Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science and Media

Detecting Vandals on Wikipedia Based on User Interaction Logging

Master's Thesis

Kristof Komlossy
Born May 24, 1992 in Leipzig

Matriculation Number 111551

1. Referee: Prof. Dr. Benno Stein

Mentor: Jun.-Prof. Dr. Martin Potthast

Submission date: July 25, 2018

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, July 25, 2018

.....
Kristof Komlossy

Abstract

This work researches the vandalism detection in Wikipedia articles using the behavior of article editors only as an indication for potential vandalism. The first part of this work covers the creation of an appropriate dataset using a crowdsourcing platform. The second part describes the experiments performed on this dataset to investigate a new way of vandalism detection on Wikipedia. The results show that it is possible to distinguish between benign and malicious edits using behavioral hints. If required, it is also possible to detect vandalism without being able to identify editors at the same time. Additionally, I investigated how much time it takes to detect vandalism.

Acknowledgements

I would like to thank my mentor Jun.-Prof. Dr. Martin Potthast from the university of Leipzig, who guided me patiently through this work from the very first and was always eager to provide help and support if needed. Furthermore, did Martin Potthast proofread this thesis and gave a lot of hints on how to improve this document. I would also like to give thanks to Jakob Wagner from the university of Erlangen-Nürnberg for proofreading this document dedicatedly and to Janek Bevendorff from the university of Weimar, assisting me setting up the servers required to perform the crowdsourcing task.

Contents

Chapter 1

Introduction

Wikipedia is the largest and most popular online encyclopedia in the world. The great success of Wikipedia comes from the circumstance that any user can edit a vast majority of articles. Despite the fact that most often a user does not even have to own an account on Wikipedia or provide other verification to make changes to an article, Wikipedia has been established as a source even for schools, yet still trusted with particular caution. But a major problem of Wikipedia is the existence of vandalism in the articles due to the open editing policy. Three examples of how articles are vandalized, also known as textual vandalism, are the insertion of random characters, the harmful alteration of the substance of a sentence, and the simple deletion of text. Vandalism on Wikipedia does not only lower the quality of the affected article but also decreases the reputation of Wikipedia as a whole; therefore, a lot of research deals with the detection of vandalism on Wikipedia articles and will be further examined in the related work section of this thesis.

At the time of writing, all the research on this topic uses techniques dealing with the offline detection of vandalism, meaning the detection of vandalism relies on properties and features of the content of the final text and/or metadata of the text or the corresponding editor. In this work, I am investigating the detection of a vandal on Wikipedia solely based on features created while the user edits an article. Instead of, for example, comparing the edited article with a previous version, I am using features extracted from the user's mouse and keyboard interactions during the editing to analyze the behavior of the user. The detection of vandalism based on the behavior of the user during the editing only could allow for shrinking, if not completely eliminating, the time gap between an article edit and the determination whether the edit was malicious. This could give the ability to perform vandalism detection online, in real time, and to develop new prevention techniques aimed at engaging the vandal.

From this possibility the following research questions arise:

1. Is it possible to detect vandals on Wikipedia based on their behavior while editing?
2. Is it possible to detect vandals without rendering individuals identifiable at the same time?
3. How much time does it take to detect a vandal?

In this thesis, I shed light on these questions for the first time. My contributions in this regard are as follows: I set up a custom version of Wikipedia, which allows users to perform edits on the articles without changing the real Wikipedia. This unique system was newly developed for this thesis and was necessary to not alter the real Wikipedia during our experiments. Subsequently, I designed different tasks and asked users to perform those tasks on my version of the Wikipedia in the context of a crowdsourcing user study and tracked and saved their interactions with the web page. To be able to track the mouse and keyboard interactions of the users I had to develop a reliable tracking framework from scratch due to the lack of such a framework suitable for my use case. As a required step, I had to review every task solved by all the users to maintain some level of result quality and I had to further annotate and classify the results for later experiments. Finally, I created a proof-of-concept prototype to detect vandalism relying on features recorded during the interaction only and performed several experiments to answer the aforementioned research questions.

In Chapter 2 of this thesis, I present some of the existing work related to the two main topics of this thesis, Wikipedia vandalism and user tracking. In Chapter 3 I describe how I prepared and performed the data acquisition to create my corpus containing the recorded interactions of the users to create the proof-of-concept prototype. In Chapter 4 I describe the creation of the machine learning model based on the corpus created in Chapter 3. In Chapter 5 I describe and evaluate the experiments to answer the research questions. In Chapter 6 I give an insight of the implementation into some of the most important modules used in this thesis. In Chapter 7 I give an outlook of future work regarding the detection of vandalism on Wikipedia.

Chapter 2

Related Work

In this chapter, I mention and briefly describe the most important work related to this thesis. This thesis covers two main topics; vandalism on Wikipedia and the detection of such vandalism based on features gathered during the interaction of a user with the Wikipedia web page. Because of this division, the related work chapter is also divided into two parts. The first section covers important work related to vandalism on Wikipedia and its detection in general. The second section covers related work on the topic of tracking users on web pages, logging their interactions, and extracting features from these logs for further use in a machine learning task in particular.

2.1 Wikipedia Vandalism

The topic of vandalism on Wikipedia was investigated by several researchers before this thesis.

Fichman and Hara [2010] published an overview on the topic of trolls on Wikipedia. To analyze the behavior of trolls they had to know the user names of the corresponding Wikipedia users. Since the English version of Wikipedia was too large to identify trolls, they decided to analyze trolls from a small, yet, active Wikipedia community and chose the Hebrew Wikipedia at the end. They contacted twenty-one system administrator from the Hebrew Wikipedia and asked them to participate in an email interview. Finally, they received the answers from eight interviewers and used their reports to identify four troll users in the Hebrew Wikipedia. Fichman and Hara used the gathered information to extract the following types of behavior of trolls and outcomes of trolling:

- Performing of intentional, repetitive and harmful actions
- Violation of Wikipedia policies

- Not limiting the activities on Wikipedia articles but also on the whole Wikipedia community
- Working in isolation under hidden virtual identities with the intent to stay anonymous

The discovered motivations for the described behavior are stated as follows:

- Boredom, attention seeking and revenge
- Fun and entertainment
- Damage tot he community and other people

As the results show, a troll shares properties of a vandal altering a Wikipedia article in a vicious manner.

The usage of a small and active community, like the Hebrew Wikipedia version, as the source for analysis allowed the manual gathering of data by conducting interviews with system administrators. This is most likely not possible to be repeated for all the languages of Wikipedia or even in the English version alone. As a result, most research on detecting vandalism on Wikipedia tried to achieve an automatic approach without the need to manually gather and evaluate data from other people.

The WikiScanner[1] was a tool developed by Virgil Griffith in 2007 and was intended to help tracing back the origin of anonymous edits. The WikiScanner maintained a database containing approximately 34 million anonymous edits combined with the IP address of the corresponding editor. Years later the tool was taken down due to high costs hosting and maintaining the tool with its database.

Two more tools are ClueBot[2] and the STiki-tool by West et al. [2010]. The former is a bot monitoring through Wikipedia, looking for malicious edits, and reverting them directly. The latter is a tool helping editors finding vandalism by listing Wikipedia edits, sorting them by their likeliness to be malicious edits.

Both of the previously mentioned approaches are using a machine learning approach to find vandalism in the edits.

Potthast et al. [2008] defined vandalism detection as a one-class classification detection. By manually analyzing 301 cases of vandalism on Wikipedia, Potthast et al. extracted a feature set which operated on the texts of each revision. In the paper, they trained a classifier with 940 article revisions as input data containing the already mentioned 301 vandalism revisions and compared

[1] `https://en.wikipedia.org/wiki/WikiScanner`
[2] `https://en.wikipedia.org/wiki/User:ClueBot_NG`

the classifier with existing rule-based vandalism detectors and outperforming them.

Subsequently, Potthast et al. [2010] and Potthast and Holfeld [2011] organized a series of competitions on the detection of vandalism. These competitions followed a similar workflow as this thesis to detect vandalism on Wikipedia. The first competition provided a corpus of of more than twenty-eight thousand different Wikipedia articles combined with a total of more than thirty-two thousand edits of these articles collected over the timespan of a week. Approximately seven hundred fifty different workers of Amazon's Mechanical Turk annotated the edits, determining whether an edit is a regular or a vandalism edit, finding 2.391 edits to be vandalism. The second competition used the corpus from the first competition as the training data and a multilingual (English, German and Spanish) newly created Wikipedia corpus with 29.949 edits as test data. The competition participants received a training and a test set of the previously described corpus to develop classifiers based on this data. The test set was used to measure the performance of the classifiers, expressed as the precision, recall, and receiver operating characteristic measures. The different features chosen by the competitors to detect vandalism can be separated into two different types: Features based on the content of the edits and features based on the metadata attached to each edit. The first competition led to the PAN'10 Meta Detector, which combined all vandalism detectors participating in the first competition. It achieved a ROC-AUC (receiver operating characteristic/area under the curve) score of 0.96 and a PR-AUC (precision recall/area under the curve) score of 0.78. The winner of the second competition achieved a ROC-AUC score of 0.95 and a PR-AUC score of 0.82. The results of these competitions show the impressive performance of machine learning techniques used as vandalism detectors and many of the following presented papers support this impression.

Thomas Adler et al. [2010] investigated the detection of vandalism using different types of features compared to the papers presented above. Adler relied on features made available by WikiTrust[3] only, like the reputation of the author or the quality of the made revision as reported by the WikiTrust database. Furthermore, Adler separated the detection of vandalism into the aspect of zero-delay vandalism detection and historical vandalism detection. The former describes the detection of vandalism using only WikiTrust features available directly after the submission of the edit. The latter also includes features calculated using future edits and can be used to process edits made at any time in history. The zero-delay vandalism detection is the more interesting way of detecting vandalism with regard to this work. Although "zero-delay"

[3]"A reputation system for Wikipedia authors and content" `http://wikitrust.soe.ucsc.edu/`

still only means zero-delay after the submission of the edit, some features used by for this aspect can also be acquired before the submission. Examples for these features are the information of the anonymity of the editor, the time interval to the previous revision and the hour of day when the revision was created. Other features like the length of the edit comment or the text trust histogram of the current edit are only available at the end of the edit or after the submission. The classifier using the historical vandalism detection achieved a recall of 0.84, a precision of 0.49, and a ROC-AUC score of 0.93. The zero-delay classifier achieved a recall of 0.77, a precision of 0.37, and a ROC-AUC of 0.9.

Adler et al. [2011] extended their WikiTrust approach and combined these features with different other, at that time, state-of-the-art approaches to detect vandalism. The used approaches were a reputation (WikiTrust), a natural language processing (M. Mola-Velasco [2010]), and a metadata (Stiki) approach. The features were further separated into metadata, text, language, and reputation features. The resulting classifier achieved a PR-AUC score of 0.82 and a ROC-AUC score of 0.97 for the zero-delay detection and a PR-AUC score of 0.85 and a ROC-AUC score of 0.98 for the historical detection.

To show the diversity of different types of features, I would also like to mention the work of Alfonseca et al. [2013]. If present, an infobox is located in the top-right section of an article and holds additional and structured information about the corresponding topic. Alfonseca et al. investigated the vandalism detection in an edit altering this infobox based on information gathered from the changes to this infobox only, resulting in an ROC-AUC score of 0.88.

2.2 User Tracking

The tracking of users operating on a website can have many use cases. The owner of a page could use the information of a tracked user to improve the workflow to fill out a form or to get statistics of the usage of the web page in general. The simple recording of user metadata sent with each request is not sufficient to make such judgments; therefore, the majority of presented papers use some sort of injected script to record, for example, the keyboard and mouse interactions like I did in this thesis.

Pusara and Brodley [2004] investigated the possibility to re-authenticate a user to a system based on the mouse inputs only. The intention was to find an alternative to existing re-authentication methods, like asking to enter the password again after using a system a specific amount of time. Pusara and Brodley recorded many mouse features like the mouse movements, clicks, and mouse wheel interactions of 18 users in a restricted environment and used the majority of the collected data to train a model for each user. The rest of the collected data was later used for evaluating the approach by performing different experiments. The first experiment investigated the difference in behavior between each pair of users and showed that, although some users are quite similar and could be confused with each other, the majority of users are different to each other. In the second experiment Pusara and Brodley investigated the false negative and false positive rate to distinguish one user from others. Pusara and Brodley concluded that the possibility to confuse an invalid user with a valid one is below 3.06% (false positive), but that a valid user was often (27.5%) confused as an invalid one (false negative). The high false positive rate is dependent on the amount of user interactions recorded in a session, influencing the user's model.

Zheng et al. [2011] built upon the work of Pusara and Brodley by increasing the size of the datasets used for their experiments and by generating more sophisticated mouse movement features. In a user authentication experiment, Zheng et al. achieved a false negative rate and a false positiverate of 1.3% both, performing better while requiring less user data than comparable related work.

Plank [2016] examined to what extent keystroke dynamics, meaning the information how long a key was pressed by an author, can be used to perform authorship attribution. Plank performed an experiment with keystroke dynamics of 38 authors and achieved an accuracy of 77% using keystroke dynamics features only. The combination of those features with textual features like word embeddings further improved the accuracy by only 1%.

Subsequently, Plank [2018] investigated the possibility to not only predict authorship but to also detect author traits, in particular age and gender, using keystroke dynamics only. Plank used two different datasets to create a support

vector machine model for each of the experiments. Plank achieved a F1-score of 85.9% and 90.2% for the datasets trying to predict the identity of an author and at least achieved a F1-score of 63.3% and 60.6% for the detection of the gender and the age of the authors, respectively. Additionally, the experiments showed that the results can be improved slightly, at max by 2.2% for author attribution and by 12.67% for age prediction, if the keystrokes dynamics are combined with text-based features.

