crawler implementation used SQLite storages for each source independently. Later on, these databases were migrated into the PostgreSQL DBMS instance which was already set up for the Collector explained in section 3.2, creating a new database to store the data. A migration script read the SQLite storages, connected to PostgreSQL and inserted all stored rows.

Because the schemata of both sources were still split, I later on created a unified schema under the name `comment` which has an `id`, a `source_id` which is a foreign key to the `source` table, the body text content and a `meta` attribute. The `meta` field used the PostgreSQL specific `HSTORE` key-value datatype. This enabled storing metadata dynamically without having to change the table columns later on. The final database schema is pictured in figure 16.

Alembic was used for the upgrades and migrations between database schema changes. Since I also used the SQLAlchemy framework for database interaction in the scripts, Alembic fit very well into the code base.

4.2. Crowd-sourced classification of comments

In this second approach I focused on implementing an easier to use classification interface for participants. Having a collection of *Comments* I created a web front end which displayed a set of ten *Comments* and offered the option to classify each *Comment* from a list of pre-defined classifications.

The front end was again implemented as a Flask web application in Python, connecting to the existing PostgreSQL database. To render the input forms the Flask extension WTForms was used.

As mentioned above a participant is presented a sample set of ten *Comments* randomly collected from the crawled collection. This sample set was a random mixture consisting of three *Comments* from *4chan /pol/* and four *Comments* from *Reddit (the_donald, funny, worldnews)*, the last three being „recycled" *Comments*. Those recycled *Comments* were collected from the pool of *Classifications* from existing sample sets, increasing the chance of multiple *Evaluations* for each existing *Classification*. If not enough *Comments* could be collected, the mixture received more *Comments* from 4chan.

To differentiate participants I decided to use a pseudonymous PIN system which does not require users to authenticate with any personal information, but also enables linking multiple *Evaluations* together by account. Participants who returned after some time could re-use their PIN to continue the evaluation process without having to re-evaluate sample sets they had already seen. Did a user return to a set evaluated earlier the previous *Evaluations* were loaded from the data base and presented in the form. The PINs consisted of four digits – a system with more users would have to increase this range to not only support more users but also to prevent users from randomly reusing a PIN owned by another user.

Logging in with a PIN additionally created a hash of the user's IP address, User-Agent and accepted languages. This fingerprinting technique allowed monitoring any reuse of PINs by different participants. But I never had to take action based on this stored data.

For monitoring of the usage of the ongoing Simulation an admin page which processed the current database entries and generated multiple statistics was added. For example I adjusted the crawler script to save crawling results into the database and could then compare different crawler runs in terms of collected *Comments* and duration of the run to detect any problems. Furthermore the statistics showed how many participants sent how many *Evaluations* and the amount of usages for each classification label.

4.3. Evaluation

The motivation for this implementation was to lower the usage barrier for participants encountered in part one of the project. After looking at the amount of submitted *Evaluations* it is obvious that this channel of input resulted in more usable *Classifications*. In total, 13 participants evaluated at least one sample set. Figure 12 shows the amount of newly created users (each with a unique PIN) by day and the amounts of submitted *Evaluations* by user. With an amount of 260 *Evaluations*, the user with PIN 4343 has submitted around 35% of the total 748 *Evaluations*. The author himself used PIN 7552 in the role of a real participant.

The graph also shows that some users submitted an odd amount of *Evaluations*. This stems from the combined data set from two deployed application versions, one where an *Evaluation* with the label „No classification" was not saved at all and the second version where every submitted sample set generated ten *Evaluation* database entries – including saving the choice of „No classification".

This means the total amount of usages of the „No classification" evaluation label could in fact be higher than extracted. It is also important to note that the admin interface offered the option to enable and disable individual classification labels during the project run – a feature that also influenced the amount of usages. Since this run was experimental to primarily test intended functionalities, an evaluation which focuses on the generation of usable crowd sourced data must run with a fixed set of labels.

When asked about their experiences after using the tool, users told me

1. the interface was simple enough to be easily understood and used,
2. the automated loading of new comments encouraged evaluating multiple sample sets,
3. it could prove difficult to classify content without any context the comment was written in (e.g. making it hard to detect irony and satire),

4. using the labels without further explanation of their meanings was too broad, the interpretations of the presented labels were partly overlapping (e.g. xenophobia and racism seemed hard to be distinguished),
5. positive labels like „funny“ and „acceptable“ would have also be used if available.

