A Study on the Evolution of the Dutch Web

Daan Kooij
Computer Science
April 26th, 2022

MSc Graduation Project
Supervised by Doina Bucur
and Mattijs Jonker

Data Management & Biometrics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Acknowledgements

First of all, I would like to thank my main supervisor, Doina Bucur. Over the past year, she supervised my project, sat with me through numerous meetings, and provided my drafts with lots of feedback. I would also like to thank my second supervisor, Mattijs Jonker. Even though he only joined the project a bit later, he gave me a number of ideas and suggestions for improving my methodology, as well as useful feedback for improving my green light draft.

Next, I would like to thank the members of my study group. Especially Bas Bleijerveld, who sat with me in the University library for hundreds of hours, but also Wouter Bos, Ivan Hop, Erik Kemp, and Jan Boerman. They all helped me stay motivated throughout the project, all the while providing the (much needed) social pressure not to slack off.

In addition, I am infinitely grateful to my parents! My mother, Liesbeth Carpai, was always there for me to release my thoughts to. She was my go-to moral support hot-line throughout the whole project, and managed to stay amazingly patient even when I did not meet my goals for the hundredth time. And my father, André Kooij, really helped me get started near the beginning of the project, (almost) always asked just the right questions, and frequently helped me put things back into perspective.

I am also thankful to my roommate, Peter Klein Kranenbarg, who was one of the most constant factors in my life during the lockdowns. He was the victim of my daily progress (or lack thereof) updates, and often reminded me of the fact that the project was finite. He even voluntarily read through the whole of my green light draft, providing me with useful suggestions for improvements, as well as confidence in the quality of the draft.

Finally, I want to thank two friends in particular. Justin Praas, for keeping faith in me throughout the whole project, and for having regular Mario Kart marathon sessions with me as a much-needed distraction. And Anna Dankers, for always being interested in what was going on, and for remaining optimistic regardless of the situation.

I am really grateful to have all these amazing people around me. Even though they might not have experienced it as such, they were an important part of my life over the past year. Thank you! :)

Abstract

Search engines need to have an up-to-date view of the Web, but Web crawling resources are limited, meaning that not all pages can be crawled continuously. With this research, the evolution of the Dutch Web is studied, having the ultimate goal of generating insights that can be used to optimise the Web crawlers of search engines.

As part of this research, daily crawls are performed on a large number of pages from the Dutch Web, using a custom-built Web crawler that circumvents cookie walls. This results in a novel high-quality dataset of the Dutch Web, which is used to carry out a large-scale study on the evolution of the Dutch Web. This study includes an investigation of change types, the discovery of temporal change patterns, and the composition of a predictive Machine Learning model that can predict whether the text on pages will change.

The main conclusions are the following. First, we discovered that over two-thirds of the pages on the Dutch Web undergo some kind of change in a given 24-hour period, but that only just over half of these changes are informative, with an even lesser fraction reflecting content being purposefully updated by human beings. Second, we identified that change frequencies differ significantly between different page features, but that all of them follow a weekly repeating pattern. Moreover, we found that the change rates of most pages are stable. Finally, we trained two types of Machine Learning models to predict whether page text will change in the coming 24 hours. The highest performing one is a Random Forest model, resulting in a *minimum recall* of 78.48%. By explaining the model's predictions and analysing prediction errors, we gained insights into the relationships between page characteristics, change likelihood and prediction accuracy.

The insights gained by performing this study can be used by search providers to optimise the Web crawlers of their search engines, for example by focusing crawling resources on pages that tend to change often. Also the trained predictive model can itself be used to guide the crawling processes of Web crawlers, by prioritising crawling the pages for which the model predicts that the text will change.

Contents

Chapter 1

Introduction

After being conceived in 1989, the World Wide Web has played an increasingly important role in our daily lives. In the early years, it was a place for information-sharing between scientists in universities and institutes around the world [11]. But only a few years later, in 1993, the Web was released to the general public, and websites for general use started to become available. Back then, it was mostly a place for businesses and technology enthusiasts who could afford the expensive hardware.

Ever since the early nineties, the Web has enjoyed continuous growth, with no sign of stopping anytime soon. Over half of the global population ($59,5\%$) is actively using the internet as of January 2021 [23], and 97% of Dutch households had an internet connection in their home in 2020 [16]. However, nowadays the Web no longer is merely an addition to our lives—it has become a requirement. A lot of institutions and businesses moved completely online, and amidst the COVID-19 pandemic we became even more reliant on the Web [14]. But not only our contact with businesses moved online. Also our contact with the government moved online (over 80% of the adult population in the Netherlands interacts with public authorities over the internet [31]), as well as our contact with social circles (the more a person uses the internet, the less (face-to-face) contact they have with their social environment [30], but also the more they use e-mail and instant messaging services [22]).

Problem statement With the Web becoming an increasingly important part of our lives, we need a way to navigate this seemingly infinite jungle of content. In the present day, people rely on search engines to find their desired content on the Web [25]. But before a search engine can serve the appropriate page to a user, the page needs to be crawled in order for the search engine to be aware of its content [7]. However, crawling resources are limited, so not all pages can be crawled continuously [43]. Instead, decisions need to be made at what order and intervals to crawl Web pages [43].

Relevance By studying the evolution of the Web, we can gain insights into how the Web changes, which can be used to optimise the crawling processes of search engines

by better allocating crawling resources (prioritising pages that tend to change often). This can provide better (more up to date) search results to users, as well as decrease power consumption by search providers (because they can refrain from recrawling a page if they are certain that it will not change). Furthermore, search engines can also use these insights to discover new (previously uncrawled) pages, by focusing crawling resources on pages that tend to gain new outgoing links.

Topic Over the years, a lot of research has been done on studying the evolution of the Web, both on the Web as a whole and on more specific subsets of the Web (see Section 2.5). However, no research has been done yet on studying the evolution of one specific subset of the Web: the Dutch Web. In this research, we study this particular subset of the Web. We do this by performing daily Web crawls over a period of ninety days on over 150.000 Web pages in the *.nl* domain, whose URLs are extracted from the Dutch Wikipedia. The crawls are performed by our custom-built Web crawler program, which circumvents "cookie walls" in an age post-GDPR [15]. This results in a high-quality dataset including the content of pages hidden behind cookie walls. Using this dataset, we investigate the type of changes that pages undergo, study change timelines, and build a predictive model that can predict whether the text on pages will change.

Objectives The main objectives of this study are the following. First, it is to chart out the type of changes that pages on the Dutch Web undergo. Next, it is to discover temporal change patterns of pages on the Dutch Web. And finally, it is to compose a predictive Machine Learning model that can predict whether the text on pages in the Dutch Web will change, which is then used to explain how predictions are made (which gives us insights into the relationship between page characteristics and change likelihood) and why prediction errors are made (from which we can learn the characteristics of pages for which it is hard to make correct predictions). This leads to the following three research questions:

RQ1 What are the type of changes that pages on the Dutch Web undergo?

- **RQ1.1** What is the fraction of pages on the Dutch Web that undergoes any change?
- **RQ1.2** For the pages on the Dutch Web that undergo any change, what is the distribution over change types[1]?
- **RQ1.3** Out of the pages on the Dutch Web that change, what fraction contains potentially informative changes[2]?

[1]Possible change types are changes in page text, scripts, internal out-links, external out-links, email addresses, images, tables, meta elements, and HTML tags. Here, "internal out-links" are defined as hyperlinks to pages within the same second-level domain, and "external out-links" are defined as hyperlinks to pages outside the domain.

[2]"Informative changes" are defined as changes that contain useful information, either for users (such as changes in page text or images) or for Web crawlers (such as changes in out-links). See Section 4.1.

- **RQ1.4** Whenever the readable text on a page on the Dutch Web changes, in what way does it change?
- **RQ1.5** Whenever a feature on a page on the Dutch Web changes, by how much does it change[3]?

RQ2 What are the temporal change patterns of pages on the Dutch Web?

- **RQ2.1** What are the daily page change frequencies of pages on the Dutch Web?
- **RQ2.2** Do the page change frequencies of pages on the Dutch Web follow specific temporal patterns?
- **RQ2.3** To what degree do page change frequencies of pages on the Dutch Web differ across different days?
- **RQ2.4** To what extent do pages on the Dutch Web keep exhibiting the same change behaviour?

RQ3 Using a collection of static page features, how well can we train a Machine Learning model to predict whether the text on pages in the Dutch Web will change?

- **RQ3.1** What type of Machine Learning model is appropriate for this case?
- **RQ3.2** How can we optimise the performance of our selected Machine Learning model?
- **RQ3.3** Which features are most important for training the Machine Learning model?
- **RQ3.4** How well does the trained Machine Learning model perform?
- **RQ3.5** How does the trained Machine Learning model make predictions?
- **RQ3.6** What are the limitations of our Machine Learning methodology?
- **RQ3.7** How can the errors made by the trained Machine Learning model be explained?

Contribution The main novelty of this research is that it is the first work to study the evolution of the Dutch Web. It is also the first work that uses a dataset including the content of Web pages hidden behind their cookie walls, meaning that the investigations are performed on high-quality data. More specifically, the research consists of three main components:

[3]Possible features are page text, scripts, internal out-links, external out-links, email addresses, images, tables, meta elements, and HTML tags.

1. We chart out the type of changes that pages on the Dutch Web undergo. This includes an investigation of: the fraction of pages that change, the distribution of page changes over change types, the fraction of page changes that are informative, the changes of page text, and the change amplitudes of page changes.
2. We discover temporal change patterns of pages on the Dutch Web. This includes an investigation of: change frequencies, the rarity of change frequencies, the stability of change frequencies, and temporal change patterns.
3. We compose a predictive Machine Learning model that can predict whether the text on pages in the Dutch Web will change. This includes an evaluation of the model's performance, a feature importance ranking, an explanation of the model's predictions, a description of the model's limitations, and an analysis of the model's prediction errors.

All three components can be considered a contribution to the research field, because they are for the first time carried out specifically for the Dutch Web, and are performed on a novel high-quality dataset including the content of pages hidden behind cookie walls. Moreover, the third component can be considered a contribution in two additional ways. First, the model itself can be used by Web crawlers to guide their crawling processes. For example, a crawler can prioritise the crawling of pages for which the model predicts that their text will change. And second, the model can be used to derive further insights: by investigating the model, we can explain how predictions are made (which gives us insights into the relationship between page characteristics and change likelihood) and why prediction errors are made (from which we can learn the characteristics of pages for which it is hard to make correct predictions).

Repository As part of this research, we develop a number of programs[4]. First, we develop a tool that extracts links from a Wikipedia dump. Second, we develop a Web crawler which is used to perform daily crawls of more than 150.000 Web pages for ninety consecutive days. Finally, we develop a collection of analyser tools that are used to analyse the crawled Web pages, as well as to generate the figures in this thesis.

Outline The remainder of this paper is structured as follows. In Chapter 2, we provide the background context for the study, as well as take a deep dive into the work that has already been done in relation to the topic. In Chapter 3, we describe our data collection in detail. Then, in Chapters 4-6 we answer research questions 1-3 respectively. Afterward, in Chapter 7 we discuss to what extent we answered the research questions. Finally, in Chapter 8 we conclude the paper by summarising our main findings and giving suggestions for future work.

[4]The programs developed as part of this research are available at `https://github.com/dkooij/scm`.

Chapter 2

Background & Related Work

Over the years, already quite some work has been done on investigating Web dynamics and predicting Web change patterns, both on general and more specific datasets. Previous work has shown that the change frequency of Web pages is correlated with page features, and can often be predicted using these features (Section 2.1). Another branch of research investigates the predictive power of a Web page's change history for predicting its change frequency, with numerous papers showing that the two are correlated (Section 2.2). Some papers also rely on Machine Learning approaches to predict page change frequencies and other page properties (Section 2.3). There has also been a lot of work done on the investigation of Web dynamics, such as the frequency of page changes, the stability of this frequency, and the velocity at which new pages emerge and die (Section 2.4). Finally, different kinds of datasets are used: some are more general and try to capture the Web as a whole, while others are more specific and try to capture a specific subset of the Web (Section 2.5).

2.1 Page Features

The change frequency of Web pages is shown to be correlated with page features [1, 4, 18, 28, 33, 37, 39, 40, 43], and can often be predicted using these features. A feature can be either *static* or *dynamic*. In order to retrieve a static feature, only a single crawl is required. On the other hand, a sequence of crawls is required to retrieve a dynamic feature. In addition, a feature can be classified as either *local* or *network*—the former can be computed for a Web page using only that Web page, while the latter needs the network (or graph) of Web pages.

Choice of feature types In our research, we build on the knowledge from these works and mostly make use of features that can be computed for a single version of a page (static features) (see Chapter 3). In Chapter 4, we also use dynamic versions of these same features, to investigate by *how much* the features change whenever they change. Similarly, in Chapter 5, we use three dynamic features (change in page text, change in

internal out-links, and change in external out-links) to gain insights into how frequently these features change. Finally, in Chapter 6, we use one dynamic feature (change in page text) as target value to train Machine Learning models that predict whether the text of a given page will change in the coming day. However, in contrast to the works of Radinsky and Bennett [37] and Tan and Mitra [43], we only consider local features, and do not consider network features. This is because with our crawling method, we crawl a collection of separate Web pages that are not necessarily connected to each other with links, and as such we are unable to compute the number of in-links. Nonetheless, not using network features makes predictive models more easily applicable in practice, because it allows for making predictions for a single page without crawling a big portion of the Web.

2.1.1 Local Features

Static local features Any combination of these two feature categories is possible. The most straightforward approach (computationally) is to only make use of static local features [1, 4, 18, 39, 40]. An example of a static local feature is the number of out-links (hyperlinks) to other Web pages.

Categorical change rate In Barbosa et al. [4], an algorithm is developed that predicts the (categorical) change rate of a page based on its content when it is first accessed, and uses the historical information to predict change rate at subsequent accesses. Observe, however, that Barbosa et al. do not use dynamic features to predict page change rates—in contrast, they use the history of change to improve predictions (see Section 2.2). Their classifier uses several static features, including the number of links, number of e-mail addresses, existence of "LAST-MODIFIED" headers, file size in bytes (without HTML tags), number of images, and depth of a page in its domain. In contrast to other papers [1, 18, 39], however, Barbosa et al. conclude that the latter has little to no correlation with the change rate [4].

Correlation with static local features Fetterly et al. also make use of static local page features [18], and they observe that the larger a Web page (in bytes or in words), the more likely it is to change. As part of our research, we make the same observation (see Section 6.2). However, Fetterly et al. also discover that this effect is stronger for *.com* and *.net* domains than it is for *.gov* and *.edu*. They then draw the conclusion that the page size and top-level domain have an independent influence on the change rate of Web pages. Saad and Gançarski investigate how change patterns can be used to improve the effectiveness of Web archives [39]. One of their main findings is that the change rate of Web pages decreases significantly as URL depth increases. In addition, Santos et al. find that topic-specific refreshing strategies can be beneficial for focused crawlers [40], as page topic is an important predictor for change frequency. Furthermore, they discovered that the majority of pages change either very often or very rarely.