Khan et al. [2008] combined mouse-click events with keyboard events of users to measure their personality, with the goal to replace long questionnaires usually used for this task. In their first study, they used a custom application installed on the personal computers of twenty users to record the mouse and keyboard interactions over an average timespan of eight days. In the second study, the users had to complete four tutorials regarding the programming language Alice while music was played in the background. Both studies showed that the recorded features and the personality traits of the users were correlated and that the standard deviation of the average time between events give the best feedback about the mood of the user's personality.

Youngmann and Yom-Tov [2018] researched to what extent mouse interaction features can be used to determine the current anxiety level of users of a search engine. Youngmann and Yom-Tov used a corpus consisting of over 22.000 user queries, searching for medical symptoms, and assumed a difference in behavior on the search engine result page for benign indications compared to life-threatening symptoms. To prove this assumption, Youngmann and Yom-Tov extracted a medical severity rank fore each query using the judgments of experts. Subsequently, Youngmann and Yom-Tov created a model to predict the medical severity rank of a search engine user based on the mouse interaction of that user. Finally, Youngmann and Yom-Tov measured the correlation of the predicted medical severity rank and the rank assigned by the experts and achieved an average Kendall rank correlation coefficient of 0.48 and an average Spearman rank correlation coefficient of 0.4.

Savithri et al. [2018] initiated a project at the Wikimedia Foundation to detect spambots registering themselves on the registration page of Wikimedia. Savithri et al. track the mouse and keyboard interactions on the registration page, transform them into features and train a classifier to distinguish real users from spambots. The goal of the project is to replace the existing CAPTCHA on the registration page with an "invisible CAPTCHA" to improve the detection rate and the user experience at the same time, since only users whose prediction score comes out "inconclusive" will be presented with a traditional CAPTCHA. Given the close relation of this idea to ours, and because this technology is dedicated to Wikipedia, the features used in this project will be presented in Chapter 4 since I am using those features in my experiments to compare the

performance with my own features.

The previous papers created promising results but almost all of them, given that the data was not already available, created their tracking data in some kind of local experiment on some prepared workstations. Since I am not able to create a supervised environment using Wikipedia, a different approach has to be used in this thesis.

Atterer et al. [2006] presented their system to track the interactions of a user on any web page and send the data to their server to save it. As in my case, they made use of an HTTP proxy which acts as a connection between the user and the requested server and is therefore able to alter the request of the user and the response of the server. The paper lists and explains several use cases for the system and the benefits (non-intrusiveness and flexibility, for example) compared to a classical web page evaluation like an eye-tracker test. Atterer et al. also elaborate on the topic of implicit interaction, referring to the kind of interactions properties the user is most often not aware of, like typing speed. Since the goal of this thesis is to fetch and analyze these properties, I will dig deeper into this topic in Subsection 3.2.4.

Chapter 3

Data Acquisition

In this chapter, I describe the data needed for answering the research questions and how I acquired this data. The first section elaborates my considerations regarding the data construction. The second section gives an overview over Amazon's Mechanical Turk in general and the type of tasks I used to get the necessary data for my thesis. The third section describes the post-processing of the recorded interactions to create the dataset. The last section presents the characteristics of the resulting dataset used for the experiments.

3.1 Considerations

To investigate this thesis' research questions, I needed a dataset of users editing Wikipedia to perform experiments on this dataset. Although there are several, even official, datasets regarding Wikipedia articles, none of them contained the user interactions during the editing as fine grained as needed by me. Therefore, I quickly came to the conclusion that for this thesis I will have to create my own dataset containing all the necessary interactions of the users. The probably best foundation for this thesis would be a dataset compiled by interactions fetched from the official Wikipedia page directly. For this I would have had to ask the Wikipedia project to allow me to record the interactions of editors, such as their mouse movements and keyboard presses. I disregarded this option since such a request to Wikipedia and the resulting process would require a large amount of time and would be rejected most probably because of privacy issues. On the other hand, the maybe most simple solution would be an experiment on a local machine providing users with a plain text and giving them the task to edit this text.

I defined two requirements regarding the data construction and the resulting dataset itself to maintain some quality of the resulting dataset and thus of the results of this thesis overall:

Large and diverse dataset One of the most important requirements for applying a machine learning approach successfully is a sufficient size of the dataset used to train the classifier. Choosing a local experiment setup would either result in a dataset too small or would require a lot of time and a large number of participants to produce a dataset large and diverse enough to form a good foundation for further investigation. Instead, I decided to perform an distributed online experiment using a crowdsourcing platform. With such a service I am able to gather data quickly from a lot of different users all around the world without requiring them to be at a specific location physically.

Authentic experiment The biggest drawback of not being able to record real editing interactions of real users is that the resulting dataset will be composed of non-representative data. To render the collected data as realistic as possible, I had to design the experiment to resemble the workflow of editing a Wikipedia article as close as possible. To simulate this workflow, I decided to create a copy of Wikipedia with all the sub-pages and functionality the real Wikipedia provides. The detailed explanation and implementation of this approach are described in Chapter 6. The goal of such a copy of the Wikipedia was to try to let the participants forget they are not on the real Wikipedia. My intention was to keep their state of mind as natural as possible with respect to editing the Wikipedia.

3.2 Mechanical Turk

To get a large dataset of mouse and keyboard interactions of users editing Wikipedia articles, I used the crowdsourcing marketplace "Mechanical Turk"[1]. Mechanical Turk is a platform provided by Amazon which allows businesses or similar parties, so called requesters, to create tasks which can be solved by workers. A requester can define a template, so called HIT (Human Intelligence Task), which consists of one or multiple tasks processed by workers. For every HIT, the requester has to determine the desired amount of workers asked to work on the HIT. The association between a worker and a HIT is called an assignment and the workers are getting paid a predetermined amount of money per assignment they submit. Every result of an assignment submitted by a worker has to be reviewed by the corresponding requester to decide whether to approve or reject the result provided by the worker. If the assignment is approved, the worker gets paid the respective amount; if the assignment is rejected, the worker does not get paid but gets a notification by the requester which explains the reasons for the rejection. In case of rejection, the requester

[1] `https://www.mturk.com/`

has the option to resubmit the HIT to Mechanical Turk to get another result for this HIT from a different worker. With Mechanical Turk and similar crowdsourcing platforms it is possible to annotate a large dataset in a relatively short period of time. The main drawbacks are that one has to pay for the resulting data and the limited control over the skill set of the workers. On the other hand, the requester is able to control the quality of the dataset to some extent by reviewing assignment on a sample basis to decide the reject or approve status and by doing some post-processing on the resulting dataset.

3.2.1 HIT Design

As already mentioned in Section 3.1, describing my considerations, the data collected via Mechanical Turk are not as authentic as they could be when recorded directly at Wikipedia. Therefore, I tried to maximize the realism of the resulting data by designing different types of HITs for the workers, to cover different types of user behavior. In the following, I describe the HIT properties and present the three tasks used in this thesis.

HIT title and description

Every HIT must have a title and a short description which are displayed to the workers while they are searching for HITs to work on. In the first version of my HITs, I chose the title "Vandalise Wikipedia" for the HIT in which the workers had to vandalise an article. I decided for this title to let the workers mentally prepare for what they are going to do in the HIT. Although I clearly stated in the layout of the HIT itself that the workers do not have to work on the real Wikipedia but on a copy only, I received a notification from Mechanical Turk. They let me know that I am not allowed to create tasks which force workers to vandalize on websites and even blocked me temporarily from their service. As a result I had to change the title and the description to a less misleading version. Finally I decided to choose the same neutral title and description for all the HITs. The title read as "Research on Writing Behaviour" and the description as "We want to study writing behaviour and how people behave given a specific task.".

HIT layouts

I provided the layout for the HITs themselves as HTML templates. Every template shared a lot of similarities and only differed slightly between the different tasks. For example, every template consisted of a detailed task description including multiple hints to the usage of our copy of Wikipedia. Figure 3.1 shows a screenshot of an example HIT layout.

Research on Writing Behaviour

Instruction

Please note, that you're **not** editing the real Wikipedia during this task; therefore, your changes will **not** affect the real Wikipedia!

Please note, that we only accept complete assignments: Every article has to be processed!

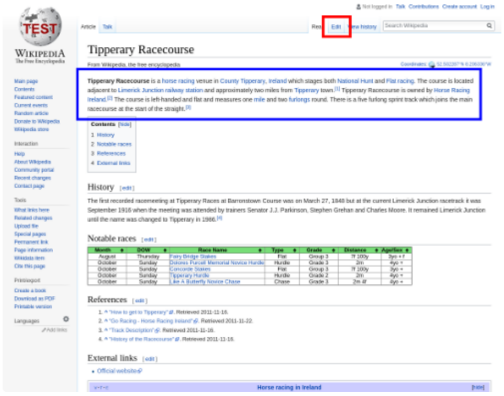
Please do **not** work on this HIT using a smartphone, a tablet or other touch-devices!

Task

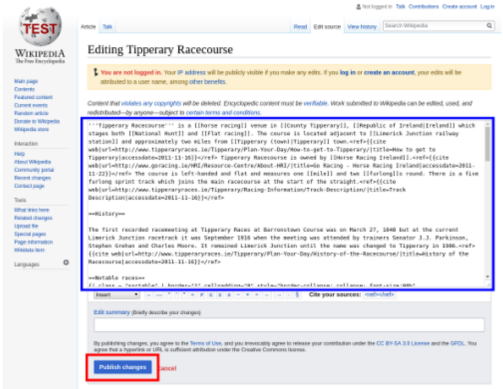
Your task is to put yourself into the shoes of a vandal and destroy an article on our Wikipedia copy.

We provide you with three links
Perform the following steps for each link:

1. Click on the link
2. Locate the summary of the article (the summary is always on the top of the article) (blue box in the first image)
3. Go to the edit page (red box in the first image)
4. Vandalize the article (blue box in the second image)
5. Click on the "Publish"-button (red box in the second image)



blue box: summary of the article,
red box: link to edit page



blue box: editarea for the article,
red box: publish button

Edit the following links (they will open in a new window)

- Article 1
- Article 2
- Article 3

Do you have any comments or did you encounter technical issues? Click here

Figure 3.1: An example HIT layout processed by the workers.

Popup As an ultimate safety precaution, to prevent getting blocked from Mechanical Turk again, the workers were provided with a popup (shown in Figure 3.2), notifying them about the Wikipedia copy as soon as they view the HIT and even before they were able to fully read the instructions for the HIT. Additionally, a screenshot of a browser window in the popup highlighted hints indicating the use of a copy of Wikipedia.

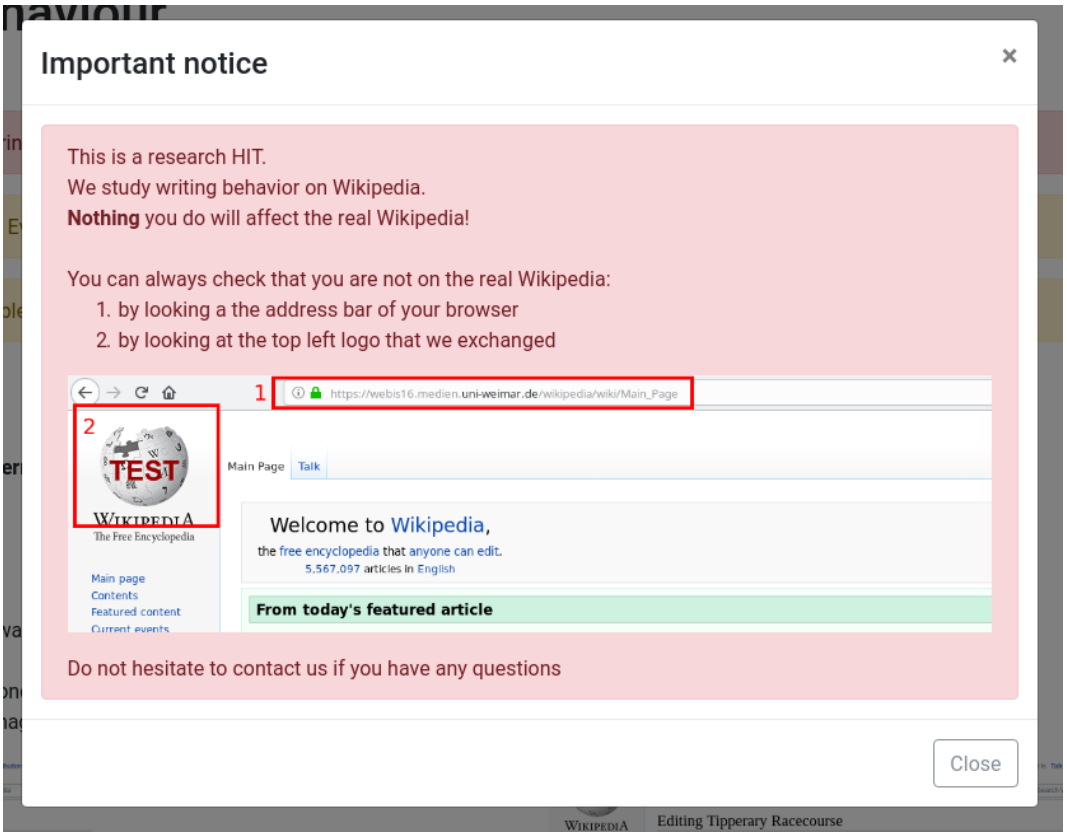


Figure 3.2: This popup-message was displayed to every worker as soon as he saw the HIT.