4.4. Data analysis

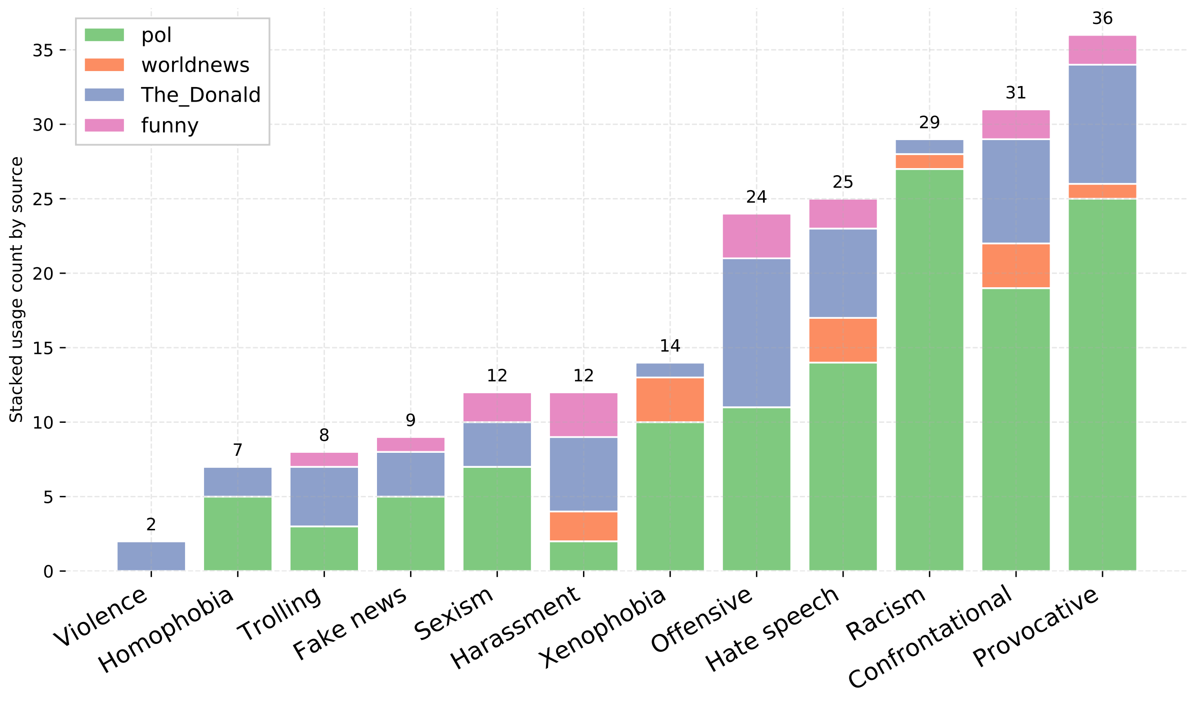
We will now look further into the collected data and generated statistics. As already mentioned there were 748 total evaluations submitted to the CrowdFilter server. 539 of these evaluations, or 72%, were submitted with the „No classification“ label. The rest were distributed over 12 classifications, with „provocative“, „confrontational“ and „racism“ being the most used labels. A detailed graph can be found in figure 10a.

Another aspect which was extracted was the difference between the number of classifications depending on the source of the *Comment*. Figure 10b shows two distributions of sources, one including the evaluation „No classification“ and one without. Comparing the first graph with the sources graph in figure 8 we can see they have similar share distributions – it looks like the mixture of Comments in the generated sample sets mentioned above did work as expected.