Finally, Adar et al. draw similar conclusions, and confirm that page change frequency is correlated with top level domain, URL depth, and page topic or category [1]. In addition, they remark that change frequency is also correlated with page popularity, which is defined as a function of visitor count (an external feature).

Dynamic local features To compute dynamic page features, multiple crawls of a single Web page are required, as dynamic features compute some kind of difference between two (or more) versions of the same Web page. An example of a dynamic local feature is the number of new out-links (hyperlinks) to other Web pages.

Prediction with random forest Meegahapola et al. crawl pages in fixed specified time intervals between 4 and 72 hours, and save for each of the consecutive intervals three types of dynamic local features: number of new elements, number of modified elements, and number of removed elements [28]. They then feed these features into a random forest Machine Learning model, which they train to predict change frequencies of Web pages (see Section 2.3).

Relation with information longevity Olston and Pandey investigate the relationship between change frequency and information longevity [33], where information longevity is defined as the average lifetime of fragments on a page. The authors do not detect a clear correlation between the two, however, they note that information longevity is still a key factor in crawler effectiveness, as well as an important metric to measure the effectiveness of crawls.

2.1.2 Network Features

Static network features The approach to use network features in addition to local ones is computationally more expensive. This is because in order to compute network features, the complete network (or graph) of Web pages is required, whereas local features can be computed using only the designated Web page. The most common example of static network feature is the number of in-links of a Web page.

Dynamic network features It is also possible to use both dynamic features and network features [37], or even features that can be classified as both dynamic and network [43]. An example of a dynamic network feature is the number of new in-links of Web page [43].

Expert prediction framework Radinsky and Bennett present an expert prediction framework [37], which uses static page features (among others) to improve prediction accuracy. With this approach, certain dynamic features (temporal content similarity) and network features (Web graph distance) are also used to select “experts”, which in turn provide signals that informs the framework’s prediction. However, these dynamic

and network features are used solely to select experts according to their degree of relatedness, and not directly as page features.

Change frequency clusters Tan and Mitra propose a crawling algorithm where Web pages are divided over clusters with similar change frequencies [43]. The clusters are ranked, where clusters with high change frequencies are ranked higher than clusters with low change frequencies. Their crawling algorithm then downloads a sample of Web pages from the highest-ranked cluster, and checks whether the downloaded pages in the cluster have changed. If a significant number of pages have changed, the complete cluster is re-crawled. Otherwise, the algorithm proceeds to the next highest ranked cluster. The clustering algorithm uses a lot of features: 36 static local features, 22 static network features[1], seven dynamic local features, and two dynamic network features (change in PageRank and change in in-links).

2.2 Change History

Another branch of research investigates the predictive power of a Web page's change history for predicting its change frequency, and numerous papers show that the two are correlated [2, 4, 9, 12, 13, 19, 20, 24, 37, 41, 42]. The change history can be distinguished from other dynamic page features in the sense that it is generally considered separately as a time series, with (often mathematical) models being developed to model its behaviour. For every consecutive pair of a page's versions in the time series, a *change value* is computed, which either denotes the binary event of change by comparing the page checksums or content digests [4, 12, 13, 24], or the degree of change [9, 19, 37]. Nevertheless, the degrees of change are often transformed into binary values by comparing them to threshold values. Besides, not all papers specify when they consider a page to be changed.

Relation to our research Although in our research we are not trying to predict the change rate of a page, we do something related. In Chapter 5, we compute the fraction of pages that changes per day to explore the fraction of pages that changes over time, as well as investigate the evolution, rarity, and stability of page change frequencies.

Partial information Avrachenkov et al. investigate how page change rates can be estimated using only partial information about page change processes [2]. They propose two approaches for online estimation of page change rates: one based on the Law of Large Numbers, and the other derived using Stochastic Approximation principles. Their results show that (together with the already existing Maximum Likelihood Estimation (MLE) method) the two proposed estimators perform consistently better than a Naive

[1] Out of the 22 static network features, twenty are *ranking* features, which are computed by ranking pages according to twenty popular search keywords.

estimator, and that the two estimators are more computationally efficient than the MLE estimator.

Categorical change rate In Barbosa et al. [4], an algorithm is developed that predicts the (categorical) change rate of a page based on changes between subsequent visits to the same page (see Section 2.3). They detect changes between visits by comparing MD5 checksums.

No underlying change model Calzarossa and Tessera represent Web page content changes as periodic time series [9], where the daily and weekly change patterns are captured by seasonal and trend components of the series. They use numerical fitting techniques to identify the parameters of trigonometric polynomials that best fit these components, and ARMA models to fit the irregular components of the time series. These models represent a basis for forecasting (i.e., predicting future dynamics based on current and historical behaviour), and as such Calzarossa and Tessera make predictions without using an underlying model of change (such as the often-assumed Poisson process model, see Section 2.2.1).

Optimal crawl schedule The main aim of Gupta et al. is to find the optimal crawling schedule that maximises the accuracy of information downloaded for a given crawling frequency [20], where accuracy is defined as the percentage of information captured by a given crawling plan with respect to the information captured by a baseline crawling plan (in this case, crawling every page twice an hour). They approach this using Operations Research methods, using mixed integer programming to discover the optimal crawling schedule. They also look into how the crawling frequency and the corresponding crawling schedule can be optimised for a given accuracy level, which they solve using a greedy strategy.

Selecting expert pages Radinsky and Bennett use the change history of a page to select related "expert" pages that can inform their prediction framework [37]. They put specific focus on temporal content similarity, which is explored through the use of cross-correlation and the dynamic time warping algorithm.

Genetic programming Santos et al. use a different approach: they introduce the GP4C (Genetic Programming for Crawling) framework [41], which generates score functions that produce rankings of pages, ordered by their probabilities of having been modified (see Section 2.3). The score functions are learnt from only the binary history of page change: for every day that a page is crawled, it is recorded whether it did or did not change with respect to the previous day.

2.2.1 Poisson Process Model

There is a general consensus in research that the change frequency of Web pages follows a discrete Poisson distribution [12, 13, 24, 42].

Estimating λ In Cho and García-Molina [13], the homogeneous Poisson process model is assumed, and it is found that the change rate λ can be estimated statistically out of an incomplete and irregularly sampled history of change for an individual page. Different change rate estimators are developed for different applications requiring different accuracies, and it is observed that the estimators work well even if the underlying change model is not "quite Poisson". In later work by the same authors [12], this result is used to propose refresh policies that allow us to maintain fresh copies of remote data sources (Web pages) when the source data is updated independently. These policies are examined on effectiveness, both analytically and experimentally (by running simulated crawls), and are shown to improve the freshness[2] of data significantly. It is found that freshness is maximised when policies consider how often a page changes and how important the page is. However, although it is stated that a Poisson process model is a good model to describe changes of Web pages, Web sites were (only) crawled on a daily basis for a period of four months, and hence the model is not verified for pages that change either very often (more than once a day) or very rarely (less than once every four months).

Reinforcement learning Kolobov et al. also address the freshness crawl scheduling problem by modelling the change process of information sources as Poisson processes, but instead they learn the change rates using a Reinforcement Learning algorithm [24]. As is also the case in the works by Cho and García-Molina [12, 13], Kolobov et al. consider pages changed once their digests (checksums) have changed.

Homogeneous Poisson As a variant on the homogeneous Poisson process model used by Cho and García-Molina [13], Singh models the processes of Web page updates as time-varying (non-homogeneous) Poisson processes [42], and focuses on determining localised update rates[3]. This approach estimates localised update rates by dividing a sequence of independent and inconsistent update points into consistent windows, and the results show that the proposed Weibull estimator outperforms other estimators. Singh then predicts future update points based on the most recent window, again yielding the highest performance using the Weibull estimator.

Challenging Poisson Grimes and O'Brien challenge the Poisson process model [19], and investigate whether it genuinely represents changes for rapidly changing pages.

[2] A collection of data is considered "fresher" when it has more up-to-date elements [12].

[3] The difference between a time-varying and a time-homogeneous Poisson process is that with the former, the rate of updates may vary over time, while with the latter, the rate of updates stays the same for a given page.

They conclude that only few pages are strictly consistent with a Poisson model, but also that only a small portion differs significantly.

2.3 Machine Learning

Some papers rely on Machine Learning approaches to predict page change events or frequencies [4, 24, 28, 41, 43], or to predict other page properties [34, 40]. Machine Learning also plays a central part in our work: we train two different types of Machine Learning models to predict whether the text on a given Web page will change. We also evaluate the performance of the models, compose feature importance rankings, provide explanations for model predictions, describe model limitations, and analyse model prediction errors (see Chapter 6).

2.3.1 Predicting Change Frequencies

Clustering In the work of Barbosa et al., the regression task of learning page change rates is turned into a classification task by discretising the target attribute (i.e., the change rate of a page) [4]. Barbosa et al. train the Machine Learning model by initially (when no change history is present) basing the prediction solely on the page's content, but in subsequent accesses basing the change rate on historical information. By quickly adapting to the observed change history, this method derives accurate predictions. It achieves the (unsupervised) discretisation of target values by using the *k*-means clustering algorithm (using an estimate of page change rate), where each of the four clusters is undersampled so that they have equal cardinalities. Further, if during classification it is discovered that a page does not belong to a given cluster, is is moved to a lower or higher one. In our work, although our Machine Learning methodology is different (we train a model to perform the binary classification task of predicting whether the text on a page will change using only static page features), we do employ a similar undersampling approach: we undersample our training data such that the prior probabilities of both classes are equal (see Section 6.1). Tan and Mitra use a similar approach as Barbosa et al., and propose a crawling algorithm which clusters together Web pages with similar change frequencies [43]. At crawl-time, their approach downloads samples of pages from a given cluster (prioritising clusters with higher change frequencies), in order to determine whether it is worth re-crawling the complete cluster (see Section 2.1).

Reinforcement learning Kolobov et al. present a model-based reinforcement learning algorithm which converges to an optimal crawl policy [24]. This algorithm uses the proposed "natural freshness optimisation objective", which is based on harmonic numbers, and for which it is shown how its mathematical properties enable efficient optimal scheduling. The algorithm combines this objective with model estimation from Cho and García-Molina [13], with as novelty that this combination lifts the known-parameter

assumption. Apart from this optimal algorithm, Kolobov et al. also introduce an approximate crawl scheduling algorithm, which requires learning far fewer parameters.

Random forest Meegahapola et al. feed dynamic page features into a random forest Machine Learning model [28], which is trained to predict the "loop-time basket" (categorical change frequency) to which the page belongs. Using 20% of the data as test data, they show that the resulting classifier has a precision of approximately 88%.

Genetic programming In Santos et al., the GP4C (Genetic Programming for Crawling) framework is proposed [40]. This is a Machine Learning based approach using genetic programming. The framework generates score functions to produce rankings of pages, ordered by their probabilities of having been modified. The score functions are learnt only from the binary history of page change, and performance is evaluated using two metrics (called "ChangeRate" and "NDCG"). The metrics serve a dual purpose: not only to be used for evaluation, but also as objective functions (to be maximised in the learning process). Santos et al. show that the NDCG metric is better for evaluating the effectiveness of crawling strategies, since it is able to take the rankings of pages into account.

2.3.2 Predicting Other Properties

Machine Learning approaches are not only used for predicting page change frequencies, but also for predicting other page properties.

Page status Pant and Srinivasan use a regression model to predict page status (which is defined as the number of in-links of a page), using as features aspects local to a Web page [34]. They find that a page is more likely to have a lower status if it is lower in a directory structure, or if it is more specific in content. In contrast, they find that the page status increases as the number of out-links of the page increases. They observe that topic-specific models are better at predicting page status than generic models, and that non-linear models (in particular neural networks and decision trees) yield a better performance than linear regression models. However, the authors also note that linear models are more transparent, and hence potentially provide us with a better understanding of page status as a phenomenon.

Topic-specific change patterns Applying Machine Learning in a different setting altogether, Santos et al. train a support vector machine classifier to recognise pages of two topics (ebola and movies), in order to investigate whether pages in different topics have different change patterns [40]. They find that page topic is indeed an important predictor for change frequency, as is explained in Section 2.1.

2.4 Web Dynamics

A lot of work has been done on the investigation of Web dynamics [1, 3, 9, 18, 19, 20, 32, 40]. This research thread considers issues such as how often pages change, whether the change frequency of pages is stable over time, and at what velocity new pages emerge and die. A core part of our work also focuses on Web dynamics. In particular, in Chapter 4 we look into in the way that a Web page changes whenever it changes, and in Chapter 5 we explore the periodic patterns and time-dependent behaviour of Web pages, as well as the stability and predictability of Web page changes.

Change velocity With their work, Adar et al. aim to provide more insights about the nature of fine-grained (short interval) changes to Web pages that are being actively consumed by users with different visitation patterns [1]. As their dataset consists of pages from the observed Web (that is, pages that are actually being visited by users), the results differ from previous works on randomly sampled pages: change rates are found to be higher in this behaviour-based sample, with a large portion of pages changing more than hourly. Grimes and O'Brien also look into change frequencies by investigating hourly versions of pages that change at least once every 48 hours [19]. They find that out of these pages, only less than 5% change in every fetch. Furthermore, they estimate that up to 25% of the pages change at least once an hour on average, and over 50% change at least once every four hours on average. However, by restricting themselves to pages that change at least every two days, the results of Grimes and O'Brien cannot be generalised to the Web as a whole. Finally, Santos et al. establish that the majority of pages change either very often or very rarely [40], which matches our own observations (see Section 5.1 and Section 5.2).

Periodic patterns Calzarossa and Tessera look into how to model and predict the dynamics of Web content changes [9]. They find that page change patterns are defined by large fluctuations, coupled with some periodicity and time-dependent behaviour. In particular, they observe that weekday and time of day are correlated with the page's change frequency. The results of our study support this observation (see Section 5.1). Also the work of Fetterly et al. yields similar results: one of the contributions of Fetterly et al. is an investigation of Web change patterns [18]. Their main finding is that for most Web pages, the change rate is very stable, meaning that the amount of week-to-week change is predictable. With our research, we detect the same phenomenon (see Section 5.1 and Section 5.2). Furthermore, Fetterly et al. conclude that past changes of a Web page are a good predictor for future changes. Also in the work of Gupta et al. about finding optimal crawling schedules, time-dependent behaviour is observed [20]. In particular, they find that fewer new Web pages (in this case: news articles) are created on weekends compared to weekdays.

Cycle of life In their research about Web decay, Bar-Yossef et al. find that pages on the Web do not only emerge rapidly, but do also die quickly (the pages are short lived) [3]. They find that entire neighbourhoods of pages in the Web graph exhibit significant decay, and that even large subgraphs of the Web can decay rapidly. Presented at the same conference, Ntoulas et al. have similar findings [32]. They also detect a high birth and death rate of Web pages, with an even higher change in hyperlinks that connect them. According to their experiments, the birth rate of pages is computed to be 8% per week, and the death rate is estimated to be approximately 20% per year. For the hyperlinks that connect the pages, these rates are even higher, with a birth rate of 25% per week, and 80% of all links being removed or replaced after a year.