Task description In every template the workers were told to only edit the summary of the respective article. The summary can be found on the top of every article, even above the table of contents block, if present. I decided for this to keep the required amount of work for the HITs low for both parties involved: The workers should not be required to read the whole text of one or multiple Wikipedia articles to solve one HIT only. By focusing the attention of the workers to one specific part of each article only, I was able to fetch more data in a specific period of time. The workers could work faster on the HITs and

were also less exhausted or maybe even bored solving the HITs. Additionally, editing the summary of each article only also decreased the amount of time I needed for the preparation and the resulting review of each assignment. Despite the focus on the summary of each article only, neither did I restrict editing of other parts of the article technically, nor did I reject such results from the workers.

To support the workers solving the tasks, I provided a list of steps they had to follow and two images displaying my copy of Wikipedia and highlighting the most important segments, buttons, and links for the tasks.

Links to the articles In addition to the description of the task, the workers were provided with three links, each referring to a Wikipedia article of my Wikipedia copy. The amount of links per HIT was set to three to keep the workload per assignment reasonable. Also, I am more interested in a high amount of different workers per article than a high amount of articles per worker. The more different workers I get to work on the articles, the more diverse and representative the resulting dataset is.

Comment box At the bottom of each HIT, I provided a form input to send any comments or to report technical issues encountered solving the HIT. In the final design, I had to collapse the comment box by default. Too many workers pasted the contents of the "edited" articles into the box instead of editing the articles directly on my Wikipedia copy. The majority of the workers did not use the comment box, but some let me know they enjoyed especially the vandalism tasks, because they always wanted to vandalise Wikipedia without any consequences.

3.2.2 Worker Tasks

In what follows, I briefly describe the three templates I used to collect data from the workers.

“Vandalize”-Task This task asked the workers to do anything a real vandal would do without any further information or restrictions. The instructions were as follows:

> "Your task is to put yourself into the shoes of a vandal and destroy an article on our Wikipedia copy."

The goal of this template was to get a diversity of different types of vandalism by keeping the type of vandalism unrestricted. The workers were supposed

to follow each of the three links in the HIT to the corresponding, unmodified article and vandalize the text in any way. I accepted all assignments altering the article in any malicious way, due to the unrestricted task.

"Edit"-Task This task was meant to get some non-vandalism edits from the workers. The instructions were as follows:

> "We found some spelling errors on our copy of Wikipedia. Your task is the detection and correction of these spelling errors."

I prepared the articles by introducing 2-3 spelling errors into the text. The following enumeration lists the different error types I used:

- Insertion of a letter (example: vandalism → vandaalism)
- Deletion of a letter (example: vandalism → vandaism)
- Swapping of two letters (example: vandalism → vadnalism)
- Wrong capitalization of a letter (example: vandalism → vandaliSm)
- Deletion of a whitespace (example: this is vandalism → this isvandalism)

The workers were supposed to visit the linked articles, scan the summary of the text for the introduced spelling errors, and finally correct the spotted errors. I accepted the assignment if at least one spelling error per article was corrected and the articles were not destroyed showing the benign behavior of the worker.

"New Fact"-Task This was the second task with the goal of getting benign edits from the workers. The instructions were as follows:

> Your task is to add some given information to the summary of a Wikipedia article. You are free to alter the existing summary of the article to include the new information in a proper way. Use your own phrasing/words!

As a preparation for this task, I extracted one or two sentences containing some information out of the summary of each article. The users were provided not only with the link to the articles but also with the previously extracted sentences for insertion in the articles.

3.2.3 HIT Parameterization and Execution

One important requirement of the resulting corpus was defined as diversity, meaning that the corpus should consist of as many different editing types as possible. Therefore, I decided to select a relatively small set of articles for the experiments and let these articles be annotated by a large number of workers.

Wikipedia articles I decided for the number of 60 articles, listed in Appendix A, for the experiments. This allowed me to be flexible regarding the division of articles into HITs during my planning of the experiments.

I selected these articles randomly using the Wikipedia-API filtering by articles whose summary contained 300-1000 characters. The lower-bound filtering happened to ensure the articles contain enough information in the summary to extract some of the sentences for the "new fact"-task. Additionally, I dismissed articles with a long summary to not fatigue the workers in the "edit"-task by forcing them to scan a lot of text and find the hidden spelling errors.

Design As mentioned previously, I chose the amount of three articles per HIT leading to a total number of twenty HITs per type of task. I set the amount of workers per HIT to thirty, meaning every HIT is processed by thirty different workers. Since each worker could work on all the twenty HITs, it is possible that all the twenty HITs are processed by a total of thirty different workers but this is unlikely to happen in practice. On the other side, it could happen that every worker only works on one HIT and I get results from 600 different workers at the end. But as the first case, this event has a low probability and in practice the actual number of workers per task will range between these bounds. In total this resulted in 600 assignments per task; therefore, I was able to get 1,800 articles per task in theory. Since some of the assignments needed to be rejected in practice, I received only 1,278 assignments and 3,808 articles at the end.

Payment and processing time I chose to pay the workers 30ct per assignment for the "Vandalism" and "Edit"-task since both tasks approximately require the same amount of work. The third task ("New Fact") requires the workers to understand the given information, rephrase it in their own words, and insert the result in the article. Because of this extra work I paid 50ct per "New Fact"-task.

Similarly to the payment I choose the maximal time for the worker per assignment: The workers were able to work one hour on each "Vandalism" and "Edit"-task and two hours on each "New Fact"-task.

Task	Assignments		
	Total	Valid	Approved
Vandalism	600	584	511
Edit	600	573	436
New Fact 1	108	105	80
New Fact 2	95	91	73
New Fact 3	48	46	35
New Fact 4	101	92	74
New Fact 5	89	81	69
Total	**1,641**	**1,572**	**1,278**

Table 3.1: The amount of total, valid and approved assignments per task.

Execution I executed the three different tasks in sequential order, one after another, to maintain some control over the process of the experiments. I started the data construction with the "Vandalism"-task followed by the "Edit"-task; Finally, I published the "New Fact"-task to Mechanical Turk.

Results In total, I received 1,641 assignments resulting in a total of 1,278 valid and approved assignments. The detailed numbers assignments are stated in table 3.1.

Technical Issues I could not reconstruct the user interactions for all of the submitted assignments due to the quite complex experiment setup and the diversity of technology, like the browser version, used by the workers. In the following, I will refer to assignments without technical issues as "valid" assignments and to assignments with technical issues as "invalid" assignments.

Other Issues Some of the workers processing the "Vandalism" task returned for the "Edit"-task and did not re-read the instruction due to the similar design of the HIT templates. As a result they vandalized the articles instead of correcting the spelling errors. I did not reject their assignments, but marked those assignments as "wrongly vandalized" to make use of their included articles as further examples for vandalized articles. After noticing these wrongly vandalized assignments, I made some visual changes to the last "New Fact"-task to highlight the changed instructions to prevent this misunderstanding for the last task.

Another issue was that my tasks were repeatedly reported to the Mechanical Turk operators. Although, I prepared the tasks with multiple notes point-

ing out the workers are not working on the real Wikipedia but on a copy only, some of the workers started reporting the "New Fact"-task. As a result my task was stopped by Mechanical Turk and I was advised to review my HITs.

To gain more control over the "New Fact"-task, I split the twenty HITs of the task into five chunks of four HITs each and uploaded each chunk sequentially to Mechanical Turk. A drawback of this splitting was the increased amount of time needed to get all the results for the "New Fact"-task because I had to wait for every chunk to be completed before I was able to submit the next chunk. For unknown reasons, every chunk took longer to be completed by the workers than the previous task. The most likely explanation is that the chunks were submitted on working days and the previous two tasks on weekends.

3.2.4 Interaction Logging

While the workers processed the given HITs, I tracked and saved their mouse and keyboard events. The workers were tracked continuously from their first visit of the task page on Mechanical Turk over all the pages on my copy of Wikipedia until they finished and left Mechanical Turk. The events were collected on the client side and were sent in chunks to my web server. Once they were received by the server, I saved them into a file without any preprocessing to minimize the needed processing time for this task. The content and the layout of these logs are described in detail in the implementation Chapter 6.1.1 of this thesis.

As a result of this logging, I received not only the explicitly submitted changes in the articles from each worker but also their implicitly submitted interactions with the corresponding web page

3.3 Post-processing

The saved logs were post-processed in an offline step after the workers finished the tasks. The goal of this post-processing step is the conversion of the raw logs fetched from each worker into a representation of interactions on an article. The raw logs contain not only the mouse and keyboard interactions of the workers editing the Wikipedia articles but also the interactions collected on the Mechanical Turk pages. These events were discarded in the post-processing because they are not available in a real-life scenario and also not needed for this thesis to investigate the behavior while editing an article. Despite the removal of these events for this thesis, they could be used to research other similar topics using the recorded raw logs. The remaining events were grouped by the Wikipedia articles from which they were recorded. Since

every assignment contained three links to an article, each assignment yielded three groups of events. Every group of events from all the tasks were converted into a quick-to-parse representation of articles. I removed ten articles with zero submission of changes. These are articles for which the workers did not submit any changes through the Wikipedia page but the corresponding assignments were still approved because the other articles in the assignment were processed in a valid way. Additionally, I annotated for every article edit whether the article was edited in the input field of our Wikipedia copy directly or in some external editor with the result copied into my copy of Wikipedia. Finally, I reviewed the vandalism collected and identifier three different kinds:

Destructive
Simple deletion of the text and/or destruction of the Wikitext[2] syntax.

Gibberish
Insertion of an arbitrary amount of random characters

Sophisticated
More or less complex, structure-preserving edits on the article.

I tagged at least one of these vandalism types to every article edit from the "Vandalism"-task to be able to compile subsets of the article dataset depending of the type of vandalism.

3.4 Dataset Statistics

In this section, I describe the statistics of the resulting dataset used for the experiments in this thesis. The structure and format of the dataset is described in Chapter 6.2.

Articles My final dataset contains 3,808 articles with the corresponding interactions. 2,021 of these articles contain vandalism resulting in a total of 53% vandalistic edits compared to 47% benign edits. Only 182 or 5% of the articles were edited externally. All other 3,626 articles were processed directly in the input field of our Wikipedia copy. In more detail, 94 articles containing vandalism and 88 articles with benign edits were edited externally. 1,020 articles contain sophisticated, 925 articles contain destructive and 340 articles contain gibberish vandalism. Table 3.2 shows the number of articles for every different combination of types of vandalism.

[2]Markup language for and by Wikipedia

Combination	Amount of articles
Sophisticated (S)	910
Destructive (D)	681
Gibberish (G)	172
D + G	148
D + S	90
S + G	14
D + S + G	6
Total	**2.021**

Table 3.2: Distribution of types of vandalism

798 of the articles with benign edits were extracted from the "Edit"-Task and 989 articles were extracted from the "New Fact"-Task and are referred to as "simple edits" and "complicated edits", respectively.

Workers In a total, the articles were submitted by 335 workers, where 132 workers submitted the articles containing vandalism and 224 workers articles with benign edits resulting in an overlap of 21 workers who submitted both types of edits. Every worker submitted at least one article and at most 123 articles resulting in a mean of 11.37 articles per worker. The distribution of articles per worker is shown in Figure 3.3.

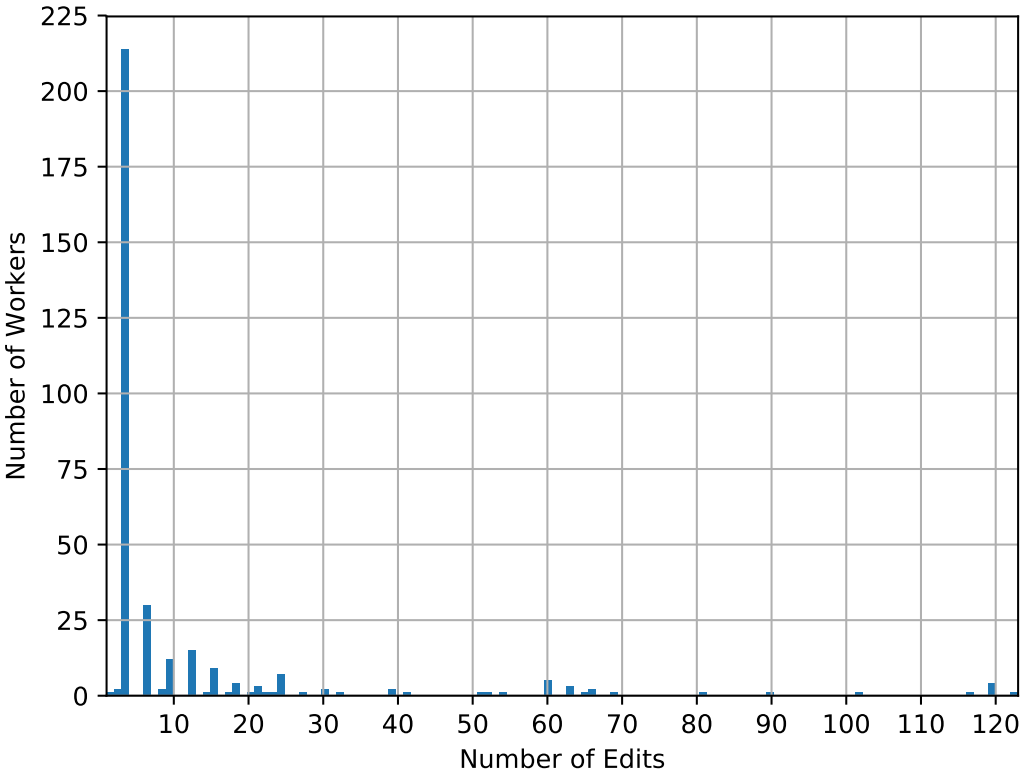


Figure 3.3: Distribution of articles per worker

Chapter 4

Vandalism Detection Model

In this chapter, I describe the approach I chose to solve the problem of vandalism detection on Wikipedia. As already mentioned in the introduction of this thesis, I am considering the problem of vandalism detection on Wikipedia as a machine learning problem. The first section in this chapter gives an introduction over the topic of machine learning related to this thesis. The second section describes how I modeled vandalism detection as a machine learning problem.