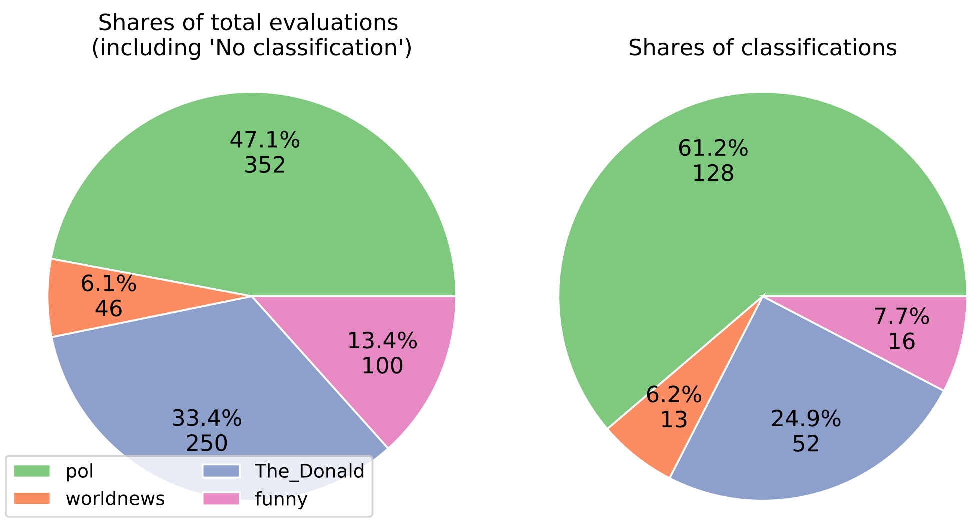
In contrast the shares shift noticeably when only the submitted Classifications are taken into account. While the *worldnews* Subreddit stays at around 6.2%, the *4chan /pol/* board takes a lot of share from *The_Donald* and *funny* – increasing from 47.1% to 61.2%. This could indicate that Comments from 4chan are more suitable to be evaluated by only looking at a single object, compared to Comments from Reddit which uses a more complex threading style within discussion topics – meaning the context between Comments plays a bigger role. But this assumption is solely based on usage experience by the author, the collected data base the analysis is based on is way to small to support this observation.

As mentioned above, the crawled data from 4chan also contains the *country* meta data. In figure 11 the percentage of Evaluations versus Classifications by each country is displayed. Fictional countries users can choose from a list were removed, only keeping those countries which are determined based on the geographic location by the 4chan software in their back end. The analysis shows that e.g. users from the Netherlands and Croatia have a high percentage (75%) of classified Comments in comparison to their total number of Evaluations. Looking at the country with the most Evaluations, the United States, we see that only 46 of 143 Evaluations (32.2%) were Classifications.

Please note that the sample set mixture algorithm did not use the *country* meta data field. The statistic does not compare the total percentage of users in all crawled Comments.



(a) Classification label usages



(b) Classification usage shares by comment source

Figure 10: Statistics regarding the usage of classification labels

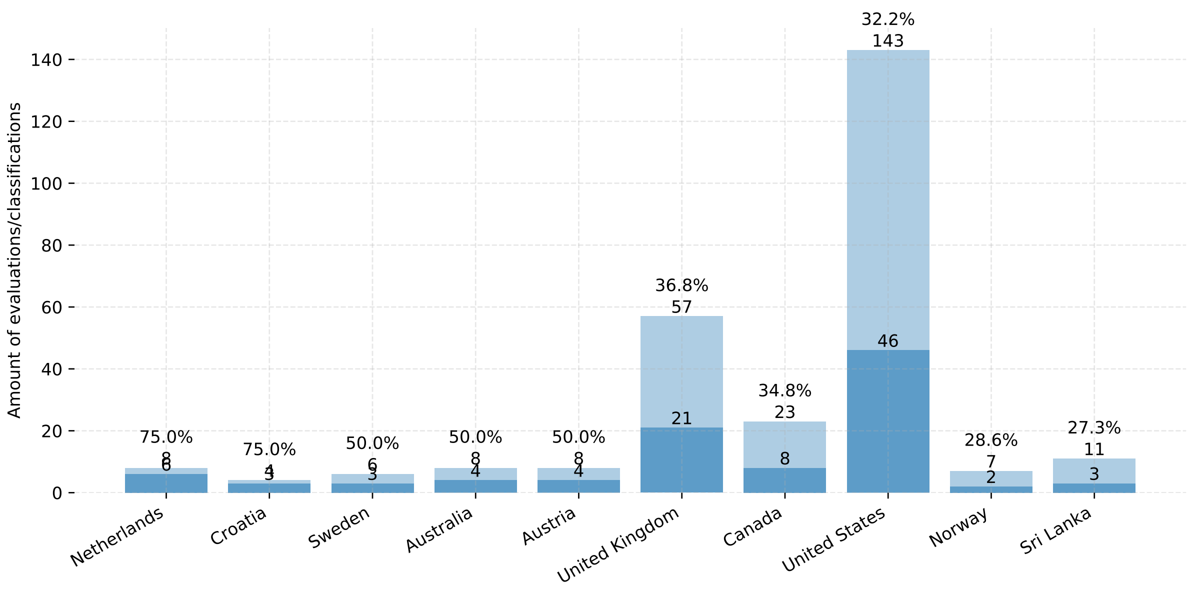


Figure 11: Percentage of Evaluations versus Classifications on 4chan /pol/ by country meta tag