2.4.1 Change Nature

One specific aspect of Web dynamics is the nature of changes done to Web pages. In particular, the longevity of information is characterised [1, 33] and the evolution of pages is mapped out [18, 32].

Information longevity In one part of their research, Adar et al. aim to characterise the nature of changes to Web page content and structure [1]. They perform a detailed content and structure analysis to identify stable stable and dynamic content within each page. Olston and Pandey do something comparable: they make a distinction between persistent and ephemeral (changing) content [33].

Page evolution In their research about the evolution of link structure over time, Ntoulas et al. discover that new pages emerging on the Web have a high tendency to "borrow" (copy) their content from other, already existing pages [32]. And when Web pages change, Fetterly et al. find that they usually only change in their markup or in other trivial ways [18]. Our work, tailored to the Dutch Web, supports this finding (see Section 4.1).

2.4.2 Quality Aspects

Another facet of Web dynamics touches upon quality aspects of Web pages, such as page relevance [17, 33, 40, 43], page status [34], and Web decay [3].

Web decay Bar-Yossef et al. attempt to increase our understanding of the Web decay phenomenon[4] [3]. In particular, they investigate how we can determine whether a Web page has decayed, and how widespread Web decay is. They find that pages on the Web do not only emerge rapidly, but do also die quickly, and that large subgraphs of the Web can decay swiftly. They arrive at this conclusion by introducing a decay score which can

[4] A page on the Web is considered to have decayed if it is no longer well-maintained—that is, it has been abandoned.

be computed for Web pages. This score results in a value that is approximately equal to the probability of arriving at dead pages by performing a random walk along the Web graph, starting from the Web page whose decay score is being computed. As this score accounts for dead neighbourhoods, the authors conclude that it is a better measure for page obsolescence than simply the proportion of dead links. In addition, they point out that the decay measure can also be used to guide Web crawlers, as it is probably not worth it to frequently re-crawl collections of pages that have sufficiently decayed.

Status Pant and Srinivasan consider the matter of predicting Web page status using page location and topic specificity [34], where page status is proxied by the number of in-links of a page. Using a regression model, the authors make the observation that page status can indeed be predicted using only local page features—at least in part. Interestingly, this means that the global target variable (number of in-links) can be predicted solely by using features that are local to the page, and hence are within the control of the page's owner.

Relevance rankings In their work, Elsas and Dumais explore the interaction between the dynamics of Web pages and relevance rankings [17]. Using their "novel probabilistic retrieval model", the authors consider Web pages as dynamic entities that may change over time, as opposed to static documents. They focus on navigational searches, where there is very little variation across users on the clicked results, and there tend to be a small number of highly relevant pages that are consistently relevant over time. They find that there is a strong relationship between the amount and frequency of content change on the one hand, and the relevance of a page on the other: the more relevant a page, the more likely it is to change. Related to this, Tan and Mitra look into which features of Web pages can predict their importance [43]. Here, "importance" is loosely defined and can represent different types of relevance of a page—such as PageRank.

Relation with information longevity Also touching upon the relevance of pages, Olston and Pandey look into information longevity, and draw the conclusion that—even though it is not a predictor for page change—information longevity can be an important factor for determining the usefulness (read: relevance) of page changes [33]. Santos et al. look into a different kind of relevance, where they consider the relevance of a page with regard to a specific topic [40]. They find that most pages tend to be stable regarding its relevance to a topic, but that there are some pages for which the relevance varies over time. Furthermore, they discover that the stability differs for the two topics they investigated: pages about ebola are more stable in relevance than pages about movies.

2.5 Data

The majority of related research makes use of general datasets, that attempt to capture the dynamics of the Web as a whole [3, 12, 13, 17, 18, 24, 28, 32, 33, 34, 37, 42, 43].

There are a number of papers, however, which use more specialised datasets, and try to capture the dynamics of a specific subset of the Web [3, 4, 9, 19, 20, 39, 40, 41].

Specific data Grimes and O'Brien restrict themselves to pages that change more often than once every two days [19]. Adar et al. only consider pages from the observed Web [1], that is, pages that are actually being visited by users. Barbosa et al. and Santos et al. use only pages from the Brazilian Web [4, 41]. Calzarossa and Tessera focus on crawls of three major international news websites based in the United States and the United Kingdom [9], while Gupta et al. use 87 news sources, most of them originating from India [20]. Next, there is Saad and Gançarski, who use crawls of French radio and television channel websites [39] and Santos et al., who only use pages belonging to specific topics (ebola and movies) [40]. Finally, Bar-Yossef et al. use both general and more specific data in their research: they investigate Web decay also for the papers published at ten WWW conferences, and for approximately 3800 FAQs from the Internet FAQ Archives [3].

Synthetic data There are also a few papers that make use of synthetic data [2, 42]. In particular, Avrachenkov et al. use synthetic change data, generated according to homogeneous Poisson processes[5] [2], and Singh uses synthetic datasets collected from the UCLA WebArchive project data[6] in addition to using more general data [42].

Going Dutch In our work, we also make use of a more specific dataset, namely the Dutch Web. Within this subset of the Web, we have no specific selection criteria for the qualifying pages. However, since we retrieve our dataset by extracting the *.nl* links from the Dutch Wikipedia, the pages in our dataset do have a tendency to be generally "high quality" (being informative or having a good reputation), as Wikipedia is curated by moderators. This has the effect that our dataset contains less "spam" pages than datasets in previous works. Spam pages are often automatically generated and thereby change on every page visit, and as such they skew the results (as is encountered in the work of Fetterly et al. [18]).

Cookie walls Another aspect of our dataset that sets it apart from previous works is that we develop a custom-built Web crawler (see Section 3) that circumvents cookie walls. This results in a novel high-quality dataset, which includes the data of pages hidden behind cookie walls.

[5]The data generated according to the Poisson processes consist of random time instances at which the pages change.

[6]The UCLA WebArchive project datasets can be downloaded from `http://webarchive.cs.ucla.edu`.

Chapter 3

Data Collection

We attempt to provide an answer to our research questions by analysing data. As we aim to study the change of Web pages in the Dutch Web, we need to not only have a data collection of Dutch Web pages, but also multiple versions of these Web pages over time.

Related studies In related research, there have been others who focused on studying Web change dynamics of specific subsections of the Web—such as the Brazilian Web [4, 41], Indian news websites [20], or French radio and television channel websites [39]—but no study so far has been done at investigating the Dutch Web change dynamics.

Crawler As such, we collect a data set ourselves, by creating a Web crawler that periodically crawls portions of the Dutch Web. Put simply, the crawler works as follows: every time it is invoked, it takes as input a list of URLs, which are then downloaded and stored to the local file system.

URL list The crawler needs to have as input a list of URLs that need to be crawled at every invocation of the crawler program. This URL list is compiled by taking a complete dump of the Dutch Wikipedia[1], and extracting all URLs ending in the *.nl* domain from it. In total, $2.549.732$ URLs are extracted from it, out of which 506.409 originate from the *.nl* domain.

Multithreading A characteristic of visiting (and downloading) pages on the Web is that most time is spent waiting for a reply of the Web server. This means that if we would have the naive implementation of a Web crawler that simply downloads all Web pages on the URL list consecutively, this would take a very long time, as most time is spent waiting for replies of the Web server. Therefore, we employ a multithreaded implementation, where there is a single producer-thread which adds URLs to a queue

[1] We use a snapshot of the Dutch Wikipedia at April 1st, 2021, which can be downloaded at `https://dumps.wikimedia.org/nlwiki/20210401`.

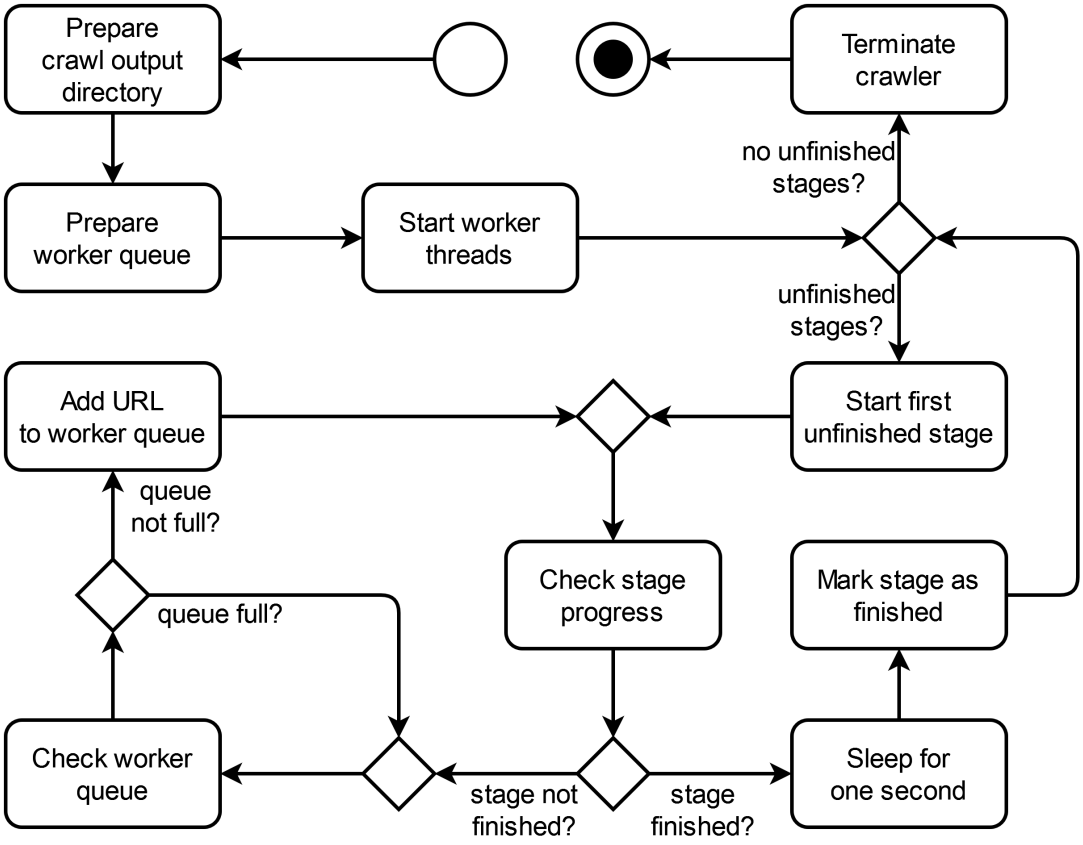

Figure 3.1: Activity diagram modelling behaviour of producer thread

(as long as the queue is not full, see Figure 3.1), and a number of consumer-threads (workers) which repeatedly take a single URL from the queue and download the corresponding Web page (see Figure 3.2).

Respecting Web servers Some second-level domains are represented frequently in the URL list. Crawling all of these Web pages may not only skew our results, but may also overload the Web servers on which the pages reside. This can in turn have two undesirable effects. First, the Web pages may become unreachable to normal users—the main target group of the server. Second, when the Web server detects frequent visits in a short period of time from the same IP address, it might suspect a denial-of-service (DoS) attack and resort to blocking the address. To prevent overloading the Web servers, we take two measures. First, we limit the maximum number of Web pages crawled from the same second-level domain to a certain value—in our case: fifty. This results in a final crawl list with 153.634 URLs across 50.016 different domains. The measure has the added advantage that our results are less likely to be skewed in favour of popular domains (because it prevents popular domains from being over-represented). Second, we enforce that there is a minimum waiting time between two visits to the same domain. We achieve this by dividing the URLs into a number of stages (again—in our case: fifty), where each stage contains at most one instance of a given second-level domain (see Figure 3.3). The stages are then crawled consecutively, with an added waiting time in between the stages.