4.1 Machine Learning

Machine learning is an approach in computer science to solve a given learning problem. In this thesis, the learning problem is a classification problem and can be defined as the following decision: "Is a given article edit vandalism?". Figure 4.1 illustrates the machine learning approach to solve such problems. Given a set of real world objects O and a set of classes C, a classifier tries to choose an appropriate class for each real world object. In this thesis, the real world objects are article edits and the classes are malicious and benign edits. The assignment of a real world object $o \in O$ to a class $c \in C$ can be expressed as $c = p(o)$ with p as the classifier. A perfect classifier, for example a human expert, is able to sort all objects in O into their corresponding classes. To apply machine learning, one has to embed all real world objects into the feature space X by extracting relevant characteristics from every object, resulting in a set of feature vectors. This transformation can be expressed as $x = t(o)$ with t as the model function. Finally, one tries to approximate a perfect classifier for feature space objects as close as possible to assign the objects from the feature space into their corresponding classes. This assignment can be expressed as $c = a(x)$ with a as the approximated classifier.

In a supervised learning approach, the classifier is trained with a set of ex-

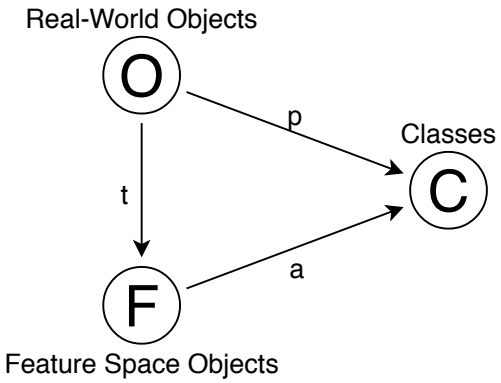


Figure 4.1: Real world objects (O) should be assigned (p) to their corresponding classes (C). In machine learning, they are transformed (t) into features (F) and fed into a classifier which tries to predict (a) the correct class

ample objects, a so-called training set. In this thesis, such objects are article edits with a known corresponding class. A chosen machine learning algorithm is fed with those examples and can be used to predict the classes of another set of example objects referred to as test set. In an evaluation step, the target classes of those examples would be known and the performance of the prediction can be evaluated. In this thesis, I decided to use the random forest algorithm for all the experiments performed in Chapter 5. This algorithm generates multiple decision trees and combines their individual results to one final score.

4.2 Model

In machine learning, the transformation of a real world object into a feature space is called a model. It consists of a set of features each representing the object in a way relevant to the classification task at hand. In the following, I describe the features used in this thesis, where most of them have been applied for other classification tasks in the related work..

4.2.1 Remarks

Interactions To investigate the behavior of workers in a more sophisticated way, I decided to consider not only raw worker events to extract features from the article edits but also to group related events into "interactions". I defined an interaction as a subsequent set of recorded events of the same type without any interruptions of events of a different type. I define two different

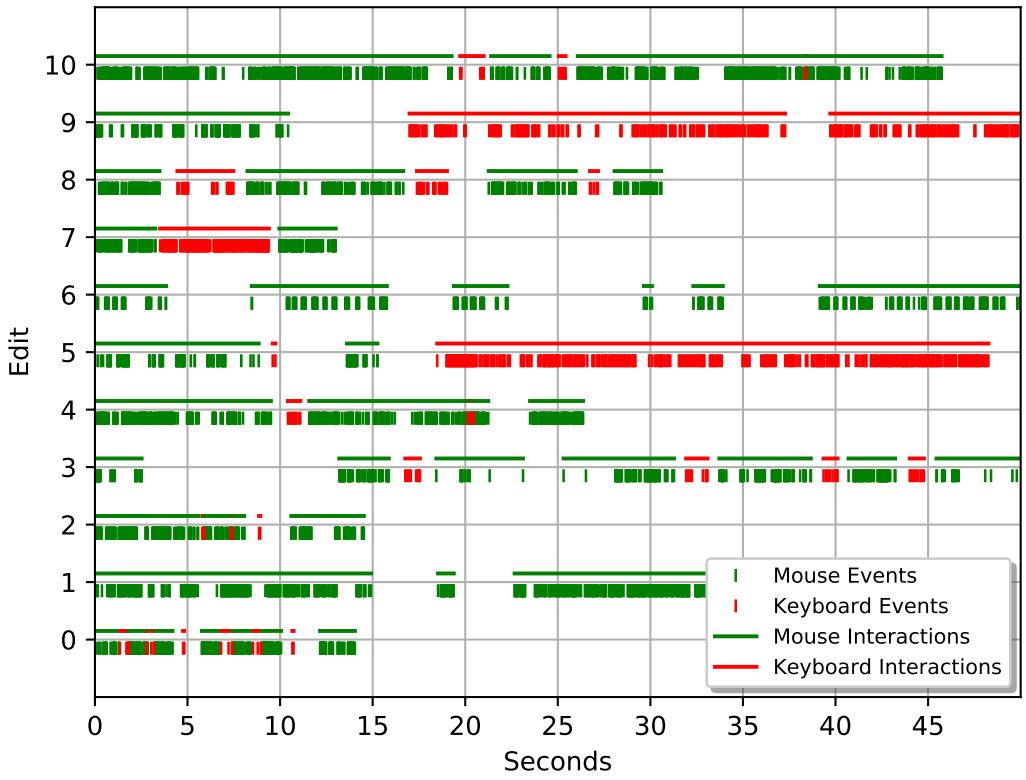


Figure 4.2: Mouse and keyboard events of ten randomly selected edits. The corresponding interactions are displayed above the events aggregating them if appropriate.

event types: "mouse" and "keyboard". The former groups all "mousedown", "mouseup", "wheel", and "mousemove" events and the latter includes all "keyup" and "keydown" events. Additionally, I split interactions if the timegap between two events is larger than two seconds even if all events are of the same type. Figure 4.2 shows the events of ten different edits for the first fifty seconds of each edit displayed as thick horizontal lines. The generated interactions are displayed above the corresponding events as thin horizontal lines. For most of the features, I separately calculated the corresponding scores for all events and for each interaction of an edit.

Statistics To aggregate different feature scores across interactions or a whole edit, I determined the following seven statistical values for the list of scores:

- Mean
- Min value
- Max value
- Standard deviation
- Skewness
- Median
- Sum

If a feature consists of one score only, for example, the distance traveled by the mouse, the statistics are applied only at the end to combine the score for all edit pages, most often a worker visited the edit page only once, resulting in a total of seven values. On the other side, if a feature returns a list of scores, for example the mouse speed between every mouse event, the list is aggregated to the seven statistical values. This is again repeated for every edit page and therefore results in a total of 49 values. If a feature score is calculated for every interaction, the statistics are applied additionally, resulting in an another multiplication of the number of statistical values by seven.

4.2.2 Features

The used features in this thesis can be divided into general, mouse movement and keyboard features. Each group of features is going to be described in more detail in the next paragraphs, followed by a complete listing of all features in Table 4.1.

General Features The majority of features in this group indicate the duration and length of the article edit. The only feature including subpages of an article besides the edit page is the total duration an article is being visited. For example, the total duration includes also the period a user remains on the main page of an article. A related feature represents the duration a user remains on the edit pages only, due to the important role of the edit page in this thesis. Additionally to the duration features, I count not only the number of pressed keys and clicked mouse buttons during an edit but also the number of mouse and keyboard interactions. Finally, I record the first and the last appearances of mouse and keyboard interactions, respectively.

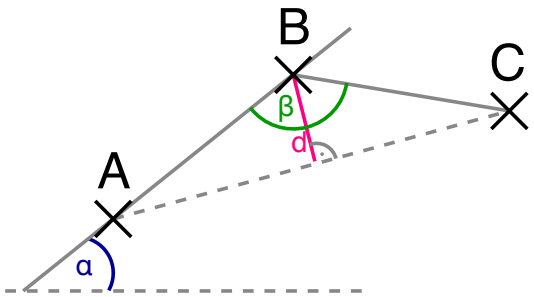


Figure 4.3: The angles α and β are used directly as features whereas the distance d is used to calculate the curvature distance.

Mouse Movement Features As shown in previous related work, mouse movements can be used to characterize users in order to recognizing their current mood; therefore, I implemented several mouse movement features. I compiled a lot of those features from Zheng et al. [2011], who made heavy use of mouse movements to verify users. The direction of a mouse movement (shown as α in Figure 4.3) between two consecutive cursor positions A and B is defined as the angle between the line $\overrightarrow{AB}$ and a horizontal line. Using the timestamps recorded at the cursor position A and B, I additionally calculated the speed of the cursor between those two points. The angle of curvature is calculated using a third cursor position C following B and is defined as the angle between the lines $\overrightarrow{AB}$ and $\overrightarrow{BC}$ (shown as β in Figure 4.3). Finally, Zheng et al. defined the curvature distance as the ratio of the length $|\overrightarrow{AC}|$ to the shortest distance from point B to the line $\overrightarrow{AC}$ (shown as d in Figure 4.3). As more general feature values, I calculated the total distance and time the cursor traveled.

Keyboard Features The keyboard features used in this thesis can be further separated in two subgroups, key statistics and keystroke dynamics. Among others, key statistics include the number how often the backspace key was pressed, the usage ratio of the left, and the right side of the keyboard, and the number of alphabetic, special, control, and numeric keys. A complete list of the calculated key statistics is shown in Table 4.1. The keystroke dynamic features are adopted from Plank [2018] and describe how long a user pressed a key, and long the pauses between two key presses are.

Feature	Overall	Interactions	Total Count
General			
Total Duration	1	-	1
Edit Duration	7	-	7
Edit Length	7	-	7
#Clicks	7	-	7
#Keyboard Interactions	7	-	7
#Mouse Interactions	7	-	7
First Mouse Interaction	1	-	1
Last Mouse Interaction	1	-	1
First Keyboard Interaction	1	-	1
Last Keyboard Interaction	1	-	1
Mouse Movement			
Mouse Angles	49	343	392
Mouse Curvature Angle	49	343	392
Mouse Speed	49	343	392
Mouse Distance Ratio	49	343	392
Mouse Distance	7	49	56
Mouse Duration	7	49	56
Keyboard			
#Keydowns	7	49	56
#Backspace	7	49	56
#Enter	7	49	56
#Space	7	49	56
#Escape	7	49	56
#Control	7	49	56
#Shift	7	49	56
#Capslock	7	49	56
#Exclamation Mark	7	49	56
#Question Mark	7	49	56
#Hyphen	7	49	56
#Underscore	7	49	56
#Asterisk	7	49	56
#Plus	7	49	56
#Special Character	7	49	56
#Control Character	7	49	56
#Alphabetic Character	7	49	56
#Lowercase Character	7	49	56
#Uppercase Character	7	49	56
#Numeric	7	49	56
#Other	7	49	56
Left-Right Ratio	7	49	56
Dwell Time	49	343	392
Flight Time	49	343	392
Total	**502**	**3,234**	**3,736**

Table 4.1: All features displayed with the corresponding number of feature scores after applying the statistics.

4.2.3 Wikimedia Features

Additionally to my own feature set, I implemented the features used by Savithri et al. in their spambot detection project at Wikimedia already mentioned in Chapter 2. I used the Wikimedia model in my experiments to compare it with my model and to evaluate the performance of both models. The Wikimedia features are similar to my own features and can also be grouped into mouse and keyboard features. In addition to the following description of the features, Table 4.2 shows the complete list of Wikimedia features.

Feature	Overall	Interactions	Total Count
Mouse Movement			
Mouse Curvature Angle	25	125	150
Mouse Speed	25	125	150
Mouse Acceleration	25	125	150
Mouse Click Times	25	125	150
Keyboard			
Dwell Time	25	125	150
Flight Time	25	125	150
Delta Dwell Time	1	1	2
Delta Flight Time	1	1	2
Total	**152**	**752**	**904**

Table 4.2: All Wikimedia features displayed with the corresponding number of feature scores after applying the statistics.

Keyboard Features Four different keyboard features were selected by the Wikimedia group. For a given list of consecutive key presses, the dwell times and the delta dwell times are calculated. The delta dwell times are the difference between successive dwell times. Analogically, the flight times and delta flight times are generated.

Mouse Features Wikimedia calculated the speed, curvature, and acceleration of the mouse movements. Additionally, they extracted the delta click time, the time between successive mouse clicks, of the mouse data.

Statistics The statistics of the Wikimedia project differ from my previously mentioned statistics:

- Mean
- Variance
- Skew
- Kurtosis
- Interquartile Range (IQR)

Additionally, Savithri et al. do not apply all the statistics to the delta dwell and delta flight times. Instead, only the mean value of the times is calculated. To ensure the comparability between Wikimedia's and my model, I calculated the final Wikimedia feature scores the same way as I did for my features which means that I generated the feature scores both, for the overall events of an edit page and for each interaction separately.