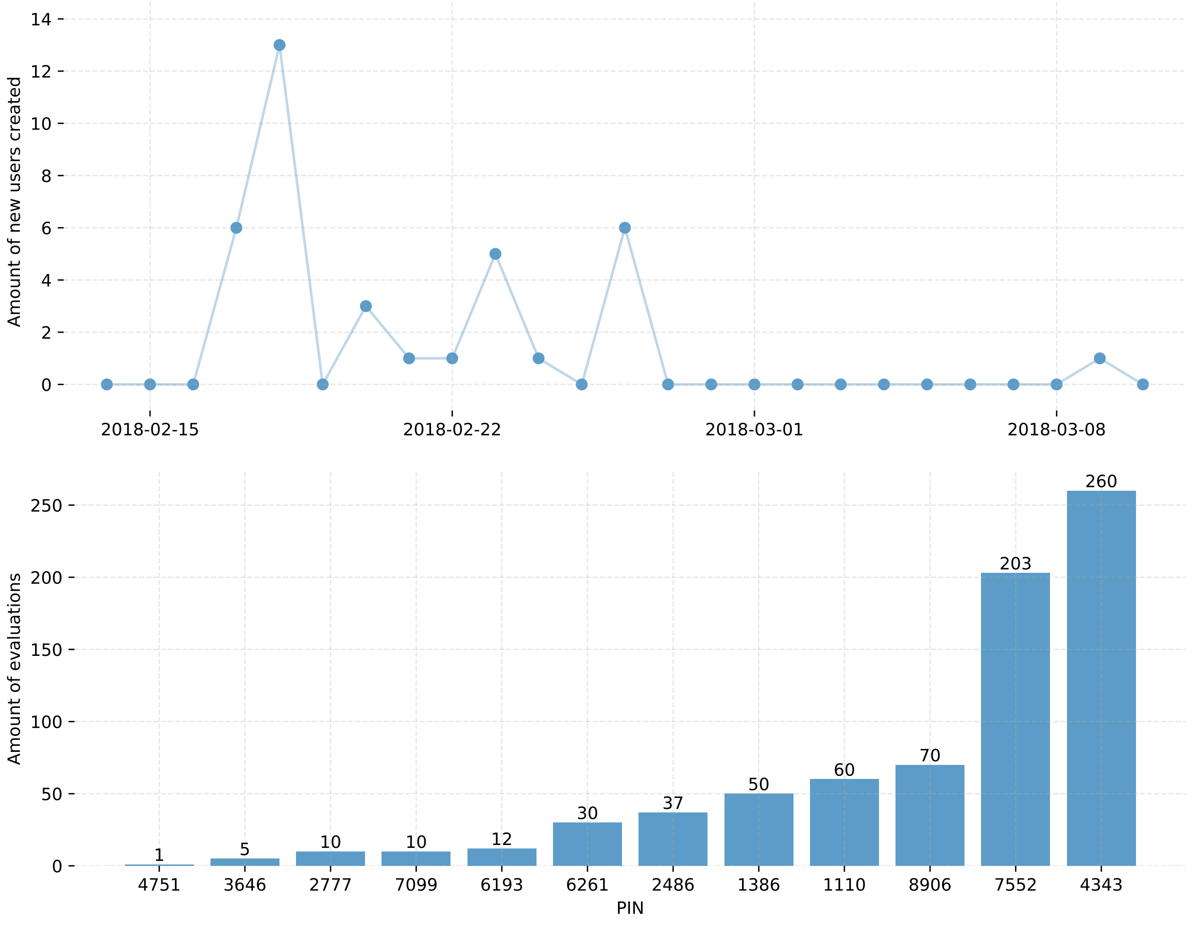


Figure 12: Amount of users created by day and Amount of evaluations by PIN

5. Difficulties of using pre-selected classification labels

In both parts of the project, the add-on and the Evaluation tool, I used a list of classification labels I deemed to fit for this purpose. Some aspects which were taken into consideration:

1. Should positive labels be included as well?
2. Some labels are more general than others – should only the most specific labels be usable?
3. And if yes, should they be visualised in a hierarchical structure?
4. Should the interface provide an input element for custom labels?
5. And if yes, should this input be completely customisable or should it reuse other custom labels?
6. How could biases the users are exposed to be distinguished?
7. Should the user be able to classify a Comment with multiple labels?