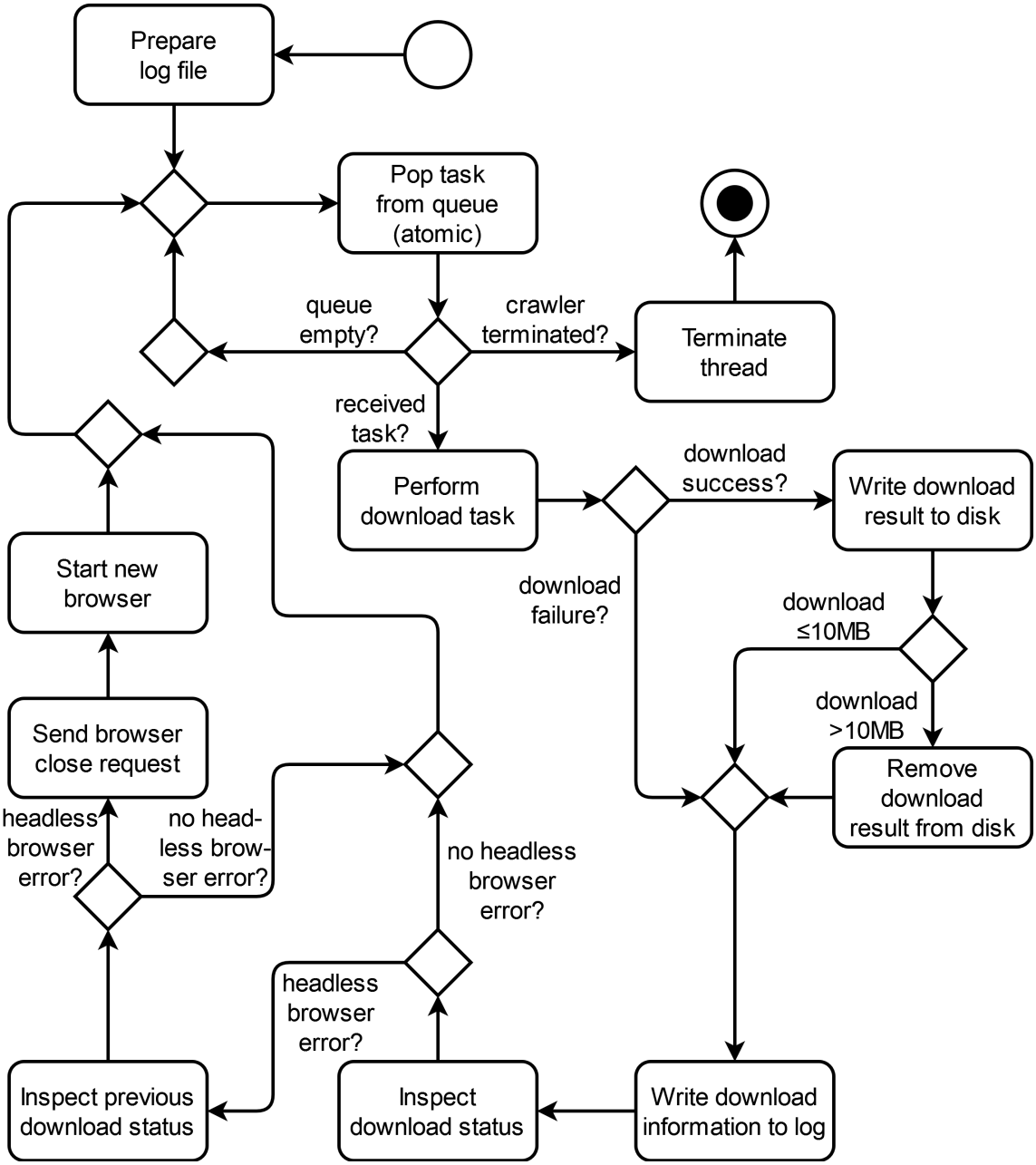

Figure 3.2: Activity diagram modelling behaviour of worker threads

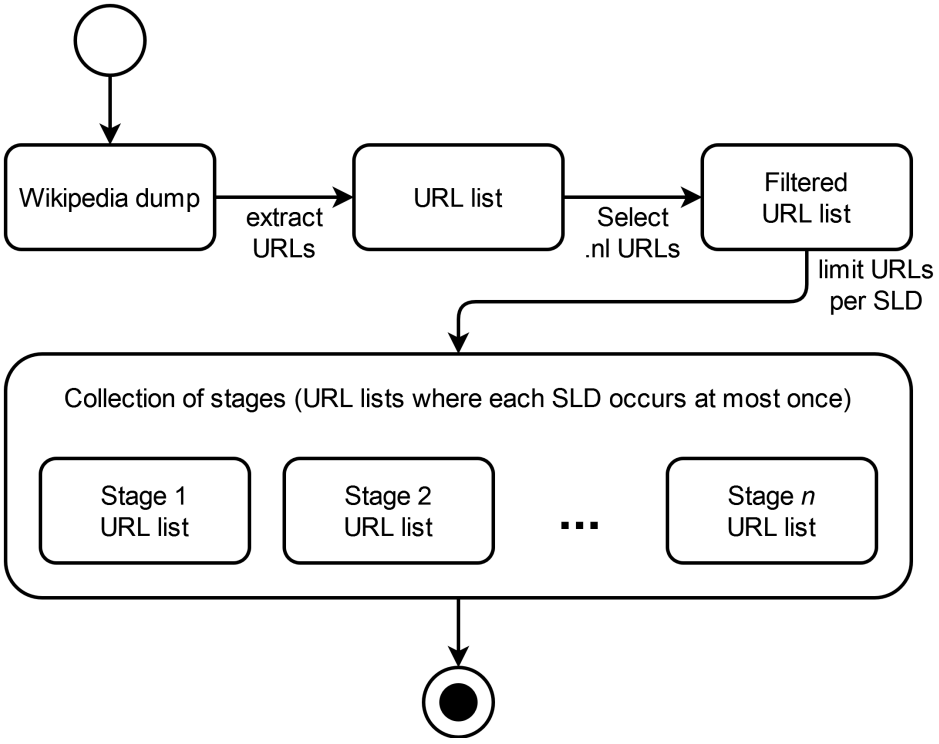

Figure 3.3: Activity diagram modelling conversion of Wikipedia dump to crawlable URL lists

Despite taking the above measures to respect Web servers, we note that the big majority of domains are only represented once or a few times in the crawl list. In Figure 3.4, we show for each of the stages the relative number of URLs in that stage compared to the first stage (which has 50.016 URLs). Already for the second stage, the relative stage size is only 32.76%, meaning that for 67.24% of all the domains in our crawl list only one page is crawled. And at stage eleven, the relative stage size is 5.04%, showing that 94.96% of the domains are represented ten times or less in the crawl lists. Finally, the relative stage size equals 1.39% at stage fifty, which means that there are only a few domains (less than 1.39%) for which we actually limited the number of crawled pages.

Cookie walls Crawling Web pages is not what it used to be. Nowadays, a lot of Web pages (especially the more popular ones) have so-called "cookie walls", on which cookies need to be accepted (or in some cases: declined) before access to the website is provided[2]. This makes crawling Web pages a lot less straightforward. Using a simple approach, Web crawling can be done by simply sending a GET request to the Web server, and storing the resulting response to disk. However, in the age of cookie walls, these responses often only contain the cookie wall, and not the actual page's content. Thus, using this simple approach would significantly decrease the quality and usefulness of our crawls. Therefore, we devise another solution. Instead of simply sending GET requests and storing the raw response results, we use headless instances of a real Web

[2]"Cookie walls" should not be confused with "cookie banners": the former completely blocks access to a Web page until a cookie decision is made, while the latter only adds an element (a banner) to the Web page, without blocking access to the page's main content.

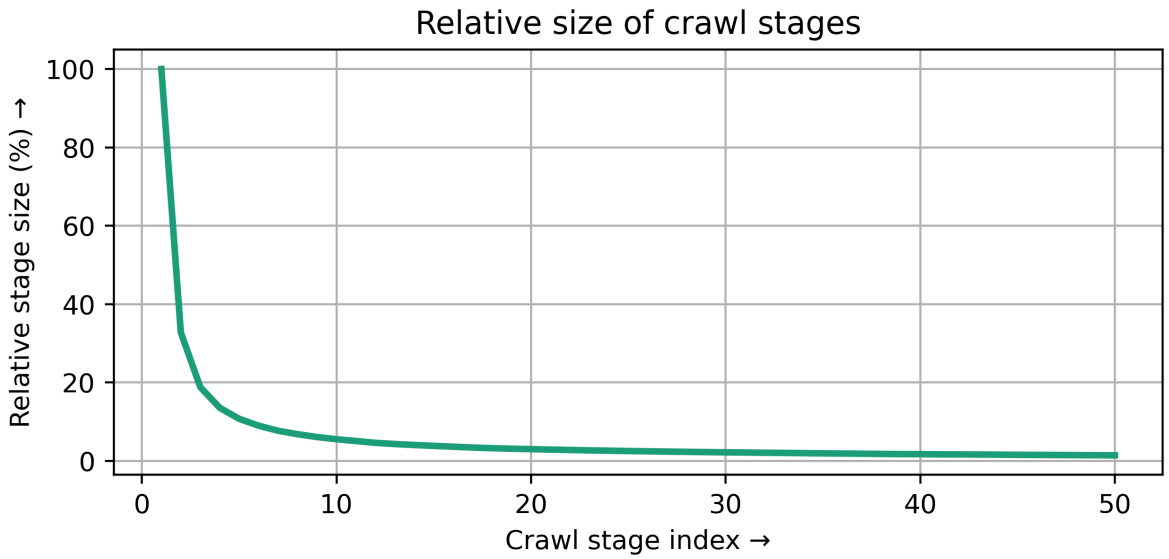

Figure 3.4: Relative size of crawl stages

browser—in our case, Mozilla Firefox[3]—and visit the Web pages using these browser instances (see Figure 3.5). To bypass the cookie wall, we use the "I don't care about cookies" browser extension[4], which automatically accepts cookies in most cases.

Repeated invocation The crawler runs from a worker node in the Hadoop cluster of the University of Twente[5]. Since we need to have multiple versions of Web pages, we schedule a cron job to invoke the crawler every day at midnight[6] (see Figure 3.6). Every job of crawling the 153.634 pages takes between twelve and fourteen hours to complete, depending on the load of the worker node and the congestion on the network.

Archiver module For every requested Web page, the crawler stores the resulting page on the local file system of the worker node. In total, every completed crawl job takes between twelve and thirteen gigabytes of local disk space. However, as the capacity of the local file system is quite limited, we employ an archiver module which moves all completed crawls to the Hadoop Distributed File System (HDFS) of the University of Twente[7] (see Figure 3.7), and we schedule a cron job to invoke it every day at 6 PM (see

[3]We use Mozilla Firefox 64-bit version 88.0.1 for Linux, which can be downloaded at `https://ftp.mozilla.org/pub/firefox/releases/88.0.1/linux-x86_64/en-US/`.

[4]We use version 3.3.0 of the "I don't care about cookies" Firefox add-on, which can be downloaded at `https://addons.mozilla.org/firefox/downloads/file/3761156/i_dont_care_about_cookies-3.3.0-an+fx.xpi`.

[5]The worker node has a dual-socket AMD Opteron 4386 setup, with 64 GB of RAM and approximately 100 GB of usable disk space. Note that we do not have exclusive access to the node, and some of the resources may be allocated to Hadoop jobs for some of the time.

[6]Midnight is defined to be 12:00 AM according to the system clock on the worker node, which is set equal to the Central European Summer Time timezone.

[7]The HDFS has a total capacity of 520 terabytes, hence with our numbers we do not need to worry about disk space here.

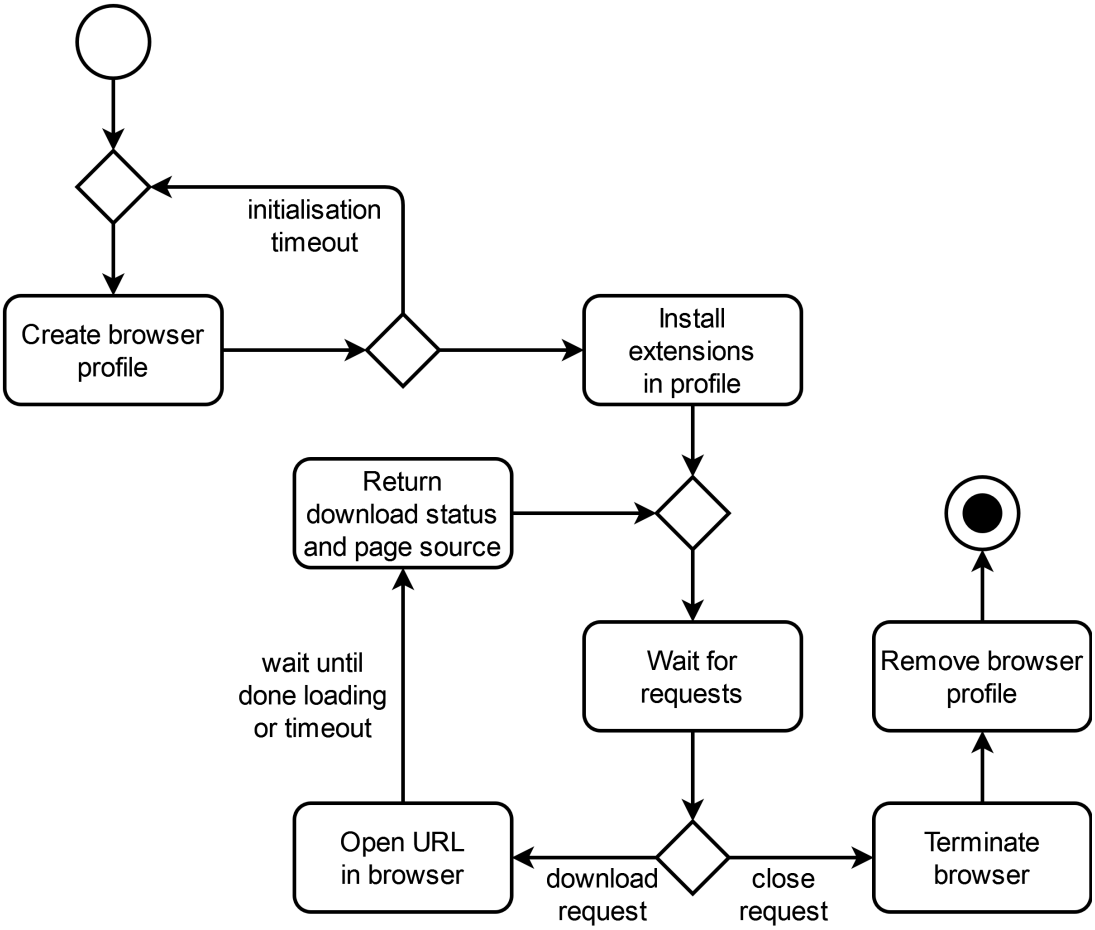

Figure 3.5: Activity diagram modelling headless browser life cycle

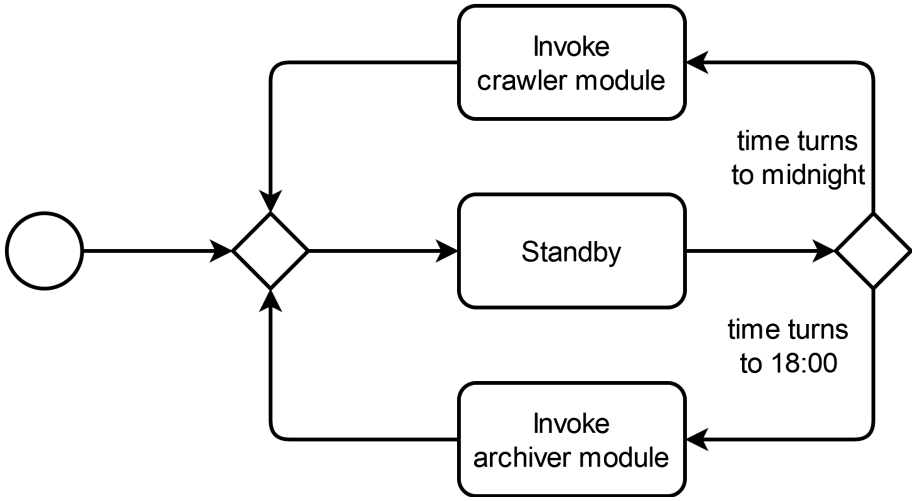

Figure 3.6: Activity diagram modelling repeated invocation of crawler and archiver modules

Figure 3.6). Note that on the HDFS, three replicas of every file are stored for reliability, hence in practice every day worth of crawls takes up 36-39 GB of HDFS storage space.

Malicious websites As the crawler is run from a worker node at the University of Twente, it is subject to the institution's security guidelines. In the early days of running the crawler, some of the traffic got flagged by Quarantainenet (an organisation monitoring traffic for suspicious activity on behalf of the University of Twente), threatening to place the worker node in quarantine. This would effectively mean that internet access would be blocked. The cause that some of the traffic was flagged, was that some of the crawled Web pages contain content (such as advertisements or Iframes) originating from malicious Web servers. To prevent our crawler from losing internet access, we use the "uBlock Origin" browser extension[8], which blocks (among others) access to known malware domains.

Broken browsers During test runs of the Web crawler, we noticed that occasionally after a requesting a Web page that results in a timeout, all subsequent requests by the same browser instance (the same thread) also result in timeouts. The browser instance becomes unresponsive—it appears as if it is "broken". This phenomenon is undesirable, as it effectively causes us to not crawl pages that are actually available. A naive solution would be to initialise a new browser in case of timeouts. However, initialising a new browser instance is a resource-intensive operation, and the big majority of timeouts are "true" timeouts, with only a small fraction being "false" timeouts. Therefore, another solution is desirable. Our solution is as follows (see Figure 3.8): in the case that a timeout occurs, we send a simple *GET* request—outside of the browser instance—to the same URL. If this simple request also does not succeed in retrieving the page, we conclude that the timeout was—as expected—a true timeout. However, if the simple

[8]We use version 1.35.2 of the "uBlock Origin" Firefox add-on, which can be downloaded at `https://addons.mozilla.org/firefox/downloads/file/3768975/ublock_origin-1.35.2-an+fx.xpi`.