Chapter 5

Evaluation

In this chapter I investigate the following research questions already mentioned in the introduction:

1. Is it possible to detect vandals on Wikipedia based on their behavior while editing?
2. Is it possible to detect vandals without rendering individuals identifiable at the same time?
3. How much time does it take to detect a vandal?

I performed several experiments to answer these questions which are described and evaluated in the following sections. Every experiment was repeated one hundred times and the different outcomes were averaged to obtain information about variance. Some charts show the standard deviation resulting from the multiple iterations of the values as an area around the corresponding plot. Furthermore, all experiments generate a training and a test set from the given edits to train a classifier with the training set and to evaluate its performance with the test set. To improve the result quality and to ensure the generalization of my model, there exists no edit in the training or test set which shares the same worker with another edit in the other set, respectively; in other words, a worker can never have edits in both sets. As a result, I worked on worker level instead of edit level to create the training and test sets. For consistency reasons, I decided for a test set size of 20% for all experiments if not explicitly stated otherwise. I evaluate the performance of the trained classifiers by computing the F1 score which combines the precision and recall scores into a single value using the following formula:

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \tag{5.1}$$

Experiment	Own Model			Wikimedia Model		
	Precision	Recall	F1	Precision	Recall	F1
Number of workers	0.81	0.74	0.77	0.74	0.66	0.69
Edits per workers	0.79	0.81	0.79	0.74	0.78	0.74
Simple vandalism	0.85	0.77	0.8	0.79	0.69	0.71
Complicated vandalism	0.77	0.69	0.7	0.71	0.6	0.62

Table 5.1: Results for the vandalism detection experiments.

Precision is defined as the fraction of edits correctly classified as vandalism and recall is the fraction of detected vandalistic edits shown in Formula 5.2 and 5.3, respectively.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{5.2}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{5.3}$$

5.1 Vandalism Detection

I considered two different scenarios to evaluate the general vandalism detection performance of my model. In the first scenario, the number of workers in the training set is increased and in the second scenario the amount of edits per worker is increased. Additionally, I compare the detection performance between simple and complicated vandalism. The results for these experiments can be found in Table 5.1.

5.1.1 Increasing Number of Workers

I separated the edits of 20% of all workers, randomly selected, into the test set and initiated the training set with the articles of the first of the remaining workers. I trained the classifier with the training set and predicted the classes of the edits in the test set. I kept adding one worker and evaluated the performance until the training set included all workers not in the test set. For comparability, this experiment was executed for my own model and for the model of Wikimedia. The results are displayed in Figure 5.1 and show that the performance of the classifier is increasing the more workers are used in the training set. The impact of additional workers is decreasing the more workers are already used, indicating that at some point adding more workers will not further improve the performance of the vandalism detection.

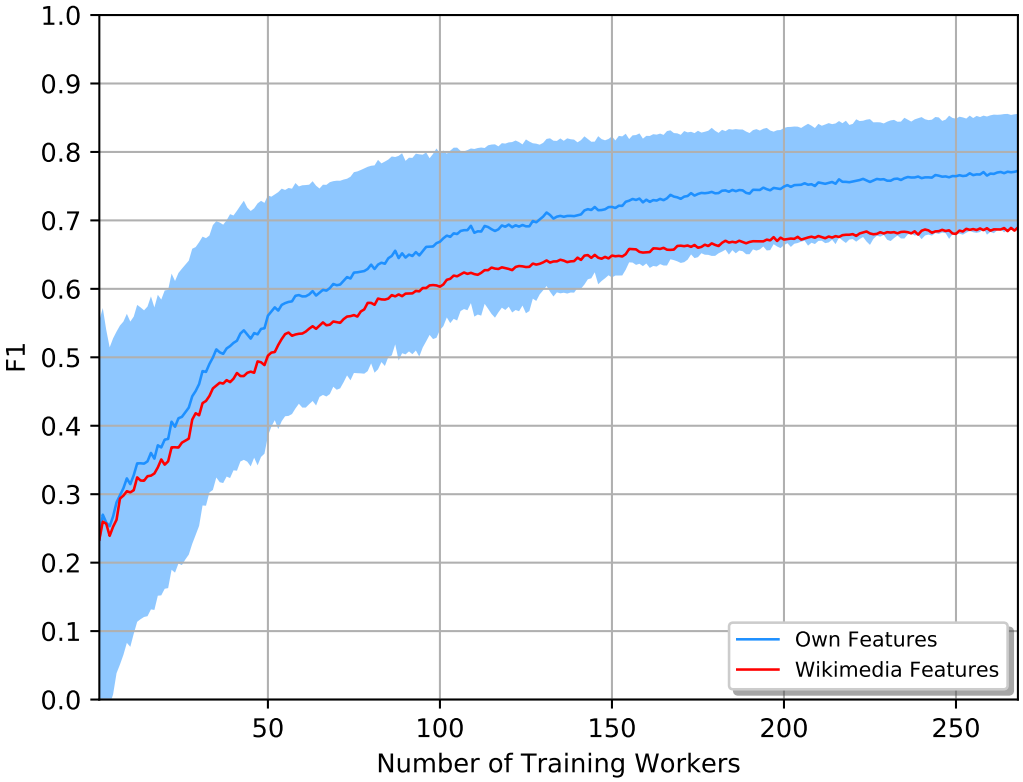


Figure 5.1: Change of F1-performance of vandalism detection for different numbers of workers in the training set.

5.1.2 Increasing Number of Edits Per Worker

In this scenario, I started with a small number of edits per worker and increased this number until all edits of the training workers were used. To ensure comparability while increasing the number of edits per worker, I decided to increase the edits by an absolute value; therefore, I used only workers with more or equal than fifty edits resulting in a total of 24 workers. I randomly chose exactly 50 articles from those workers and randomly selected 20% of those workers for the test set. All other workers are used in the training set but only five edits from each of those workers are selected for the initial training set. I iteratively increased the number of edits per worker by five until all fifty edits per worker are used in the training set. In each iteration, I trained the classifier and evaluated the performance with the test set. Figure 5.2 shows the results of this experiment. Like in the first experiment, the performance of the classifier improved the more edits per workers were used, but adding more edits has less impact to the performance than adding more workers. Furthermore, the figure shows that a good prediction can be achieved even with a low amount of edits per worker, whereas the first experiment shows that it is not possible to predict vandalism reliably with a low amount of workers. This indicates that it is more important to know edits from many different workers to improve the performance of vandalism detection instead of using a set of some few workers with a lot of edits per worker.

5.1.3 Detection of Simple and Complicated Vandalism

Initially, I divided the edits into simple and complicated edits. Simple edits are either edits from the "Edit"-task or edits annotated as "destructive", "gibberish", or "destructive and gibberish" from the "Vandalize"-task. Complicated edits are either edits from the "New Fact"-task or annotated as "sophisticated" from the "Vandalize"-task. The simple edits were performed by 116 different workers and the complicated edits by 253 workers. To ensure the comparability between the simple and the complicated case, I randomly selected 116 workers from the complicated edits. For the simple and the complicated set of workers, I separated the test set and trained the classifier with one worker of the remaining set, respectively. Until all the remaining workers were used in the training set, I added one worker in each iteration to the training set and evaluated the performance of the classifier. The results are shown in Figure 5.3 and show that for both models, mine and Wikimedia's, it is easier to predict vandalism for simple edits than it is for complicated edits. This result was expected, since complicated edits do not include simple-to-detect properties like holding a key for a longer period of time or simply deleting text only. For both,

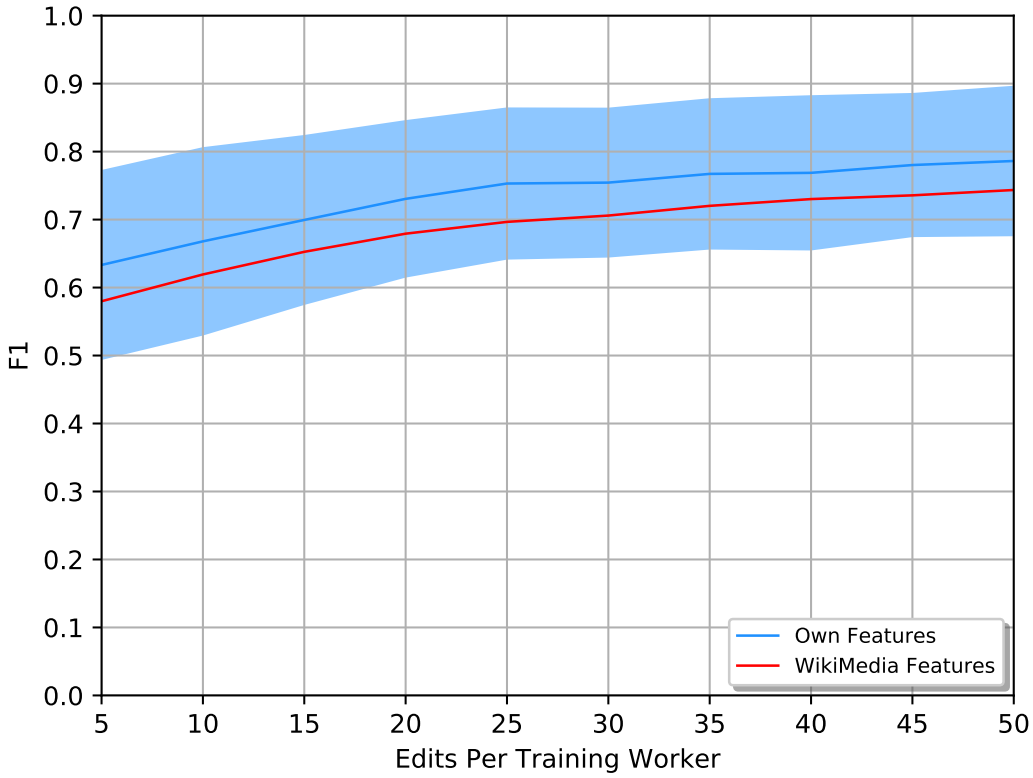


Figure 5.2: Change of F1-performance of vandalism detection for different number of edits per workers in the training set.

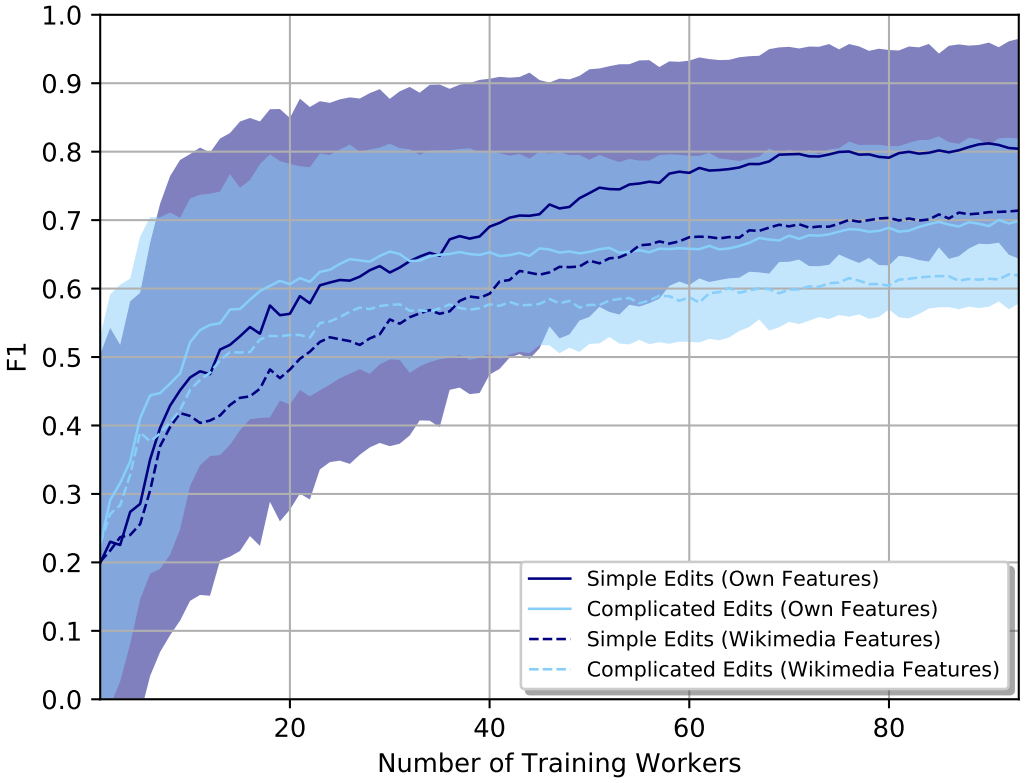


Figure 5.3: Comparison of F1-performance of vandalism detection between simple and complicated edits for different number of workers in the training set.

sophisticated vandalism and sophisticated edits, a user has to interact more carefully with the article than he would have to do for simple edits. The way the "Edit"-task was designed, the interactions of the users can be more easily distinguished from simple vandalism which does not include the modification of at most three words in the article.

5.1.4 Conclusion

The presented results show that, given a sufficiently-sized training set, it is possible to distinguish between benign and vandalistic edits using the recorded mouse and keyboard interactions of users only. In more detail, simple and sophisticated vandalism can be detected, although simple vandalism seems to differ more from simple benign edits than sophisticated vandalism from complex benign edits and therefore the performance of simple vandalism detection is better.