I did experiment with the usage of positive labels mentioned in question 1. The labels „Good contribution“ (in contrast to „Bad contribution“ and later „Useless contribution“) and „Compliment“ were available in version 1 of the add-on, but were dropped in version 2. Regarding question 4, the use of custom labels, I implemented this feature in the second part of the project by offering an input text field for extra remarks in the evaluation web form. Question 6 aimed at the possibility that some participant groups might preferably use some labels more than others, e.g. users who react very sensitive to racism might tend to use the label more often.

As mentioned in section 4.3 the received feedback for the add-on very early illustrated the difficulty of mutually shared interpretations between all participants. I therefore later added a list of interpretations I associated with each label and the users could look into it on the add-on settings page. To illustrate this approach, the list looked as follows:

Harassment Content that primarily attacks someone

Trolling Content intended to provoke extreme reactions

Racism Bad content that aims at race, origin or ethnic of the poster

Fake News Content that is provably factually false

Useless contribution Contribution does not add value to the discussion

Sexism Targeting the sex of the attacked person

Xenophobia Attacking a person because of their geographical origin

The list might produce even more questions than answers. Where does one draw the line between harmless banter and harassment? Whats the clear difference between racism and xenophobia? What sources should be agreed on to prove the validity of facts? Which one is the „factually true“ one if sources contradict each other? It is obvious that even with the list of explanations it is still difficult to find a common ground everybody agrees with.

The Evaluation tool did not offer such explanation list but I chose to use more specific labels and remove some of the labels from before. In the end the following labels were activated to be used by users of the web interface: Confrontational, Hate speech, Homophobia, No classification, Offensive, Provocative, Racism, Sexism, Violence and Xenophobia.

The removal of „Fake News“ and „Trolling“ was based on feedback which made clear that those were too hard to identify without having proper context the Comment was written in. Other labels seemed to be easier to use, e.g. if a Comment contained a racial slur it seemed likely the Comment had racist intentions. For label usage statistics have a look at section 4.4.

6. Conclusion

I was looking for a solution to regulate content on platforms without having to force the liability of interpretation onto a single central authority. CrowdFilter was a technical approach to shift these regulatory decisions to a crowd sourced alternative. Looking back, the crowd sourced approach proved to be very difficult. The engagement rate of participants kept low despite multiple occasions of advertisement, resulting in a too narrow data set to continue. Nevertheless, the three iterations CrowdFilter went through - two versions of the add-on and thirdly the web interface with pre-collected Comments - built upon another, based on feedback from real participants.

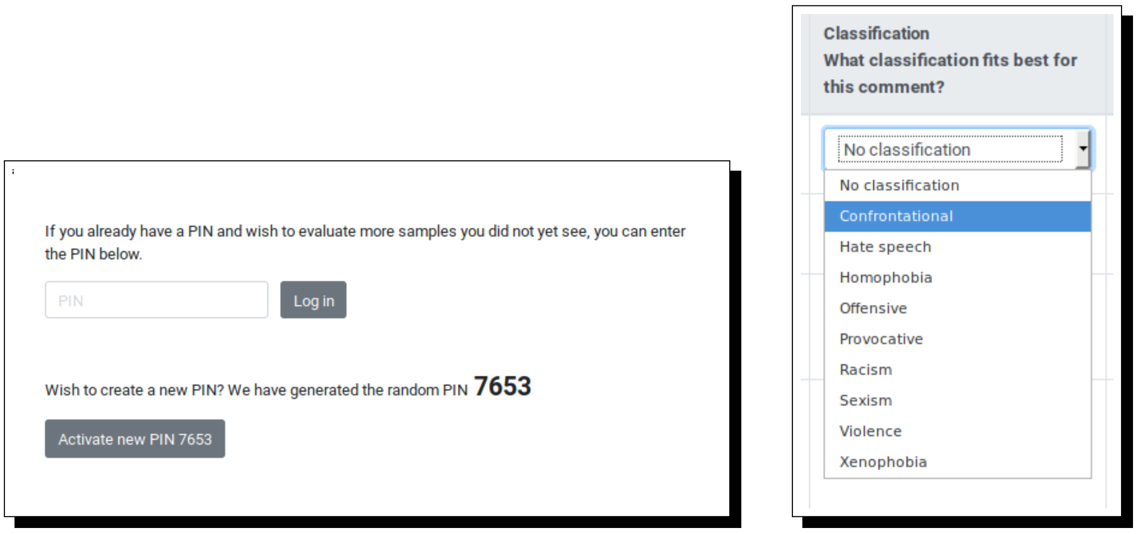
In the end a combination of the add-on bundled with the web interface seems to be a good basis for similar projects. To extend the software, platforms could for example input reports from their websites into the Comment database, making the content available to be evaluated by participants in the evaluation process. The Evaluation tool could further be integrated into existing tools, adding other features to increase involvement (e.g. gamification).