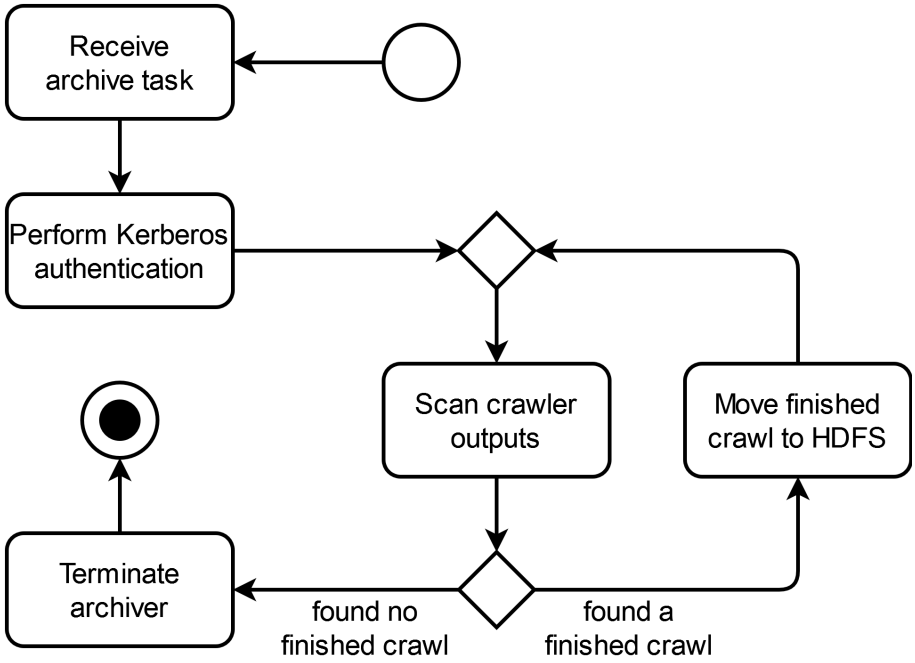

Figure 3.7: Activity diagram modelling behaviour of archiver module

request manages to successfully retrieve the page, then we initialise a new browser instance, and use it to do another attempt to fetch the page. If it now succeeds to get a response from the Web server, we store the retrieved response. However, if the request still results in a timeout, we initialise yet another new browser instance, as perhaps the previous new instance immediately became broken. We do another attempt to fetch the page using the new browser, and store the retrieved page if it does not result in a timeout. If it still does, however, we are in a strange situation: we can retrieve the page using a simple *GET* request, but cannot do so in a Web browser. Most likely, this is caused by the Web server in question being poorly reachable. In this case, we fall back to storing the response we got from the simple *GET* request[9].

Big files At the crawling stage, we do not yet look at the contents of the crawled files. Still, in the scope of this project we limit ourselves to studying the change of Web pages, not those of other files. In test runs of our crawler, the crawled archives quickly became quite large, caused mainly by a small number of very big files (such as videos or big *.pdf*s). In order to limit the size of our crawls without decreasing data quality, we only store crawled files with a size of less than or equal to 10 MB. This way, storing big files is prevented, while all Web pages are still stored to disk[10].

Logging Every worker thread spawned by the crawler program keeps track of a log file. In this file, it keeps track of the timestamp at which the crawl of an URL was initiated, the status of the crawl attempt (success, timeout, or other error), the stage

[9]The reason that we do not immediately use the response from the simple *GET* request is that it results in a lower data quality: cookie walls are not bypassed.

[10]None of the >10 MB files we encountered were Web pages.

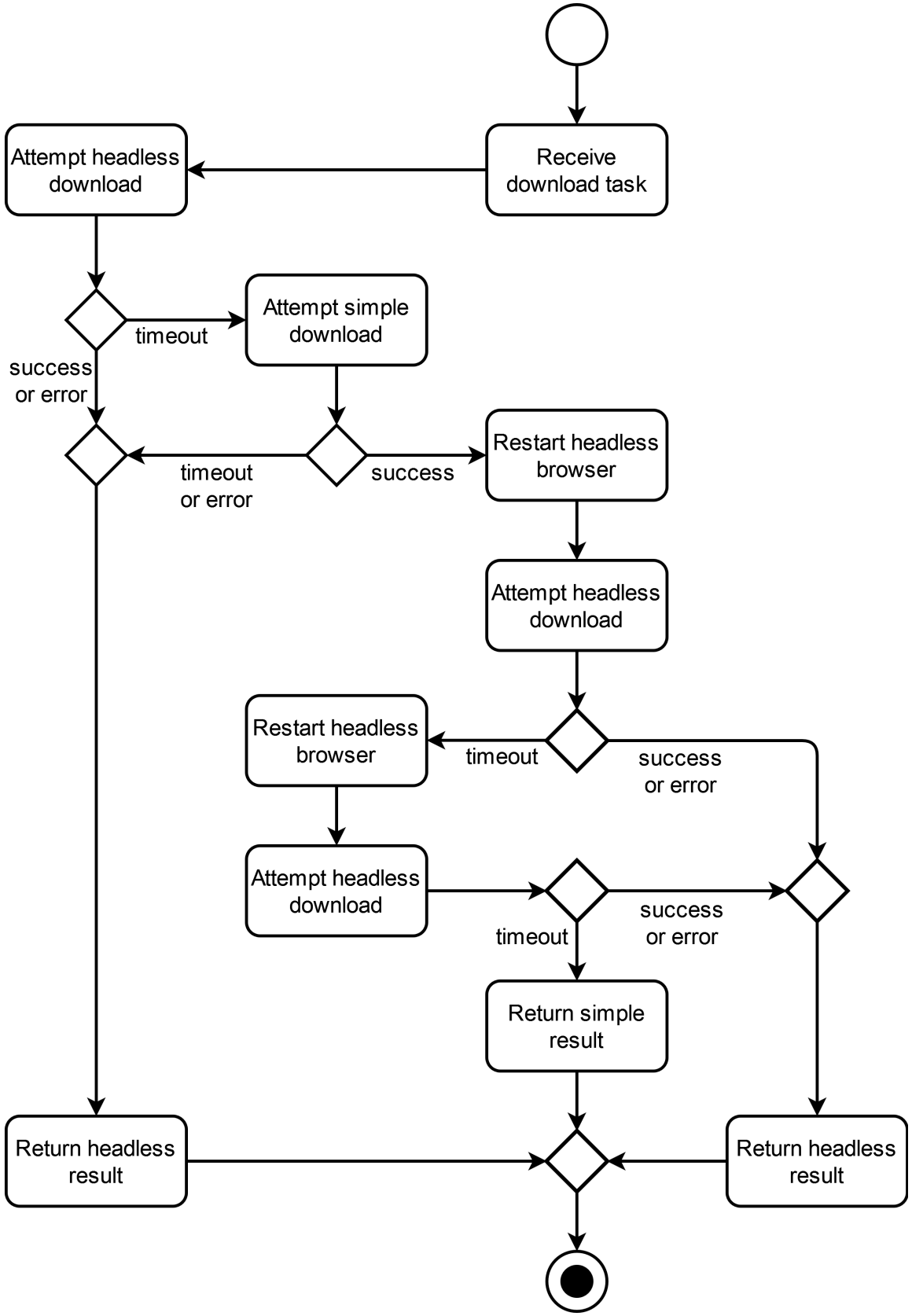

Figure 3.8: Activity diagram modelling headless browser page download procedure

filename and line number of the URL in question, whether a response is stored on disk, and whether the response was received using a headless browser instance or a simple *GET* request. Using these logs, we are able to quickly locate files, gain insights about the crawling process, and link crawl results to crawling threads (which can be useful if it later turns out that something went wrong with a specific thread).

Unresponsive browsers After the crawler program had been running for some time, we discovered a phenomenon in the same vein as the false timeouts. In some events, it appears that a browser instance becomes (temporarily) broken while not reporting any errors or timeouts. At that point, the Web browser becomes unresponsive to our requests, and does not navigate to another URL if we request it to. However, we are still able to save the page, meaning that in those cases we save a duplicate[11] of a previously requested page for a new page. This issue was only discovered later in the real crawling process, meaning that this error persists in our output data. As such, before actually using the data, we filter out false duplicate pages by removing all consecutively crawled pages by the same thread having the exact same page text.

Crawl duration In total, the URL list of 153.634 URLs was crawled for ninety consecutive days (from Saturday, June 12th, 2021 up until and including Thursday, September 9th, 2021), resulting in approximately 1.1 TB of raw data. However, on four of the ninety days (days 65, 68, 71, and 81), the crawler program crashed before finishing all the crawls, resulting in incomplete crawls on those days. As such, when analysing the crawl data, care needs to be taken to account for these incomplete crawls.

Compression Initially, all the crawled pages were stored as individual uncompressed files on the Hadoop Distributed File System of the University of Twente. However, this caused crawl processing to be very slow (since file input/output can be quite slow, especially on distributed systems), and it caused disk space to be "wasted". As such, after the ninety-day crawl period ended, we converted the data of each of the crawl days to a small collection of Parquet files, reducing the overall disk usage footprint to roughly 204 GB (a 81.5% reduction!).

Feature extraction From the crawled Web pages, we extract a number of "features". In the scope of this project, we limit ourselves to studying the change of *HTML* Web pages (which is the big majority in our data set), and not the change of other file types (such as *.pdf*s or videos). As such, we can extract features that are specific to HTML pages, increasing the expressiveness of the extracted features. In total, we extract nine across two categories:

[11] Actually, we save an almost-duplicate, as the page may have changed in the meantime, for example using JavaScript code or scheduled refresh intervals.

1. **Linkage features:**
   (a) Internal out-links (hyperlinks to pages within the same second-level domain)
   (b) External out-links (hyperlinks to pages outside the second-level domain)
   (c) E-mail addresses
2. **HTML features:**
   (a) Page text
   (b) Images
   (c) Scripts
   (d) Tables
   (e) HTML tags
   (f) Meta properties

We only consider local features (i.e., features that can be computed individually for a single page, without requiring data of other pages), and do not consider network features. This is because with our crawling method, we crawl a collection of separate Web pages that are not necessarily connected to each other with links, and as such we are unable to compute the number of in-links. Nonetheless, not using network features makes predictive models more easily applicable in practice, because it allows for making predictions for a single page without crawling a big portion of the Web.

Feature distributions In Table 3.1, we show the distributions of numerical feature values. This table is generated as follows. First, for all pages the features are transformed into numerical values by counting the number of occurrences (such as the *number of images* on a page). For each of the features, these values are put in a list. These lists are then sorted, and the numerical feature values at eleven intervals ($[0\%, 10\%, ..., 100\%]$) are taken. Using these distributions, we can derive properties of our dataset. For example, by inspecting the 50% column, we determine that the median page in our dataset has six images, 120 lines of page text, and fourteen scripts.

Table 3.1: Distributions of numerical page features

Feature \ Percentile	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
E-mail addresses	0	0	0	0	0	0	0	0	1	2	647
External out-links	0	0	0	2	3	5	7	10	16	28	6.961
Images	0	0	1	2	4	6	9	14	21	36	7.651
Internal out-links	0	1	9	18	29	41	55	74	100	161	15.179
Meta properties	0	2	3	5	7	9	13	16	20	25	3.369
Lines of page text	2	12	47	74	100	120	152	193	252	375	84.632
Scripts	0	0	3	7	10	14	17	22	29	41	515
Tables	0	0	0	0	0	0	0	0	0	2	2.508
HTML Tags	3	47	155	233	298	369	457	568	735	1.060	85.975

Chapter 4

Change Types

In this chapter, we investigate our first research question:

RQ1 What are the type of changes that pages on the Dutch Web undergo?

- **RQ1.1** What is the fraction of pages on the Dutch Web that undergoes any change?
- **RQ1.2** For the pages on the Dutch Web that undergo any change, what is the distribution over change types? Possible change types are changes in page text, scripts, internal out-links, external out-links, email addresses, images, tables, meta elements, and HTML tags.
- **RQ1.3** Out of the pages on the Dutch Web that change, what fraction contains potentially informative changes?
- **RQ1.4** Whenever the readable text on a page on the Dutch Web changes, in what way does it change?
- **RQ1.5** Whenever a feature on a page on the Dutch Web changes, by how much does it change? Possible features are page text, scripts, internal out-links, external out-links, email addresses, images, tables, meta elements, and HTML tags.

The goal of this chapter is to get a good idea about the type of changes that pages on the Dutch Web undergo.

Single day-pair We answer this question by selecting two consecutive days out of our crawls, and analysing the difference between all pages that were successfully crawled both days. There are two reasons that we limit ourselves to a single day-pair worth of crawls. First, we make the assumption that the proportions of the change types do

not differ much between day-pairs[1], and hence analysing the difference for multiple day-pairs does not have a lot added value. Second, due to the big number of pages, analysing the crawls takes a lot of computation time, which given the expected insights gained by analysing additional day-pairs is better spent elsewhere.

Any change In our case, we select the first two days of our crawls (Saturday, June 12, 2021 and Sunday, June 13, 2021) to do the investigation on. Now the most basic thing to do is to look at the fraction of pages that changes between the two days, where we define a page to be changed if the MD5 page hash computed on page source is different on the second day than it is on the first day. We measured this fraction to be 0.687. In other words, for over two thirds of the pages on the Dutch Web, a page shows some kind of change within 24 hours. To explain what these changes entail, we dive deeper into the changes.

4.1 Existence of Change

Change fractions In Figure 4.1, we plot for different page features the fraction of pages for which that feature changes in a day-pair. As can be seen, the feature of pages that changes most often are the scripts, with a change fraction of approximately 0.4. This feature is followed by text, then HTML tags, after that images, and not much later internal out-links, all of which have change fractions between 0.27 and 0.19. Finally, with change fractions under 0.08, features that change least often are external out-links, meta elements, tables, and email addresses.

Informative changes Not all features result in equally useful information, neither for human nor machine. For example, the change of page scripts is usually not a very interesting event, as more often than not the change is caused by an external dependency update, and not immediately visible on the page. In contrast, other feature changes are inherently more interesting events, such as the change of text—which is immediately visible on the page, or the change of out-links—from which we can discover new websites (external out-links) or discover the updated structure of the current website (internal out-links). We make a division of features into those that are presumably informative (page text, internal out-links, external out-links, email addresses, images, and tables) and those that are presumably not (scripts, HTML tags, meta elements, and other changes). After that, we determine for each of the changed pages whether the

[1]As we later verified, this assumption is not completely true. If we compare the first day-pair (corresponding to the changes discovered on Sunday, June 13th, 2021) to the third day-pair (corresponding to the changes discovered on Tuesday, June 15th, 2021), the relative change frequencies per feature of the latter day compared to the first day are the following: email addresses: 125.2%, internal out-links: 118.8%, external out-links: 117.8%, meta elements: 117.5%, text: 116.3%, HTML tags: 111.3%, images: 110.0%, scripts: 105.8%, tables: 99.5%. But despite this false assumption, the findings in this chapter are still meaningful, with the small side note that the results are slightly different on different weekdays.