Experiment	Vandalism Detection			Attribution		
	Precision	Recall	F1	Precision	Recall	F1
All own features	0.83	0.75	0.78	0.78	0.78	0.78
Keyboard (KB)	0.8	0.76	0.78	0.56	0.56	0.56
Keystroke (KS)	0.79	0.69	0.73	0.61	0.61	0.61
Mouse movement (MM)	0.68	0.59	0.63	0.49	0.49	0.49
KB + KS	0.81	0.75	0.77	0.67	0.67	0.67
MM + KB	0.79	0.71	0.74	0.72	0.72	0.72
MM + KS	0.78	0.69	0.73	0.73	0.73	0.73
MM + KB + KS	0.82	0.73	0.77	0.77	0.77	0.77
Wikimedia features	0.75	0.64	0.69	0.61	0.61	0.61

Table 5.2: Results for the experiment reducing the amount of worker information.

5.2 Anonymous Vandalism Detection

To investigate the question whether it is possible to detect vandalism without being able to identify the workers, I first decided to design an experiment to measure how well it is possible to predict the author of an edit using my model and the model of Wikimedia. The prediction of an author is called authorship attribution and is defined in this thesis as the prediction of the corresponding worker to a given edit. In this thesis, the primary goal regarding authorship attribution is to achieve a low detection performance of individual workers due to the privacy issues created by tracking the users. For other use cases than vandalism detection, a high attribution performance might be desirable to assign different edits of unknown workers to their corresponding worker, for example. In the second experiment for this research question, I additionally investigated the performance of several subsets of my feature set to further reduce the amount of data recorded about a worker's interactions.

The results for these experiments can be found in Table 5.2.

5.2.1 Authorship Attribution

To evaluate the performance of authorship attribution, I extracted the 24 workers from the set of workers with more or equal than fifty edits and randomly selected exactly fifty edits from those workers. Subsequently, I divided the edits of those workers into a test and a training set. Instead of detecting whether an edit contains vandalism or not, the trained classifier tried to predict the worker of the edits in the test set. The result is shown in Figure 5.4 at the

Figure 5.4: This plot shows not only the authorship attribution performance at the leftmost values but also the decreasing performance if more workers are added to the training set.

leftmost position on the x-axis. My model achieves a F1 score of 0.78 and the Wikimedia model a F1 score of 0.61. Additionally, I added one worker of the workers with less than fifty articles to the training set until no worker is left to increase the set of possible workers an edit can be assigned to. Figure 5.4 shows how the performance of authorship attribution decreases the more workers are added to the training set. Considering the massive amount of workers on Wikipedia, it is unlikely that an individual worker could be identified if the set of potential workers is unrestricted.

5.2.2 Reduced Amount of Worker Information

In this experiment, I decided to split my feature set further into three subsets. I called the first set "Mouse Movement Subset" which includes the mouse movement features listed under "Mouse Movement" in Table 4.1 in Chapter 4. The second set, the so called "Keyboard Subset", contains all the features

listed under "Keyboard" in Table 4.1 except the dwell and flight times of the key presses. Those two features form the last subset the "Keystroke Subset". It is promising to split the keyboard and mouse features into different subsets to possibly be able to completely omit the one or the other from the tracked user data. Additionally, I decided to split the keyboard features into the keyboard subset and the keystroke subset because the latter does not contain any information which keys are pressed by the user but only contain information about the timings of the key presses.

In this experiment, I performed the first vandalism detection experiment from Section 5.1 with all the workers and the authorship attribution experiment presented in the previous subsection for every combination of feature subsets. I compared not only the three feature subsets alone, but also all possible pairs with these subsets and combined all of the three subsets to one single subset. This new subset of the mouse movement, keyboard and keystroke subset is not the same as the whole model presented in Chapter 4, since the general features listed in Table 4.1 are missing. Additionally, I compared the feature subsets with all of my own features and with the model of Wikimedia. Figure 5.5 displays the results of this experiment and shows that authorship attribution seems to be more vulnerable to reduced feature sets than vandalism detection. The largest performance drop of authorship attribution relative to the vandalism detection happens using the keyboard subset only. This feature set contains statistics about which keys were pressed and those statistics seem to be enough to isolate malicious from benign edits but not to distinguish different workers. The results from the keystroke and mouse movement feature sets are also interesting due to the low amount of recorded data needed from each user. The good vandalism detection of the keystroke subset shows how well a detection system could be applied while ensuring the privacy of the users by not recording any sensitive data like the pressed keys for example. The mouse movement subset is also able to detect vandalism to some extent but has the benefit to not be able to identify single authors; therefore, this feature set could be used to support other systems or feature sets in detecting vandalism without introducing privacy issues.

There is no significant performance loss in vandalism detection using the combinations of the mouse movement, keyboard and keystroke subsets compared to the usage of all features; Therefore, it is possible to heavily reduce the amount of recorded data and still maintain a high vandalism detection performance. Combining any of the mouse movement or keystroke subset to the keyboard subset it is possible to identify workers even if this should be necessary.

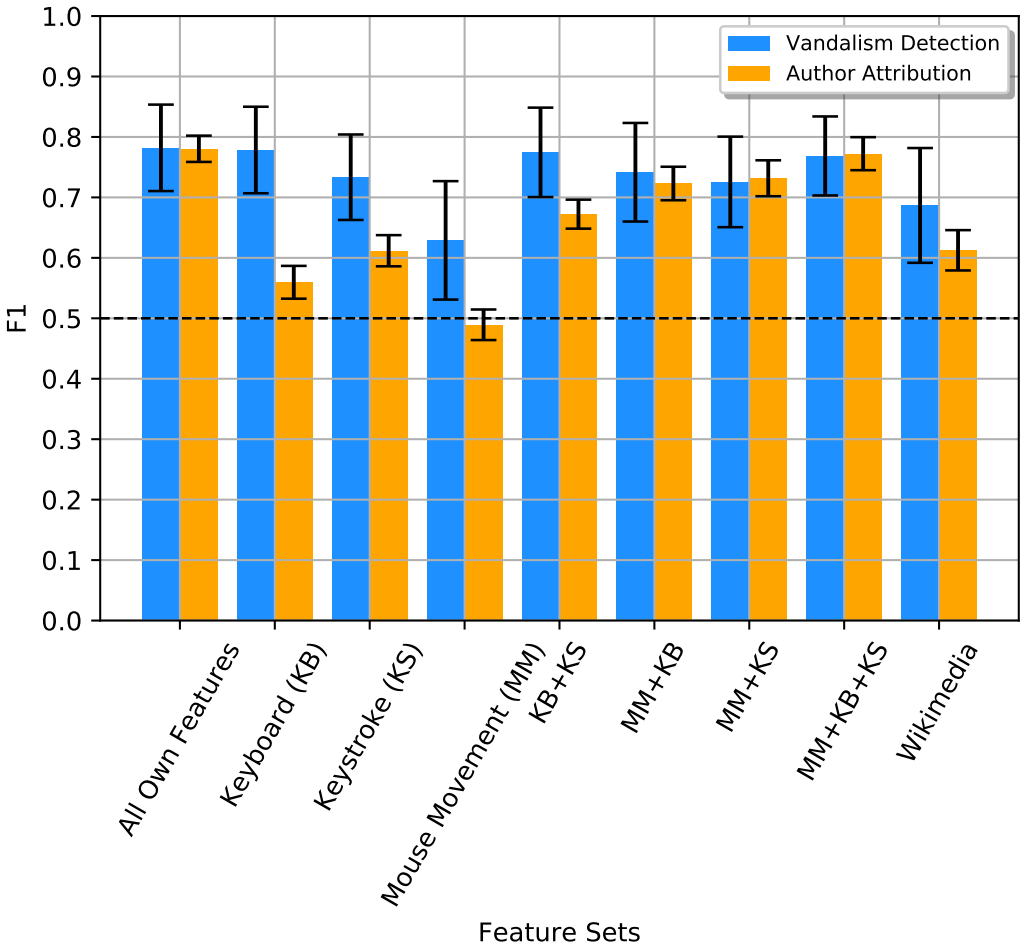


Figure 5.5: Comparison of vandalism detection and author attribution performance using different subsets of features.

5.2.3 Conclusion

Keeping in mind that the dataset the experiments were performed on is relatively small and should be extended in the future, the results show that it is possible to achieve a high vandalism detection performance while being unable to identify single workers. Choosing different feature subsets, it is also possible to heavily reduce the amount of user interactions to be tracked and still maintain a high detection performance; additionally, one could extend existing detection systems without creating any privacy issues.

5.3 Time Needed For Vandalism Detection

The third research questions asks how much time is needed to detect vandalism. In practice, this knowledge could, for example, allow to stop the tracking of a user after some time to make a decision whether the user is a vandal or not. To answer this research question, I investigated the vandalism detection performance using user interactions within a specific timespan after the first active interaction only where I defined an active interaction as a mouse or keyboard interaction initiated by a user. I decided to investigate the detection performance at the following timestamps, starting from the first active interaction of an user: 200ms, 400ms, 600ms, 800ms, 1s, 1.2s, 1.4s, 1.8s, 2s, 3s, 4s, 5s, 10s, 15s, 20s, 30s, 40s, 50s. Additionally, I compared the detection performance when all interactions of a user are considered. I decided to investigate the features until 50 seconds because the median of the duration users spent on modifying an article is 44.5 seconds. I investigated not only the general vandalism detection performance at the given timestamps but also the detection performance of simple and complex edits and using different feature subsets.

5.3.1 Time-Dependent Vandalism Detection

In this experiment, I randomly split the set of workers into a test and a training set and trained a classifier for every timestamp I defined earlier; finally, I used the test set to evaluate the detection performance for every timestamp. The results for my model and Wikimedia's model are shown in Figure 5.6. The vandalism detection performance is increasing steeply until 1.4 seconds after the first interaction of a user and is reaching a performance of approximately 0.2 below the final F1 value. From there, the performance does not increase that much, anymore and the performance grows slowly until it reaches the final F1 value when all interactions of the user are considered.

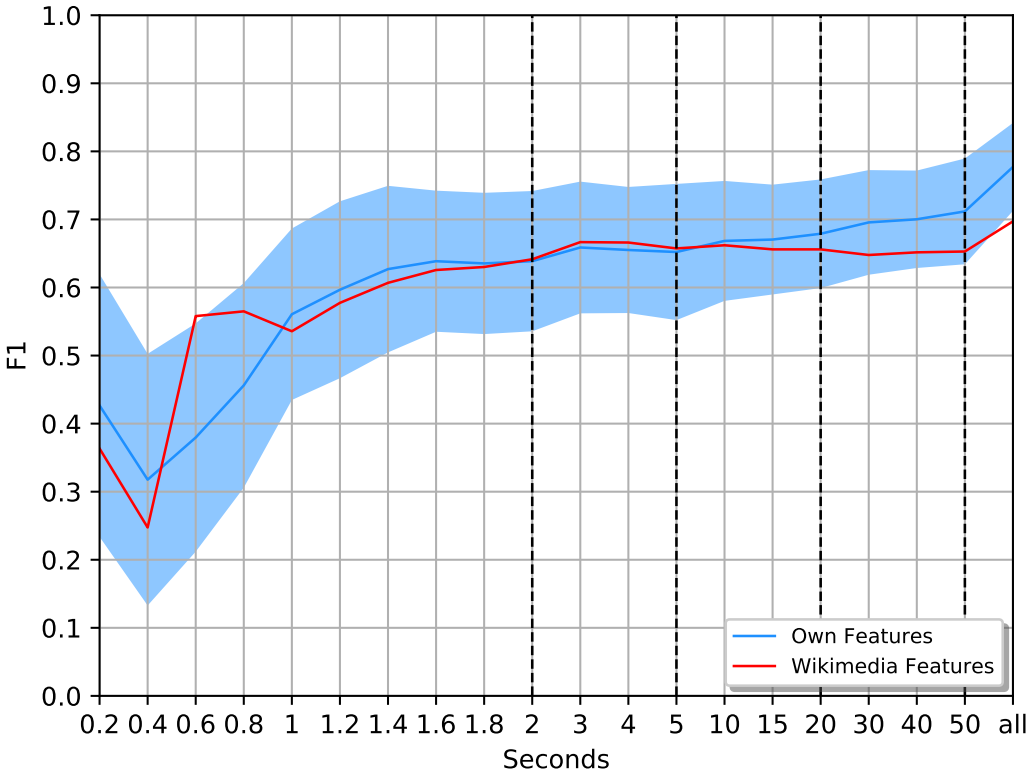


Figure 5.6: Development of vandalism detection performance over time.