Regarding the continuation of the project from my side, I have taken both the technical and the social issues which surfaced during the project into account. On the technical side a solution to the question above is – given an appropriate data basis – realisable with a collection of tools.

But looking at the social implications tools like CrowdFilter introduce, I strongly caution against seeing software as a solution to social problems like Fake News and hate speech. Software should only serve as support for structuring information and visualising facts – it should not take over the role of moderators regulating human communication.

A. Appendix

A.1. Screenshots



If you already have a PIN and wish to evaluate more samples you did not yet see, you can enter the PIN below.

PIN | Log in

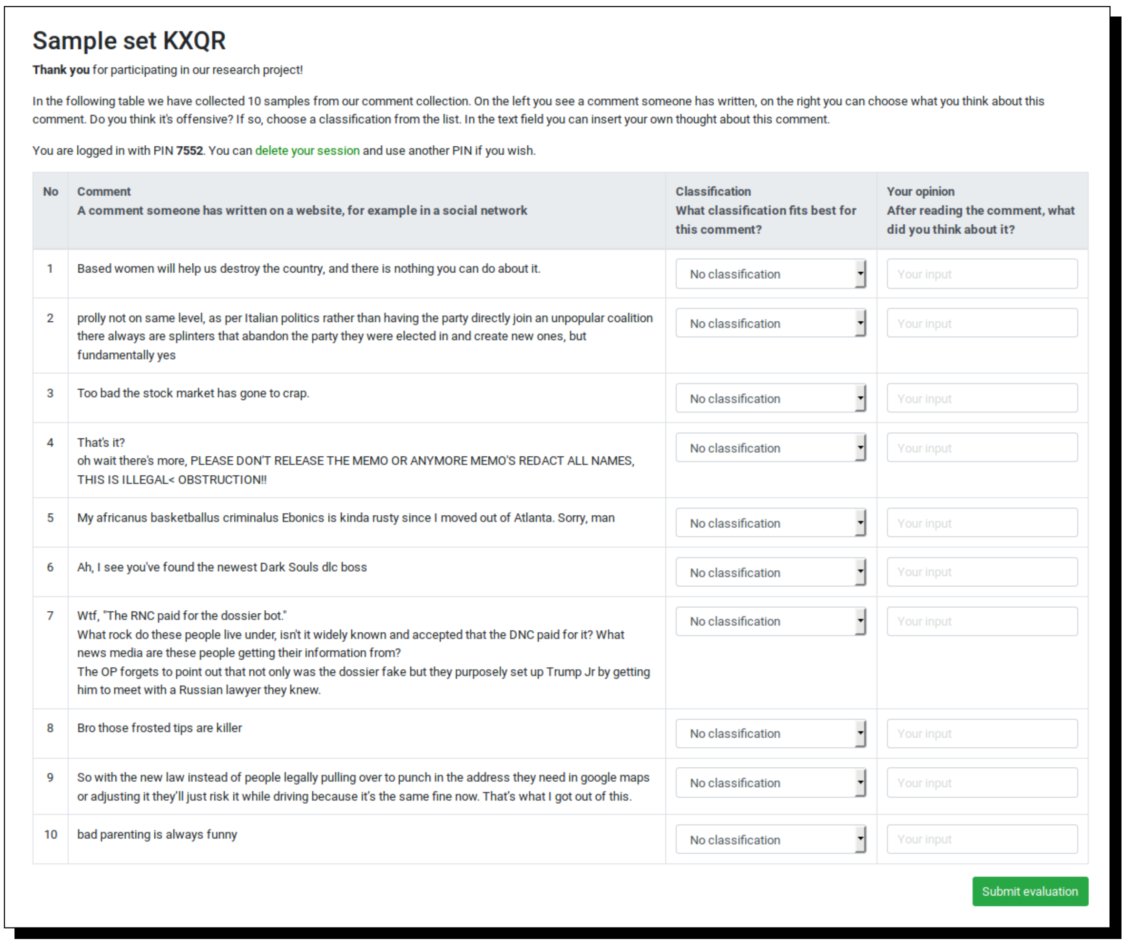
Wish to create a new PIN? We have generated the random PIN **7653**