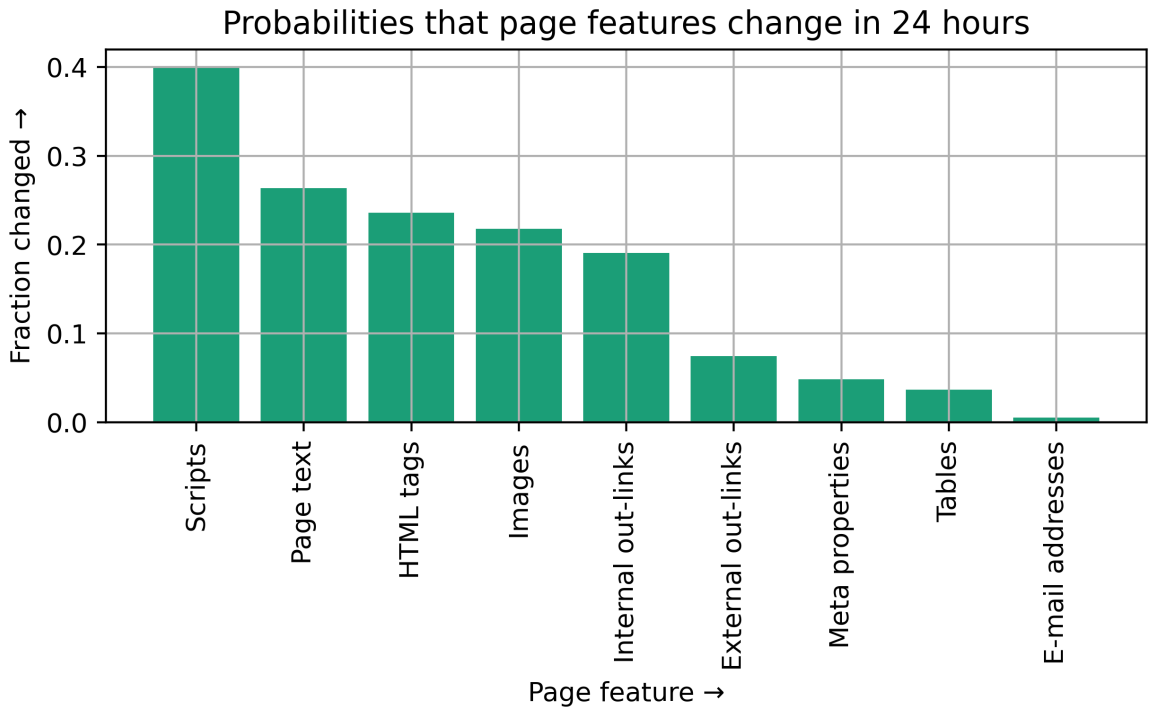

Figure 4.1: Probability that page features change in 24 hours

changes they undergo are presumably informative (meaning that presumably informative features have changed), not informative (meaning that presumably uninformative features have changed), or both[2]. The results are plotted in Figure 4.2. As can be seen in the pie chart, just over half of the page changes bring informative changes, with the remainder only bringing uninformative changes. This high fraction of presumably uninformative changes matches the findings of Fetterly et al., who discover that when Web pages change, they often only change in their mark-up or other trivial ways [18].

Text changes We are also interested in what way readable text on Web pages changes. To investigate this, we took a random sample of day-pairs for which the text changed[3]. Next, we manually inspected the differences in text between the two pages in a day-pair, and assigned at least one change category to each pair. From this, we plotted the encountered distribution over text change categories in Figure 4.3.

In total, we encountered 23 change categories:

1. **External element** (43): text generated by external page elements not contributing to main page content, such as a music player, a social media share button, or a cookie banner.

[2]Although it appears to be contradictory, a page can undergo both informative and uninformative changes at the same time. This happens whenever both an informative feature and an uninformative feature change in a single day-pair.

[3]We took this random sample by selecting day-pairs with probability 0.0092 (given that the text changes in the day-pair), out of the first five crawled pages from every second-level domain. This resulted in a sample of 155 pages out of the 16847 day-pairs satisfying the selection criteria.

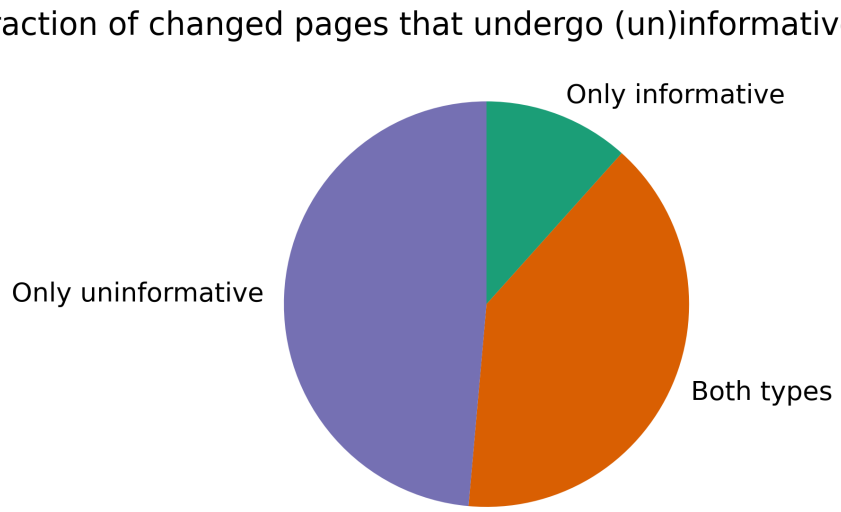

Figure 4.2: Fraction of changed pages for which page changes are likely informative

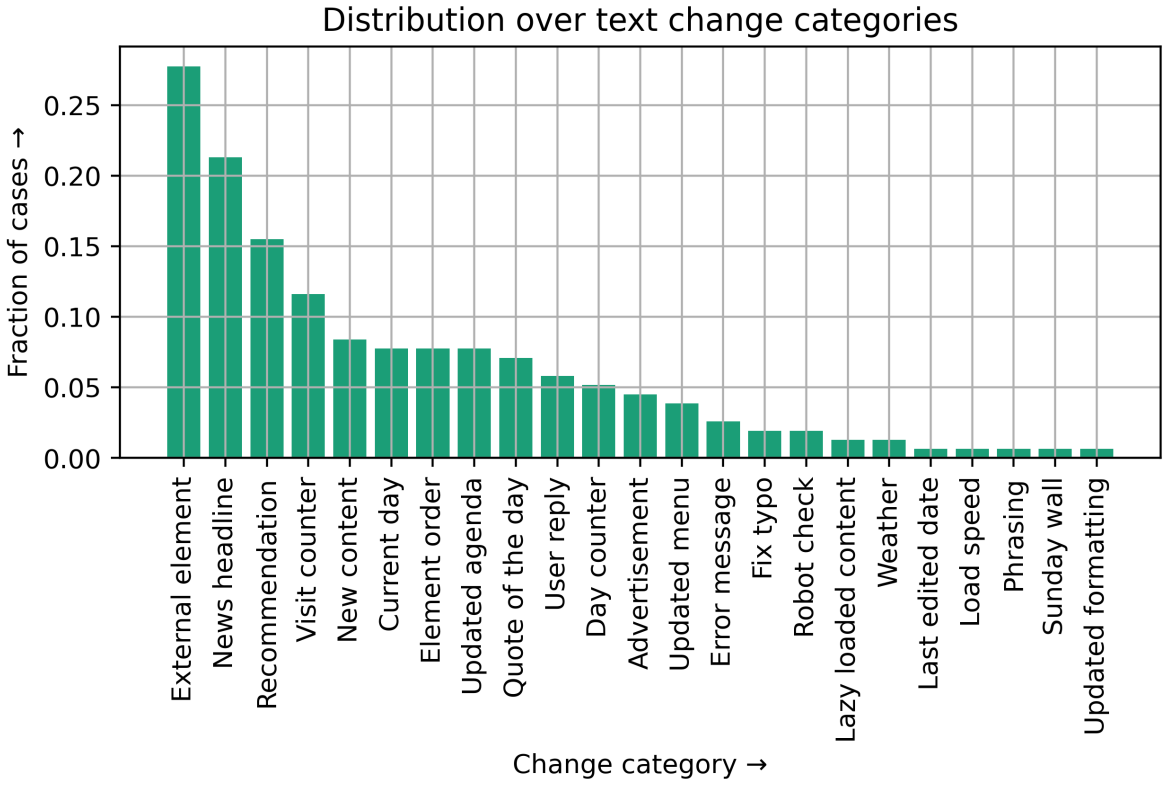

Figure 4.3: Distribution over text change categories

2. **News headline** (33): new news headlines or excerpts, new links to articles, or new links to blog articles.
3. **Recommendation** (24): recommendations to the user on pages to visit next generated by some kind of algorithm, such as suggested articles, popular articles, or popular tags.
4. **Visit counter** (18): number counting the visits to a page or article.
5. **New content** (13): new information in the main page content, such as an added paragraph with new information.
6. **Current day** (12): text displaying the current weekday or date.
7. **Element order** (12): updated ordering of the page text, such as a paragraph being moved to the top of the page.
8. **Updated agenda** (12): updated calendar, such as new calendar entries, or updated event details (for example time or location).
9. **Quote of the day** (11): a daily featured element, such as a quote of the day, a featured image, or a daily highlighted article.
10. **User reply** (9): changes in user comments or user reviews, such as a new user comment, or an updated rating.
11. **Day counter** (8): number counting the days since some event, such as the number of days since the release of an article.
12. **Advertisement** (7): text from an advertisement or from a sponsor.
13. **Updated menu** (6): changes in the site menu, such as a new menu item, or an interchanged order of menu items.
14. **Error message** (4): text about an error message, such as an internal server error (HTTP status code 500).
15. **Fix typo** (3): correction of a typo, adjustment of white-space, or replacement of a wrongly used word.
16. **Robot check** (3): text from a robot-check, such as a simple math problem.
17. **Lazy loaded content** (2): differing degrees of lazy-loaded page content, such as a different amount of loaded articles for infinite-scrolling pages.
18. **Weather** (2): text displaying the current weather conditions or temperature.
19. **Last edited date** (1): updated last-edited date text.
20. **Load speed** (1): number displaying the milliseconds it took for a page to load.
21. **Phrasing** (1): difference in the phrasing of main page content, but with no new information.

22. **Sunday wall** (1): a wall of text replacing the page content, preventing it to be read on Sundays.
23. **Updated formatting** (1): change in page formatting, such as moving a heading to a different level (e.g. turning a page title into a paragraph title).

Useful intentional text changes Not all change categories are equally interesting to users. The categories that are particularly useful to discover new information and that are purposefully added by site moderators as opposed to an algorithm are "news headline", "new content", "updated agenda", and "user reply". In 36.1% of the sample, at least one of these four categories is present.

Miscellaneous changes Unfortunately, the features that we use do not cover all the possible page changes. There exists a fraction of pages for which none of the described features change, but where there still is some kind of change compared to the day before. This fraction is equal to 0.125. By inspecting a sample of such pages, we found that these miscellaneous changes are mostly caused by one of two things: either by (authentication) tokens, or by randomly generated element IDs (for programmatically generated HTML elements, such as ranking lists) that are different on every page visit.

Summary Within a given 24-hour period, over two-thirds of the pages on the Dutch Web undergo some kind of change. Page scripts change most often (40% of the time), followed by text (26%), HTML tags (24%), images (22%), and internal out-links (19%). Email addresses change least often ($< 1\%$), followed by tables (4%), meta elements (5%), and external out-links (7%).

Just over half of the page changes are presumably informative (meaning that they add new content or links), the remainder are presumably not. This high portion of presumably uninformative changes matches the findings of Fetterly et al., who discover that when Web pages change, they often only change in their mark-up or other trivial ways [18]. And even when the text (a presumably informative feature) of a page changes, the changes are often not very useful. Only in about 36% of the cases, text changes reflect new content being purposefully added by a human being (as opposed to automatically generated content, such as visitor counters and advertisements).

4.2 Amount of Change

Change amplitudes In Figure 4.1, we already showed the probabilities that page features change for a given day-pair. What we did not show yet is *how much* they change. To investigate this, we look at change amplitudes. The change amplitude of a given feature consists of two values: a positive value and a negative value, representing respectively the fractions of the feature that are added and deleted in between the two page crawls. This is similar to the approach by Meegahapola et al., who use the number

of new and removed feature elements as features for training a Random Forest Machine Learning model [28].

Amplitude definition In more detail, when extracting a feature from a given page, we put all page elements satisfying the requirement for that given feature in a sequence. For example, for the external out-links feature we put all the outgoing URLs to pages outside of the second-level domain in a sequence, and for text we create a sequence with as entries all the non-empty lines of plain text. We do this for all of our features, for both pages in an day-pair, resulting in two sequences per feature (one for each day). We compare these two sequences to compute three values: the number of equal elements between the two sequences (e), the number of elements that need to be inserted into the first sequence to retrieve the second sequence (i), and the number of elements that need to be deleted from the first sequence to retrieve the second sequence (d). Using these three values, we can compute the positive value (p) and the negative value (n) of the change amplitude for the feature as following:

$$p = i/(e + i + d), \tag{4.1}$$

$$n = d/(e + i + d). \tag{4.2}$$

We repeat this procedure for every feature in all day-pairs, and then average the amplitudes per feature[4] to retrieve the results as plotted in Figure 4.4.

We can make several observations from this figure. First, there is a notable difference between the change amplitudes of the different features, meaning that some features (notably email addresses and tables) relatively change much more than other features (such as scripts and page text). One possible explanation for this is that pages on average do not have a lot of email addresses and tables, and hence once such a feature changes, it proportionally automatically changes a lot.

Close absolutes Another observation we can make is that (the absolute values of) the average positive and negative values of the change amplitudes are very close to each other, meaning that on average just as much content is added to a page as is removed from it, which is true for every feature.

Percentile curve In addition to looking at average change amplitudes per feature, we also look at each of the change amplitudes for individual pages. We do this by for a given feature sorting the change amplitudes (both positive and negative) in descending order, and plotting these values against a linear space from 0 to 100. We repeat this procedure for every feature (see Figure 4.5). The graphs can be read as follows: for a given percentile (x-value) and the corresponding change amplitude (y-value), the percentile denotes the percentage of pages having that change amplitude or higher.

[4]Note that we average the positive and negative values of the change amplitudes separately.