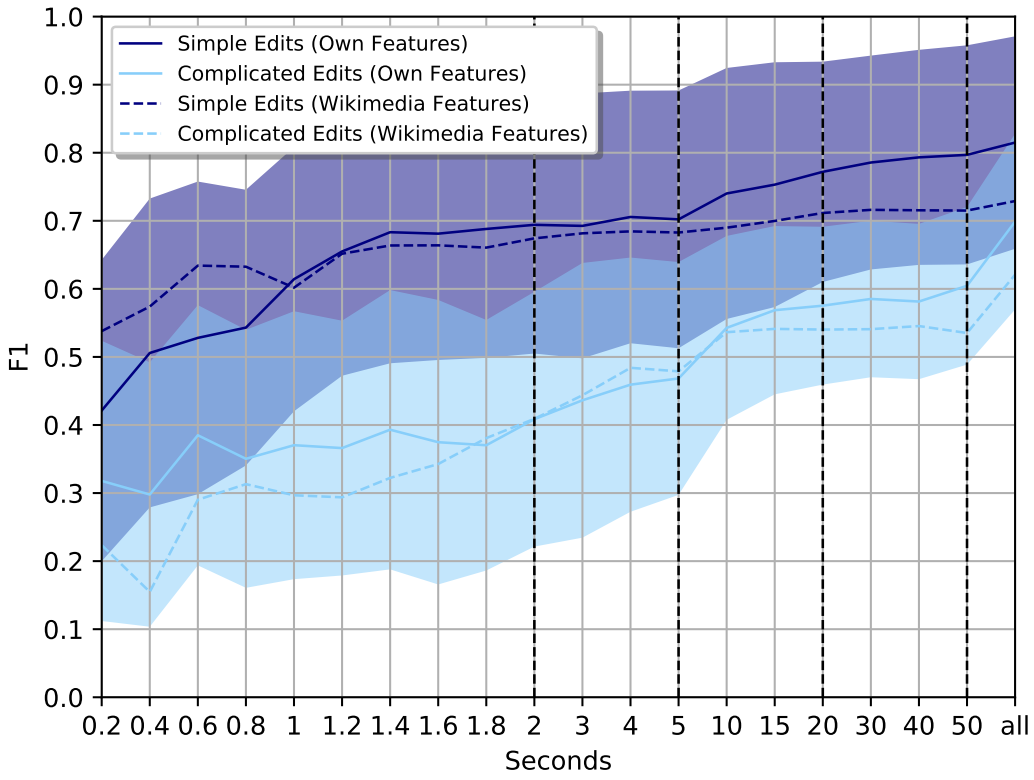


Figure 5.7: Comparison of the detection of simple and complicated vandalism over time.

5.3.2 Time-Dependent Detection of Simple and Complicated Vandalism

This experiment was designed the same way as the "Detection of Simple and Complicated Vandalism" experiment from Section 5.1 with the exception that all workers are used to create the test and training sets for the simple and complicated edits, respectively. Figure 5.7 shows the result of this experiment. As in the previous experiment, the vandalism detection for simple edits is increasing steeply in the first 1.4 seconds and then continues to increase slowly. Additionally, the rise seems to increase more rapidly again beginning from the fifth second until the tenth second, to decrease again from the tenth until the 20th second, and to decrease even more from the twentieth second until the end. The detection performance rise of complicated vandalism decreases more or less smoothly over the time also with a strong performance incrementation from the fifth until the tenth second.

5.3.3 Time-Dependent Vandalism Detection For Different Feature Sets

In this last experiment, I compared the detection performance over time for the keystroke, keyboard and mouse movement feature subsets presented in the "Reduced Amount of Worker Information" experiment. I split the workers into test and training sets and evaluated the prediction scores for every timestamp and displayed the resulting graph in Figure 5.8. As in the previous time-dependent experiments, the plots stabilize at around 1.4 seconds after the first active interaction but the plots differ in their behavior over time. The development of the keyboard subset behaves mostly the same as development for all features, whereas the keystroke performance is similar to these plots over time but increases slower. The prediction performance of the mouse movement subset increases until the fifth second and starts to decrease until, at some point beyond 50 seconds, it is decreasing again. This descent should be investigated further in future experiment; for now, it looks like one could stop the recording of mouse movements data at the fifth second and predict the behavior since the vandalism prediction performance will not be significant higher even if all interactions are used.

5.3.4 Conclusion

As expected, the prediction performance improves the more interactions of a user are considered in the feature calculation, but the previous experiments showed a significant stabilization of the prediction performance around 1.4 seconds after the first active interaction of a user. In a time-critical use case, this result could be used to set the earliest time a prediction is made to 1.4 seconds. In contrast to the other plots, the detection of complicated vandalism needs more time, around fifteen seconds, until the performance does not increase significantly, anymore.

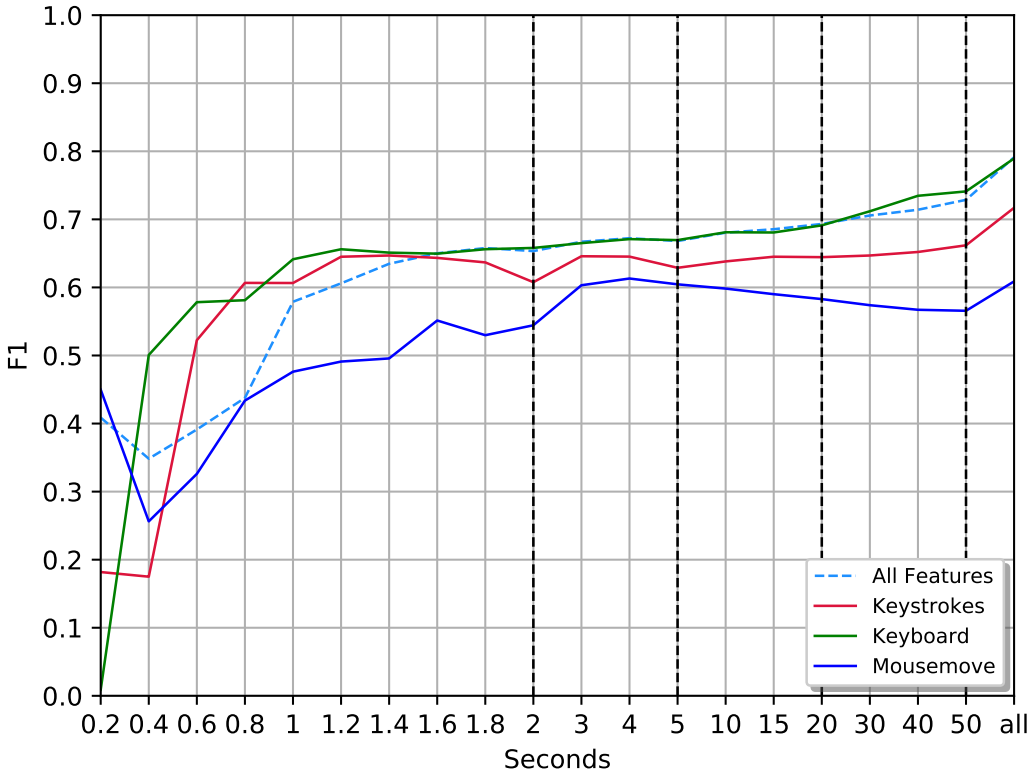


Figure 5.8: Comparison of vandalism detection performance with different feature sets over time.

Chapter 6

Implementation

In this chapter, I describe the implementation of several parts of the thesis which were important or developed specifically for this thesis. In the first section, I elaborate the creation of the dataset of article edits with recorded user interactions. In the last section, I describe the structure of the final dataset in more detail.

6.1 Setup

To be able to create the dataset, I needed a copy of the Wikipedia to let the workers edit articles. I decided not to use a backup dump from Wikipedia to set up my own version, but to use the data directly from the real Wikipedia. In that way, the workers were provided with an exact copy of Wikipedia including all Wikipedia functionality besides article editing.

6.1.1 Server configuration

To be able to track the interaction of the users during their editing, I had to inject the tracking code into the Wikipedia page. Since it is not allowed to add own JavaScript code to foreign web pages, I had to come up with a proper solution for this problem. My final server setup to achieve the tracking of users on a foreign web page is shown in Figure 6.1.

I set up my own forward proxy server which redirects the request (①) of a user (client) to the real Wikipedia (②). The same proxy server is also set up as a reverse proxy; therefore, the answer (③) from Wikipedia is returned by the proxy server to the client (④). In that way, the client is able to fully explore Wikipedia but remains on my own proxy server. In other words, I am able to observe and modify all the communication between the client and Wikipedia

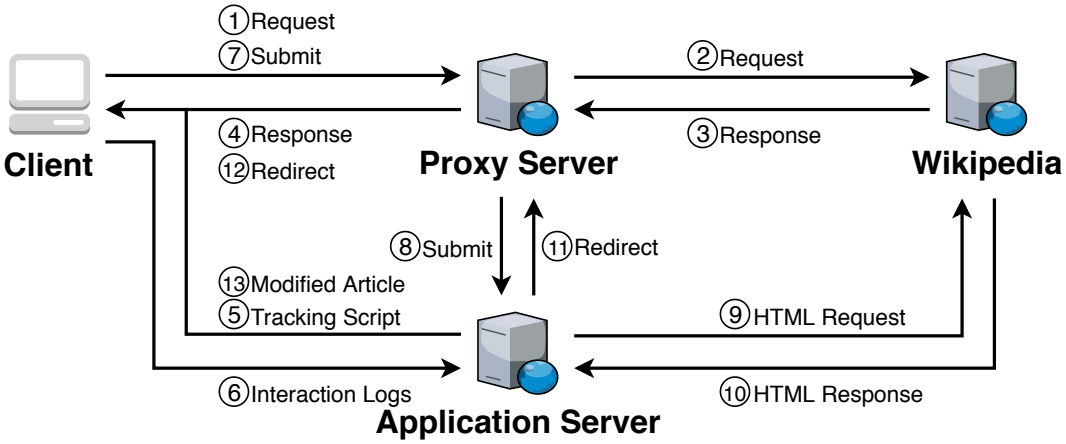


Figure 6.1: Information flow between the different components used to realize the recording of client-interactions.

through the proxy server. The server is run by Apache[1] and is configured to inject the tracking script (⑤), explained in detail in Section 6.1.2, into every requested page of Wikipedia. The injected tracking script records the interactions and properties of the client and sends the result periodically to a second web sever (⑥) which saves these logs. The second web server is not only responsible for receiving the recorded interaction logs, but also for delivering the tracking script itself and the content of the Wikipedia articles modified by the client (explained in Section 6.1.3). With this server setup it is possible to inject the tracking script not only into Wikipedia but into any arbitrary web service. To maintain the data protection of the clients, I prevent the access of any log in or register subpage of Wikipedia since otherwise, I would be able to extract the security credentials of the user from the interaction logs. On any attempt of the client to visit such subpage the proxy server redirects the user to a special subpage which simply states to not enter the credentials. Furthermore, I block any submit of changes to an article on the proxy server (⑦) to prevent the forwarding of those changes to the real Wikipedia; instead, I redirect such requests to the application server (⑧). The application server extracts the Wikitext of the request containing the changes of the user and saves it locally. Additionally, the application server generates a submit request containing the Wikitext to the sandbox of Wikipedia (⑨). As a result, the sandbox returns the main page of the sandbox with the edited article from the user as its content (⑩). The application server parses this response and extracts and saves the article content rendered as HTML. Using this approach, I ensure that the HTML representation of the edited article is exactly the same

[1] https://httpd.apache.org/

as if the article would have been edited on the real Wikipedia. Finally, the application server redirects the submit request of the user to the main page of the edited article (⑪ and ⑫). As described in more detail in Section 6.1.2, the tracking script in the client checks whether the user submitted the article previously by requesting the application server, which returns the edited article rendered as HTML, and replaces the result with the version from the server. If the user visits the edit page of the article, the tracking script requests the application server, receives the edited article as Wikitext and inserts this Wikitext in the input field of the edit page.

6.1.2 Tracking script

I wrote a JavaScript script which can be inserted into any web page and which automatically starts tracking the user's behavior without any manual initialization. The script maintains a queue which holds the recorded interactions and every time a new interaction of the client is recorded, it is added to this queue. The content of the queue is flushed periodically and sent to a given URL. With this behavior it is possible to reconstruct the user interactions on the client side at least partially even if the client closes the web page unexpectedly.

Script Parameters The following properties can be set to adapt the behavior of the tracking script to a specific use case:

HOST
: The script will send the recorded interactions to this URL.

TIME_UPDATE_MAX
: Sets the maximum time interval (in milliseconds) between two flushes of the interaction queue.

SIZE_QUEUE_MAX
: Sets the maximum number of recorded interactions until the interaction queue is flushed and the recorded interactions are sent to the host.

For this thesis I chose a TIME_UPDATE_MAX value of 2000ms and a SIZE_QUEUE_MAX value of 50. With these settings one chunk of interaction log sent from the client to my webserver never exceeds the amount of 50 included interactions and is never sent with a timegap of more than two seconds.

Interaction	Description
Mouse	
Movement	Cursor position in x and y coordinates (pixel)
Button click	Type of pressed mouse button
Wheel	Length of scrolled distance (pixel)
Keyboard	
Key press	Type of pressed key
Key release	Type of released key
Page	
Resize	The window size (pixel) after the browser was resized
Unload	The current page was left

Table 6.1: Tracked interactions of the users. Every recorded interaction is associated with the corresponding timestamp.

Script Behaviour After the script is loaded, it records several properties of the client, like the used browser, the viewport-size, the URL, and the current timestamp, and sends an intial request to the webserver with these properties. This initial event is further called "Start-Event". The tracking-script is designed to record the Start-Event as soon as possible even before the webpage has loaded all assets like images. Once the whole page is loaded another special event, the "Loaded-Event", is recorded. All other events initiated actively by the client are recorded as soon as the "Start-Event" was triggered. Table 6.1 shows the different data and interaction types of the users which are tracked continuously during the visit. Every recorded interaction is associated with the corresponding timestamp allowing the chronological ordering of the interactions to reconstruct the behavior of the user.