Activate new PIN 7653

Classification
What classification fits best for this comment?

No classification
- No classification
- Confrontational
- Hate speech
- Homophobia
- Offensive
- Provocative
- Racism
- Sexism
- Violence
- Xenophobia

Figure 13: Screenshots of (a) the PIN login form and (b) the classification drop down



Sample set KXQR

Thank you for participating in our research project!

In the following table we have collected 10 samples from our comment collection. On the left you see a comment someone has written, on the right you can choose what you think about this comment. Do you think it's offensive? If so, choose a classification from the list. In the text field you can insert your own thought about this comment.

You are logged in with PIN **7552**. You can delete your session and use another PIN if you wish.

No	Comment A comment someone has written on a website, for example in a social network	Classification What classification fits best for this comment?	Your opinion After reading the comment, what did you think about it?
1	Based women will help us destroy the country, and there is nothing you can do about it.	No classification	Your input
2	prolly not on same level, as per Italian politics rather than having the party directly join an unpopular coalition there always are splinters that abandon the party they were elected in and create new ones, but fundamentally yes	No classification	Your input
3	Too bad the stock market has gone to crap.	No classification	Your input
4	That's it? oh wait there's more, PLEASE DON'T RELEASE THE MEMO OR ANYMORE MEMO'S REDACT ALL NAMES, THIS IS ILLEGAL< OBSTRUCTION!!	No classification	Your input
5	My africanus basketballus criminalus Ebonics is kinda rusty since I moved out of Atlanta. Sorry, man	No classification	Your input
6	Ah, I see you've found the newest Dark Souls dlc boss	No classification	Your input
7	Wtf, "The RNC paid for the dossier bot." What rock do these people live under, isn't it widely known and accepted that the DNC paid for it? What news media are these people getting their information from? The OP forgets to point out that not only was the dossier fake but they purposely set up Trump Jr by getting him to meet with a Russian lawyer they knew.	No classification	Your input
8	Bro those frosted tips are killer	No classification	Your input
9	So with the new law instead of people legally pulling over to punch in the address they need in google maps or adjusting it they'll just risk it while driving because it's the same fine now. That's what I got out of this.	No classification	Your input
10	bad parenting is always funny	No classification	Your input

Submit evaluation

Figure 14: Evaluation list of comments with classification drop down and comment field