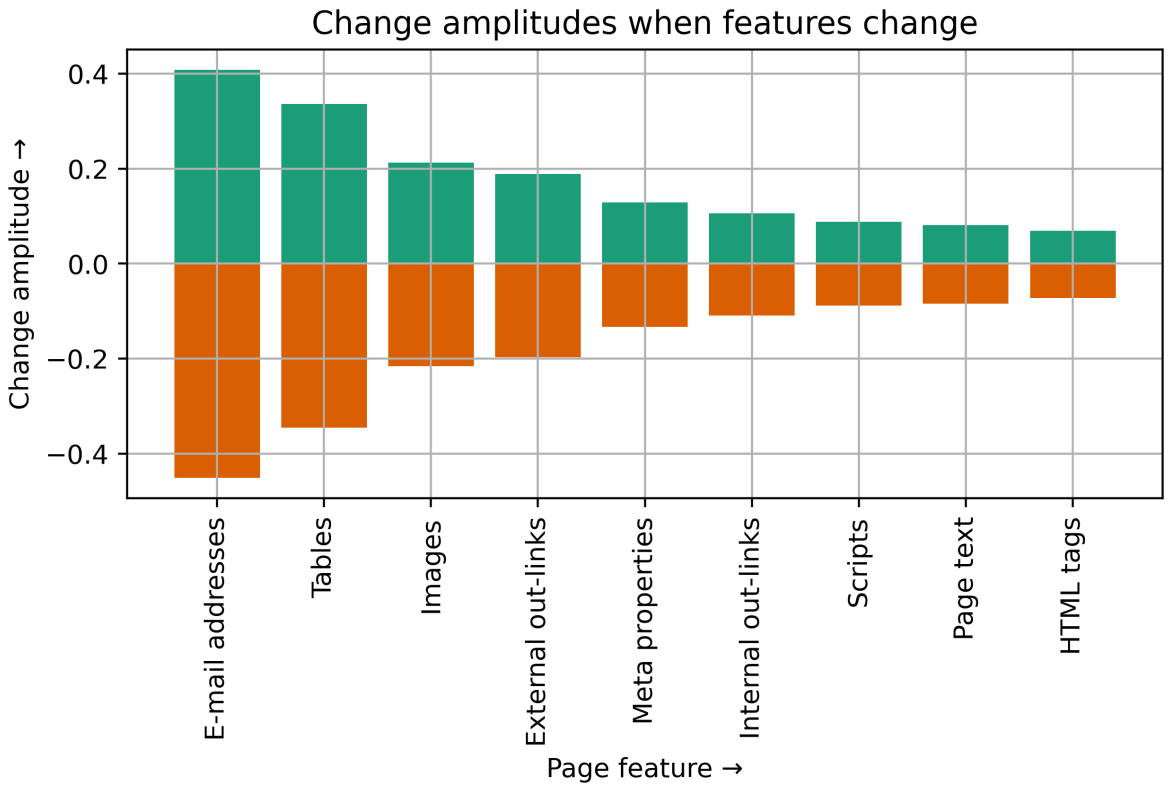

Figure 4.4: Change amplitudes when features change

Overlapping curves By inspecting the plots in Figure 4.5, we can make several observations. First, note that in every plot, both the positive and negative values of the change amplitude are displayed, although the latter curve is barely visible. This is because the two curves almost completely overlap, which is in line with the observation we made from Figure 4.4 about that on average just as much content is added to a page as is removed from it. Still, now we can make a stronger claim, as we observe that not only the average of the positive and negative values is very similar, but also the distributions of positive and negative change amplitude values. The only plot of Figure 4.5 in which the negative value curve is notably visible, is at the email addresses. The negative value curve here is slightly above the positive value curve for a portion of the plot, indicating that in those cases more email addresses are removed from a page than are added to it. This effect is also somewhat visible in Figure 4.4, where the negative bar for email addresses is slightly longer than the positive bar (whereas for the other features there is not such a notable difference between the bar lengths).

Attractive corners Second, we observe that in most plots of Figure 4.5, the curves appear to gravitate towards the bottom-left corner. This shows that for these features, there is only a small fraction of pages for which the features change considerably, and for most of the pages the magnitude of the changes is quite small. The exception to this observation are the email addresses and tables features, which matches a previous observation about Figure 4.4, where we noted that these features on average change

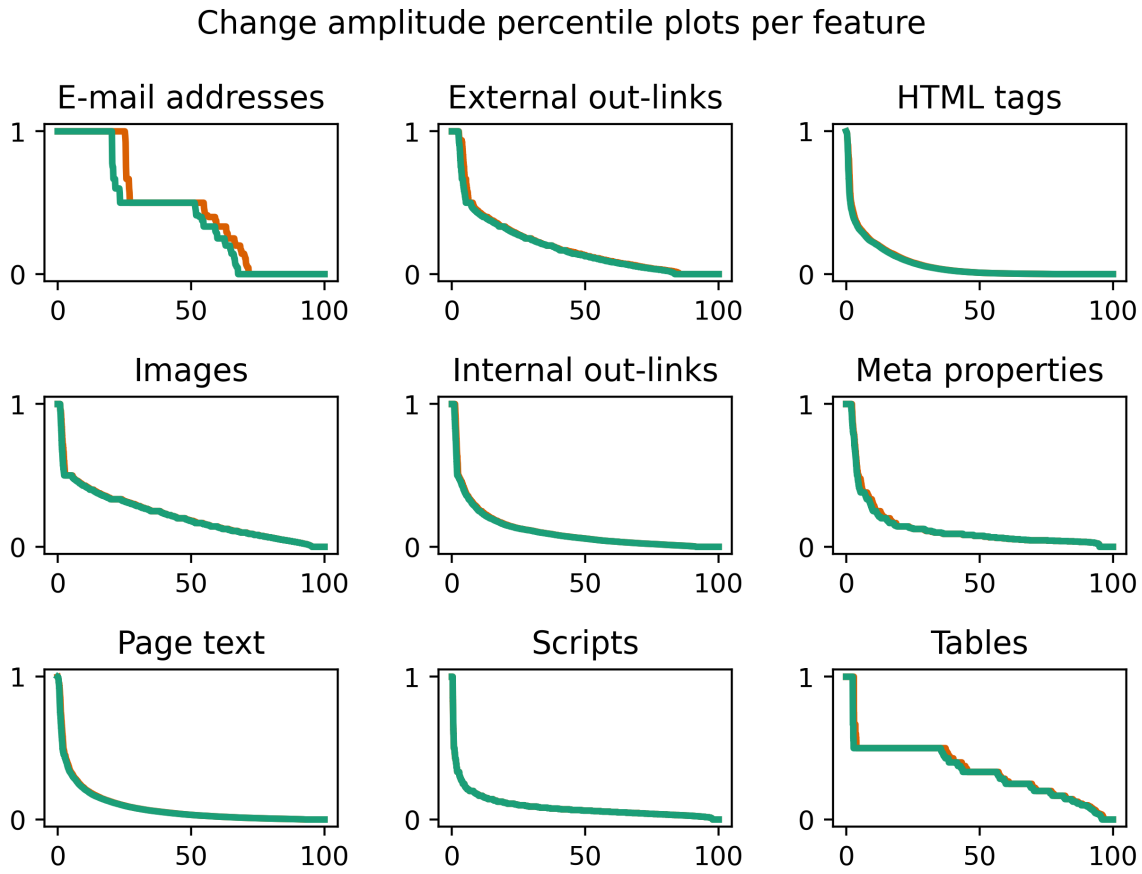

Figure 4.5: Change amplitude percentile plots per feature. Legend: x-axis: percentile, y-axis: change amplitude, green curve: positive amplitude value (Equation 4.1), orange curve: negative amplitude value (Equation 4.2). The graphs can be read as follows: for a given percentile (x-value) and the corresponding change amplitude (y-value), the percentile denotes the percentage of pages having that change amplitude or higher.

relatively much more than the other features.

Staircase behaviour Third, we notice that although the curves are smooth in most plots of Figure 4.5, this is not particularly the case for the email addresses and tables plots. In particular, in these plots we can see that there is a large fraction of pages that have specific change amplitude values such as 0.5, 1, and 0. This staircase-behaviour again indicates that pages on average do not have a lot of such elements (which can also be observed in Table 3.1 in Chapter 3, where it is shown that over 90% of the pages contain two or fewer e-mail addresses). For example, if a page has only one email address, then the addition of a second email address already causes the positive amplitude to be exactly equal to 0.5 (since one email address is added and one stays the same).

Change amplitudes of zero Related to this, although seemingly contradictory: if a certain page feature rarely changes, it is easier for that feature to achieve a change amplitude value of exactly 0. This appears contradictory, since by definition the change amplitude percentile plots only include pages for which the given feature changes. However, a change can mean both an insertion or a deletion, and a page can undergo an insertion without undergoing a deletion (or vice versa). And if a feature rarely changes, the probability that the feature has a deletion given that it also has an insertion (or vice versa) is lower than it would be if the feature were frequently changed. Combined with the fact that the positive and negative change amplitude values are sorted independently, it is easier for the curves in the percentile plot to reach zero as the percentile (x-axis) nears 100 for features that rarely change. In Figure 4.5, this phenomenon is most apparent for the email addresses plot, which matches our previous observation that this feature rarely changes.

Summary Whenever a page feature changes, how much it changes depends on the feature type. Although rare, when email addresses or tables change, they change a lot. On the other hand, when HTML tags, page text, scripts, or internal out-links change, they only tend to change in small amounts. Nevertheless, for all features, on average just as much is added to the feature as is removed from it. In fact, even the distributions of positive change amplitudes and negative change amplitudes are almost equal.

Chapter 5

Change Patterns

In this chapter, we investigate our second research question:

RQ2 What are the temporal change patterns of pages on the Dutch Web?

- **RQ2.1** What are the daily page change frequencies of pages on the Dutch Web?
- **RQ2.2** Do the page change frequencies of pages on the Dutch Web follow specific temporal patterns?
- **RQ2.3** To what degree do page change frequencies of pages on the Dutch Web differ across different days?
- **RQ2.4** To what extent do pages on the Dutch Web keep exhibiting the same change behaviour?

The goal of this chapter is to discover how pages on the Dutch Web change over time.

Text and links We answer this question by focusing on the changes of three page features: *page text*, *internal out-links*, and *external out-links*. The reason that we focus on these features is that they are the most informative: changes in page text frequently indicate new page content, and changes in page out-links allow for the discovery of new Web pages.

Binary change For every day that a page is crawled, we compute the changes compared to the day before. We achieve this by considering binary change events for a given page feature: either the feature of the page in question changed compared to the day before, or it did not change. We repeat this computation for all the pages we have crawled, for the three page features that we are interested in, for all the 90 days of crawls. If a crawl of a given page failed on a specific day, we say that the page did not change on that day (since if we cannot inspect the page, we cannot detect any changes). Also if the crawl of a given page failed on the day preceding a specific day, we assume

that the page has not changed on that day (since we have no previous day to compare the page against).

False unchanging pages Given the fact that we consider a page to not have changed if the page crawl of that day has failed, we may incorrectly conclude that a page did not change on a given day. However, we reason that this issue uniformly affects each of the crawled days (after adjusting the dataset, see below), since the probability of a random page crawl failure is small, and the majority of page crawl failures occurs for a small collection of badly reachable pages that stays the same over the days. Hence, we conclude that this assumption has an insignificant influence on the results.

5.1 Daily Changes

Adjusted dataset Before investigating the crawls, we have to make an adjustment to the crawled data. Unfortunately, due to crawler failures, the crawler program was terminated early on four out of the ninety days that pages were crawled. Still, even on these four days, the first page from every domain in our crawl list was successfully crawled, giving us enough data to work with (50.016 crawled pages per day, approximately 32% of the full dataset). Hence, in the remainder of this chapter (except when mentioned otherwise), we use as dataset all the first pages from every domain.

Repeating pattern Using the adjusted dataset, we plot the fraction of pages for which page text, internal out-links, and external out-links change per day in Figure 5.1. From this figure, we can draw two conclusions. First, we notice that all three of the target features exhibit a pattern repeating every seven days. The lower two change fractions of the pattern correspond to changes occurring during the weekend (Saturday and Sunday), while the upper five values of the pattern correspond to the weekdays (Monday, Tuesday, Wednesday, Thursday, and Friday)[1]. Hence, we conclude that these features change more often on weekdays compared to weekend days. This is in line with the observation of Calzarossa and Tessera, who remark that weekday is correlated with a page's change frequency [9]. Second, we observe that besides these weekly patterns, the change fractions remain approximately constant. There is *some* variation, but the change fractions of each of the target features stay in the same region.

Link changes In Figure 5.1, we also plotted the fractions of pages per day for which internal out-links and external out-links change. We notice here that the change fractions of these features are notably lower than the change fractions of page text, with the change fractions of external out-links being the lowest of the three. This is in line

[1] Actually, the two lower values in each cycle of the seven-day repeating pattern correspond to the change values computed on Sunday and Monday (i.e., the changes between Saturday-Sunday and Sunday-Monday). But since each of the daily crawls started at midnight, the change value computed on a given day can mostly be interpreted as the changes that occurred the day before.

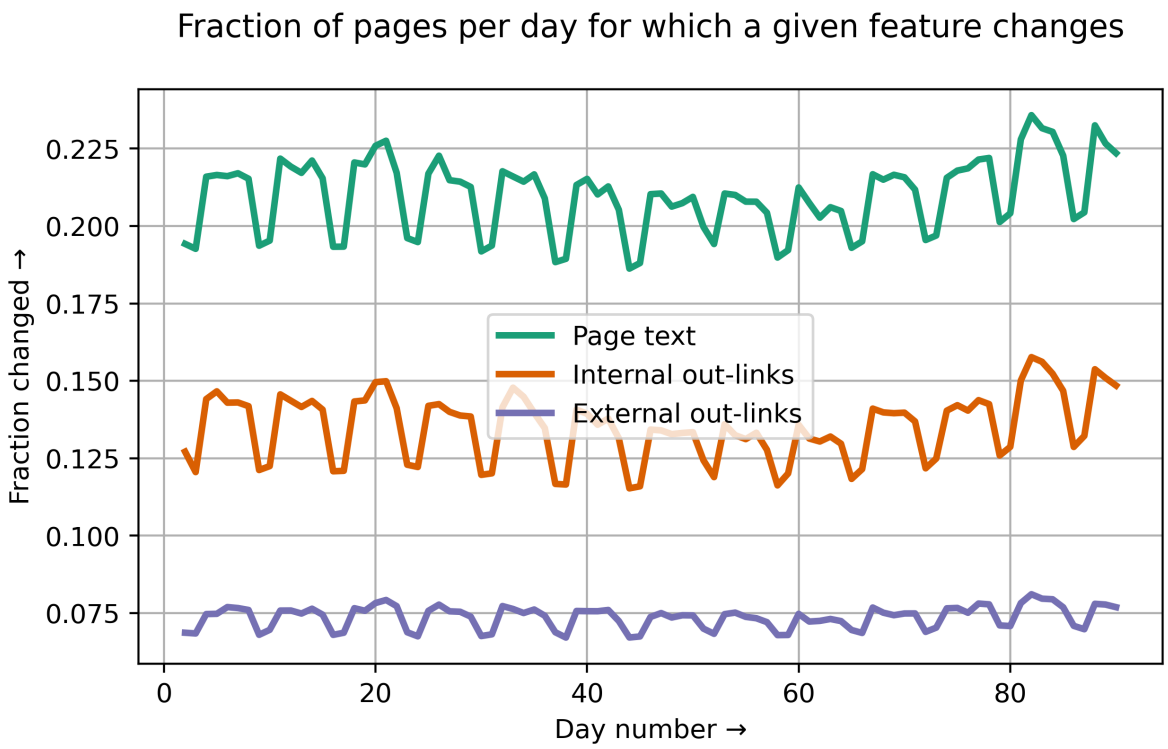

Figure 5.1: Fraction of pages for which a given feature changes

with the results displayed in Figure 4.1 of Chapter 4, since the change fractions of the three features have approximately the same proportions compared to each other as the average change fractions in Figure 5.1. Still, the weekly patterns of both the out-link features are mostly the same as the weekly pattern of the page text.