6.1.3 Dynamic Articles

All the tasks for the workers required the modification of a given Wikipedia article. To enable the users to make complex interactions with my version of Wikipedia, I had to show the changes made by a user to the corresponding user not only on the final text of the article but also on the edit page itself as Wikitext. The system I created is illustrated in detail in figure 6.2. Every time a worker submitted changes on an article, the Wikitext of the modified version was sent to my web server. On server side, I request the Wikipedia-API[2] with the received Wikitext to get the modified version rendered as HTML source

[2] `https://www.mediawiki.org/wiki/API`

code. Subsequently, I save these two different formats of the modified article to the local storage. Furthermore, I extended the tracking script by a function which is initiated once the script is loaded and the worker is currently on the main page or on the edit subpage of an article. If initiated, the function requests my web server to load the most recent version of the article corresponding to the worker. The server checks whether the worker already did changes to the article by scanning the local storage for saved files corresponding to the worker. If there are already any files from the current worker, the latest version is returned back to the script. On the client side, the returned content of the article is inserted into the current main page of the article as HTML or into the edit page as Wikitext, respectively. This way, the workers always see the real content of the article or their most recent changes if there are any. For the "Spelling Correction" and the "New Fact"-task the original content of the articles initially had to be returned modified to the workers and should contain spelling errors or should miss the fact to be inserted, respectively. For these two tasks, I added another step on the sever side once the server received the request from the script which version to show to the user. Additionally, I prepared for every article and for each of the two tasks the corresponding modified version, one as HTML and one as Wikitext. If the server could not find at least one version of the article already modified by the worker, it looked for this prepared version and loaded the appropriate one. Once the worker modified my prepared version of the article, he automatically is able see his changes.

6.2 Dataset Structure

Every item in the dataset is represented by a Wikipedia article edited by a worker in a specific task. Each item is structured as a folder and contains a metadata file describing the article, a file containing the logged interactions of the worker and the HTML and Wikitext for every version of the article the worker created. The meta data of a corpus item is formatted as JSON and contains the following keys.

id The id of the article is a combination of the id of the assignment and the title of the edited article.

title
The title of the edited article.

id_assignment
The id of the assignment for which the article was edited.

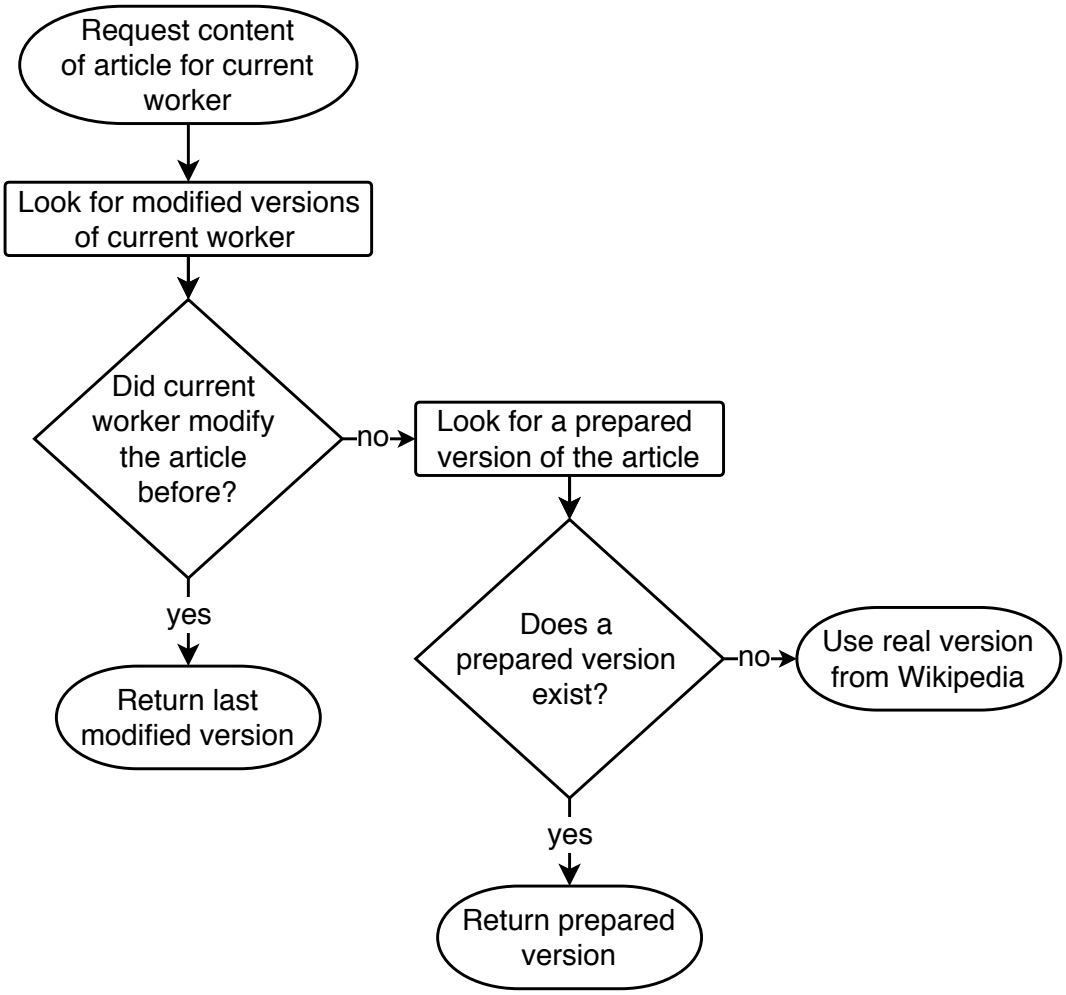


Figure 6.2: This flowchart shows how the appropriate version of an article is chosen if requested.

id_worker
The id of the worker who edited the article.

count_events
The number of all events recorded for this article

count_submitted_files
The number of times the worker submitted his changes.

list_pages
A list of URL-paths the worker visited while editing the article sorted by the time of the visit.

is_vandalism
A boolean value indicating whether the article was edited in a vandalism task or not.

edited_externally
A boolean value indicating whether the worker edited the article on the Wikipedia page directly or in an external application.

type_vandalism
A list of different vandalism types introduced to the article. Possible values are "Deletion", "Sophisticated" and "Gibberish". If the article does not contain any vandalism, the value is "null".

is_complex_edit
A boolean value indicating whether the article was edited for the "New Fact"-task or not. If the article was edited for the "Vandalism"-Task, the value is "null".

The logged interactions are stored in line-delimited JSON with each line representing one recorded interaction. Every recorded interaction contains the following keys.

timestamp
The timestamp at which the interaction was recorded.

type
The name of the recorded interaction for example "start", "mousemove" and "keydown".

data
The payload depending on the interaction type. For example, if the interaction type is "mousemove" the payload contains the x and y coordinates of the cursor.

Chapter 7

Conclusion and Future Work

The findings of this work indicate that it is possible to detect vandalism analyzing the interactions of users only. Although individual users could be detected if necessary, this vandalism detection can be performed in compliance with privacy protection by tracking a small set of interactions only; therefore, the aim should be to compile a new dataset containing interactions from the real Wikipedia to significantly improve the reliability of the results created in this thesis. Additionally, the overall size of the dataset should be increased to be able to feed more feature vectors into the machine learning process, not only to stabilize the detection performance, but also to ensure a good generalization. The machine learning process itself could be further improved without changing the dataset by performing a hyper-parameter optimization for the used machine learning algorithm. In this thesis, the default parameters for the used random forest algorithm were applied without any further investigation of better-performing parameter values. Despite the already promising performance of vandalism detection using behavioral features only, the combination of this new technique with already existing vandalism detection systems will improve the overall vandalism detection process and performance. The feature set used in this thesis contains Wikipedia-independent features only, like the number of pressed keys or the movement of the mouse cursor. Although this allows the model to be adopted for other use cases than Wikipedia, a performance improvement could be achieved by adding Wikipedia-specific features to the feature set. For example, one could measure whether and how an editor leaves an edit comment to the corresponding box on the edit page or how often an editor previews and edits his changes to an article.

Appendix A

Wikipedia Articles

The following sixty articles were used to create the dataset used in this thesis.

- Watchfield
- Ghost town
- Stockspot
- Wilford Power Station
- Mathematical anxiety
- Escape tunnel
- Women in South Sudan
- The North American
- Cowley Wright
- Emergency vehicle lighting
- Rainer Fetting
- East Ward School
- Pangsuma Airport
- Ghalegaun
- Ford Global Anthem
- Stone Creek Jamboree
- Hyundai Getz
- Chen Han
- Benik Afobe
- Survival Sunday
- Humacao Airport
- Gilded flicker
- Richard McKenna
- Vadim Gerasimov
- Thomas Coulter
- Lati Grobman
- HSU First Street Gallery
- 1961 Rebel 300
- Austrian border barrier
- Kyle of Lochalsh line
- Sindh Museum
- Pacific Reporter
- Madison Maersk
- Said Mubarak
- The Gersch
- Luis Saguar
- Dwarfina
- Roman Podolyak
- Jeju World Cup Stadium
- Adolf Hitler Fund of German Trade and Industry
- Paines Plough
- Argentine North Western Railway
- Elizabeth Weiffenbach
- Shatakshee Dhongde
- Tipperary Racecourse
- Margaret Billingham
- Kayo Inaba
- Mass deacidification
- Letter to Blanchy
- Franz Sala
- David Aronson
- Debug code
- Belfast Marathon
- Men of Porn
- Anini
- North Carolina Blumenthal Performing Arts Center
- Fife power station
- Fisher Building
- Active Worlds
- Crotched Mountain

Bibliography

B. Thomas Adler, Luca de Alfaro, Santiago M. Mola-Velasco, Paolo Rosso, and Andrew G. West. Wikipedia vandalism detection: Combining natural language, metadata, and reputation features. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, pages 277–288, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19437-5. 2.1

Enrique Alfonseca, Guillermo Garrido, Jean-Yves Delort, and Anselmo Penas. Whad: Wikipedia historical attributes data. *Language Resources and Evaluation*, page 28, 2013. 2.1

Richard Atterer, Monika Wnuk, and Albrecht Schmidt. Knowing the user's every move: User activity tracking for website usability evaluation and implicit interaction. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 203–212, New York, NY, USA, 2006. ACM. ISBN 1-59593-323-9. doi: 10.1145/1135777.1135811. URL `http://doi.acm.org/10.1145/1135777.1135811`. 2.2

Pnina Fichman and Noriko Hara. Beyond vandalism: Wikipedia trolls. 36: 357–370, 05 2010. 2.1

Iftikhar Ahmed Khan, Willem-Paul Brinkman, Nick Fine, and Robert M. Hierons. Measuring personality from keyboard and mouse use. In *Proceedings of the 15th European Conference on Cognitive Ergonomics: The Ergonomics of Cool Interaction*, ECCE '08, pages 38:1–38:8, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-399-0. doi: 10.1145/1473018.1473066. URL `http://doi.acm.org/10.1145/1473018.1473066`. 2.2

Santiago M. Mola-Velasco. Wikipedia vandalism detection through machine learning: Feature review and new proposals: Lab report for pan at clef 2010. 10 2010. 2.1

Barbara Plank. *Keystroke Dynamics for Authorship Attribution*. Tributes. College Publications, 2016. ISBN 9781848902305. 2.2

Barbara Plank. Predicting authorship and author traits from keystroke dynamics. pages 98–104, 01 2018. 2.2, 4.2.2

Martin Potthast and Teresa Holfeld. Overview of the 2nd International Competition on Wikipedia Vandalism Detection. In Vivien Petras, Pamela Forner, and Paul D. Clough, editors, *Notebook Papers of CLEF 11 Labs and Workshops*, September 2011. ISBN 978-88-904810-1-7. URL `http://www.clef-initiative.eu/publication/working-notes`. 2.1

Martin Potthast, Benno Stein, and Robert Gerling. Automatic Vandalism Detection in Wikipedia. In Craig Macdonald, Iadh Ounis, Vassilis Plachouras, Ian Ruthven, and Ryen W. White, editors, *Advances in Information Retrieval. 30th European Conference on IR Research (ECIR 08)*, volume 4956 of *Lecture Notes in Computer Science*, pages 663–668, Berlin Heidelberg New York, 2008. Springer. ISBN 978-3-540-78645-0. doi: http://dx.doi.org/10.1007/978-3-540-78646-7_75. 2.1

Martin Potthast, Benno Stein, and Teresa Holfeld. Overview of the 1st International Competition on Wikipedia Vandalism Detection. In Martin Braschler, Donna Harman, and Emanuele Pianta, editors, *Working Notes Papers of the CLEF 2010 Evaluation Labs*, September 2010. ISBN 978-88-904810-2-4. URL `http://www.clef-initiative.eu/publication/working-notes`. 2.1

Maja Pusara and Carla E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC '04, pages 1–8, New York, NY, USA, 2004. ACM. ISBN 1-58113-974-8. doi: 10.1145/1029208.1029210. URL `http://doi.acm.org/10.1145/1029208.1029210`. 2.2

Vinitha Venugopal Savithri, Gergő Tisza, and Adam Wight. Spambot detection via registration page behavior, 2018. URL `https://meta.wikimedia.org/wiki/Research:Spambot_detection_via_registration_page_behavior`. 2.2, 4.2.3, 4.2.3

B Thomas Adler, Luca de Alfaro, and Ian Pye. Detecting wikipedia vandalism using wikitrust: Lab report for pan at clef 2010, 01 2010. 2.1

Andrew West, Sampath Kannan, and Insup Lee. Stiki: An anti-vandalism tool for wikipedia using spatio-temporal analysis of revision metadata. 07 2010. 2.1

Brit Youngmann and Elad Yom-Tov. Anxiety and information seeking: Evidence from large-scale mouse tracking. In *Proceedings of the 2018 World*

Wide Web Conference, WWW '18, pages 753–762, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-5639-8. doi: 10.1145/3178876.3186156. URL `https://doi.org/10.1145/3178876.3186156`. 2.2

Nan Zheng, Aaron Paloski, and Haining Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 139–150, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046725. URL `http://doi.acm.org/10.1145/2046707.2046725`. 2.2, 4.2.2