A.2. Database schemata overviews

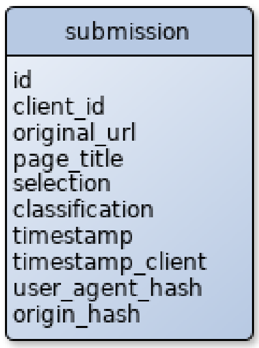


Figure 15: Database schema of the add-on Collector back end

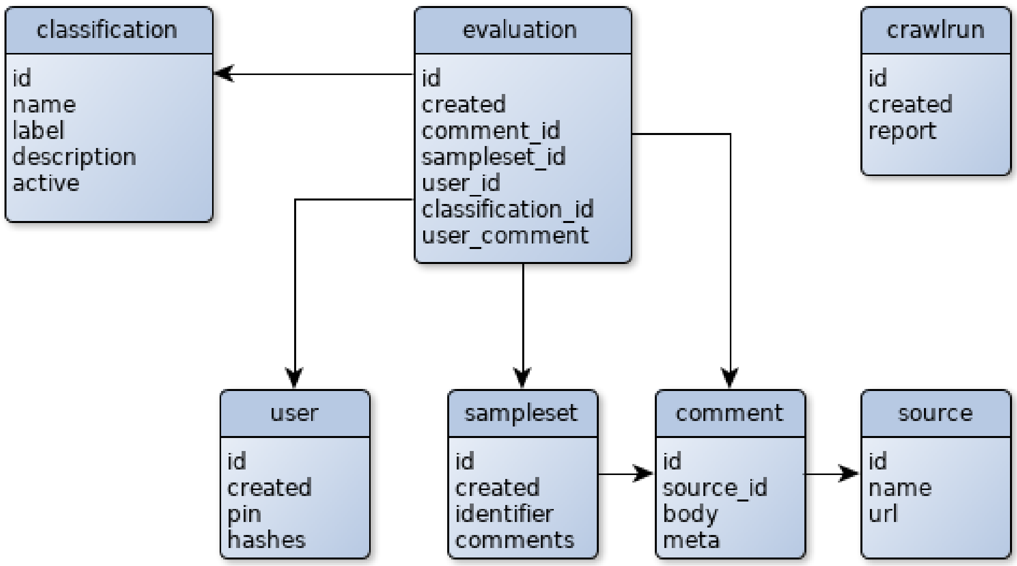


Figure 16: Database schema of the Evaluator back end

References

[Ant] Universiteit Antwerpen. *New technology automatically exposes German hate speech*. URL: https://www.uantwerpen.be/popup/nieuwsonderdeel.aspx?newsitem_id=3216&c=OZEN21178&n=113016 (visited on 22/02/2018).

[De+] Nabanita De et al. *Students Code An Open Source Fix For Facebook's Fake News Problem At A Hackathon*. URL: https://fossbytes.com/fib-chrome-extension-detect-facebook-fake-news/.

[Exa] Irish Examiner. *Facebook warns users about fake news stories*. URL: https://www.irishexaminer.com/ireland/facebook-warns-users-about-fake-news-stories-469409.html.

[Glo] The Boston Globe. *College students come up with plug-in to combat fake news*. URL: https://www.bostonglobe.com/news/politics/2017/12/25/college-students-come-with-plug-combat-fake-news/G0S7FkG2FvKjTwcgQ8wcpM/story.html (visited on 18/01/2018).

[Jus] Bundesministerium für Justiz und für Verbraucherschutz. *Gesetz zur Verbesserung der Rechtsdurchsetzung in sozialen Netzwerken*. URL: https://www.gesetze-im-internet.de/netzdg/index.html.

[Kre] Michael Kreil. *Social Bots, Fake News und Filterblasen*. URL: https://github.com/MichaelKreil/twitter-analysis.

[Moz] Mozilla. *WebExtensions - MDN*. URL: https://developer.mozilla.org/en-US/Add-ons/WebExtensions (visited on 25/11/2017).

[Pat] Dominik Pataky. *Firefox WebExtensions: injecting, sending data and detecting AJAX*. URL: https://bitkeks.eu/blog/2018/01/firefox-webextensions-injecting-sending-data-and-detecting-ajax.html (visited on 16/01/2018).

[Pos] Washington Post. *Twitter is looking for ways to let users flag fake news, offensive content*. URL: https://www.washingtonpost.com/news/the-switch/wp/2017/06/29/twitter-is-looking-for-ways-to-let-users-flag-fake-news/.

[Sal] Amir Salihefendic. *How Reddit ranking algorithms work*. URL: https://medium.com/hacking-and-gonzo/how-reddit-ranking-algorithms-work-ef111e33d0d9 (visited on 18/03/2018).

[Tim] New York Times. *Facebook to Let Users Rank Credibility of News*. URL: https://www.nytimes.com/2018/01/19/technology/facebook-news-feed.html (visited on 19/01/2018).

[Tri] Chicago Tribune. *Facebook combats fake news with new warning label*. URL: http://www.chicagotribune.com/bluesky/technology/ct-facebook-fake-news-warning-label-20170307-story.html.

List of Figures

Acronyms

ACME Automated Certificate Management Environment. 9

AJAX Asynchronous JavaScript and XML. 4

AMO addons.mozilla.org. 5

API Application Programming Interface. 4

CA Certificate Authority. 9

CORS Cross-origin resource sharing. 8

DOM Document Object Model. 4, 6

MDN Mozilla Developer Network. 4

PII Personally Identifiable Information. 9