Weekdays versus weekends Using the adjusted dataset, we zoom in on the phenomenon where pages change more frequently during weekdays compared to during weekends. We do this by picking a weekday (Tuesday) and a weekend day (Sunday), and comparing the respective change fractions for these days. The results are shown in Figure 5.2. As can be observed, for all of the three features the change fractions of the Sundays are consistently below the change fractions of the Tuesdays. This reconfirms the previous finding that pages change more frequently during weekdays than they do over the weekend.

Rarity of change frequencies Next, using the full (not adjusted) dataset, we find out how common it is for the text of a page to change a given number of times over the crawl period. We do this for all pages that were changed at least once during the ninety-day crawl period (101.361 pages). For each of these pages, we compute the fraction of days that the page changed. Afterwards, we sort these fractions in descending order and plot them against percentile of pages that change at least once, as is shown in Figure 5.3. This plot can be read as follows: for a given fraction of days (y-axis), the percentage of pages (out of all pages that change at least once) that changes at least that fraction of

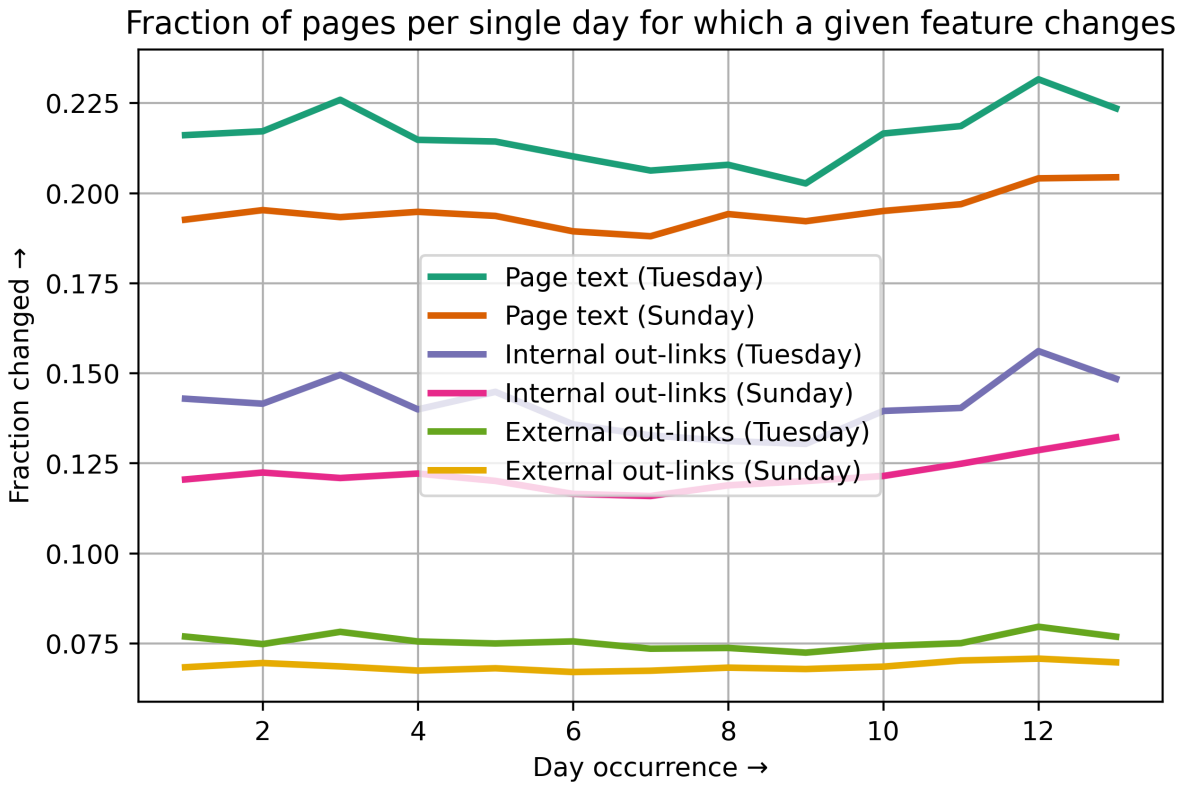

Figure 5.2: Fraction of pages per single weekday for which a given feature changes

the time is equal to the percentile (x-axis).

By reading this plot, we get an idea about the distribution of the number of times that pages change. The first thing we observe is that the distribution of the pages over the change fractions appears relatively spread-out, since the plotted line is close to the diagonal (especially near the beginning). From left to right, the 20% of the pages that change most often change between 100% and 82% of the time. The next 20% of pages cover a bigger portion: these pages change between 82% and 39% of the time. The next 40% of the pages change between 39% and 4% of the time. Finally, the last 20% of pages change in only less than 4% of the cases.

However, when we define a page to change "often" whenever it changes in at least 80% of the cases and "rarely" if it changes in less than 20% of the cases, we find that the majority of pages (68%) change either often (21%) or rarely (47%), with the remaining 32% having a change frequency somewhere in between. This is slightly lower than the percentage that can be taken from the work of Santos et al., where approximately 85% (depending on page topic) of the pages reside in these change frequency intervals [40].

Summary Daily page change frequencies differ between the three features we consider. On a given day, page text changes on $18\% - 24\%$ of the pages, internal out-links change on $12\% - 16\%$ of the pages, and external out-links change on $6\% - 8\%$ of the pages. Nonetheless, all three features exhibit a repeating pattern every seven days,

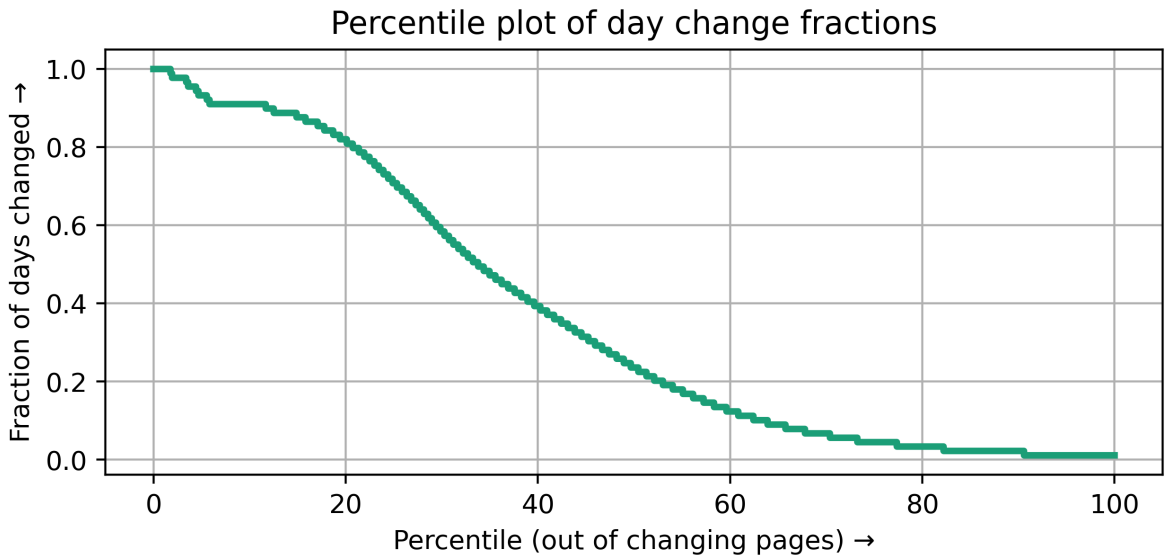

Figure 5.3: Percentile plot of day change fractions

with page features changing more frequently on weekdays as compared to weekend days. This matches the results of Calzarossa and Tessera, who detect the same change weekly pattern in their work [9]. Besides the weekly change patterns, the page change fractions stay mostly constant. Finally, the majority of pages (68%) change either often (at least 80% of the days) or rarely (at most 20% of the days), which is slightly lower than the percentage (approximately 85%, depending on page topic) taken from the work of Santos et al. [40].

5.2 Weekly Changes

Change behaviour Next, we move from daily to weekly timelines, and investigate change behaviours. A "change behaviour" is defined for every single page in a given week, and denotes the number of times that a certain feature on the page changes in that week. Since our crawls were performed daily, the maximum change behaviour is seven (meaning the page feature changed every day), and the minimum is zero (indicating the page feature did not change in that week). We compute the change behaviour for the pages in all full weeks of our crawls, which are weeks 24 up until and including 35 of 2021. As before, we use the adjusted dataset for our analysis.

Alluvial plot Using our definition of change behaviour, we compose an alluvial plot displaying the development of the change behaviours of the page text feature over the weeks of our crawls (see Figure 5.4). This figure can be interpreted as follows. First, each of the twelve columns in the figure represent each of the twelve weeks of the crawls. Each column is divided into eight smaller bars. These smaller bars represent the relative proportions of the different change behaviours. For example, in the first column, the bar of the *0*-behaviour is the largest, followed by *7*-behaviour and the *1*-

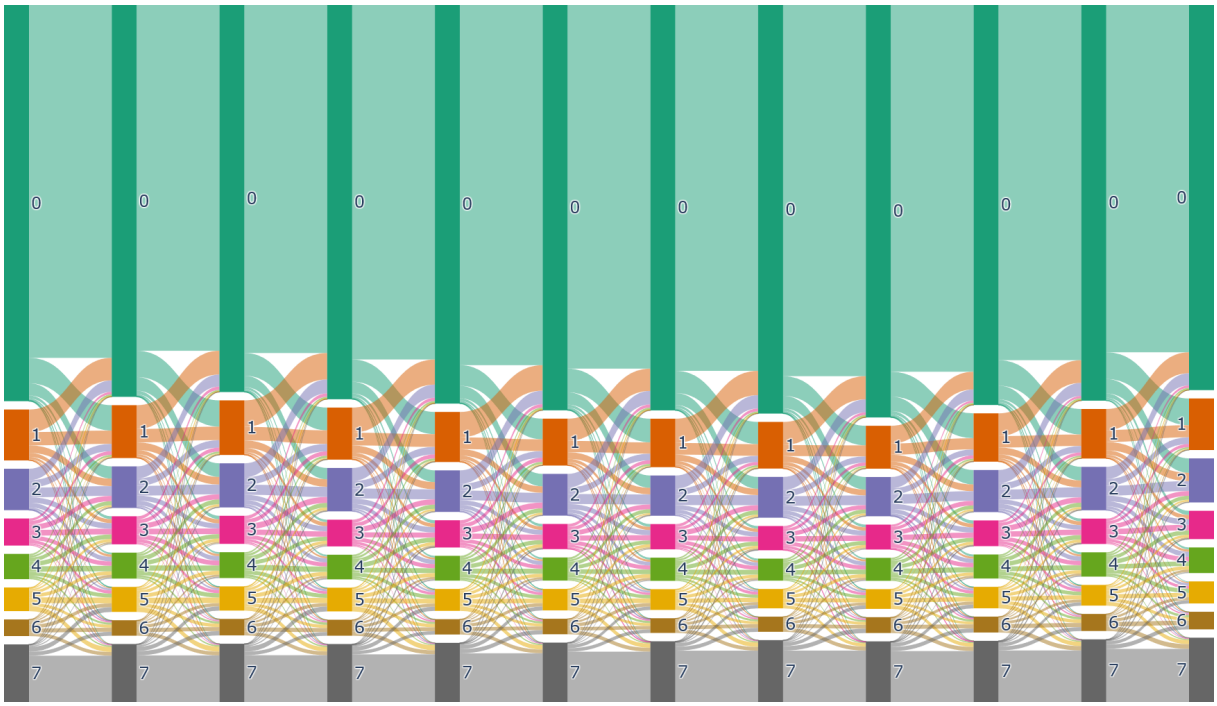

Figure 5.4: Alluvial plot displaying the number of days that page text on pages in the Dutch Web changes per week

behaviour. This means that in the first full crawl week, the majority of the pages never changed, and a smaller fraction of pages changed every day or exactly once. Then the alluvial plot also has "flows" between each consecutive pair of columns. These flows represent the part of a given source bar that is part of a given target bar in the next week. For instance, the flow from the *0*-behaviour in the first column to the *1*-behaviour in the second column denotes the fraction of pages that did not change in the first crawl week, but changed a single time in the next crawl week.

The main conclusions that become apparent from this alluvial plot, are that the proportions of change behaviours do not change much over the weeks, and neither does the development of change behaviours. Fetterly et al. found something similar: they established that for the majority of pages (56%) the change rate is very stable, meaning that week-to-week change is predictable [18]. According to them, this has the implication that past changes of Web pages are a good predictor for future changes.

Average change behaviour The alluvial plot contains a lot of information, which makes it hard to make finer-grained observations. This is why we compose several other plots focusing on specific aspects of the data displayed in the alluvial plot. First, we create a pie chart showing the proportions of the averaged change behaviours (see Figure 5.5). We see here that the majority of pages (roughly 60%) does not change in a given week. The next most frequent change behaviour is seven, followed by one. After that, the probabilities of the change behaviours are approximately sorted with lower change behaviours having a higher probability. This reconfirms the findings of Santos et al., in whose work it can be found that the majority of pages ($\sim 85\%$) change either

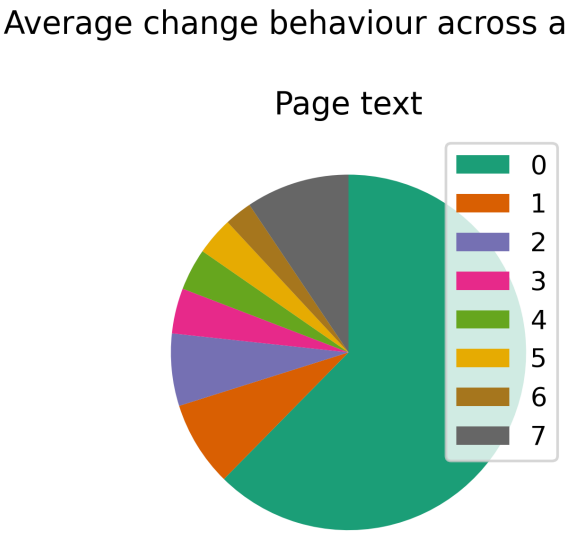

Figure 5.5: Average change behaviour across all weeks (page text)

often (in this case: every day of the week) or rarely (not at all in a given week) [40].

Link change behaviour So far in this section, we focused on change behaviour of page text. But it turns out that the change behaviour of both internal out-links and external out-links behave very similarly, with the main difference being that these features are more likely to not change at all in a given week (the other change behaviours have roughly the same proportions). This allows us to draw similar conclusions for these two features. For completeness, we still show the alluvial plot of the internal out-links change behaviours in Figure 5.6, the alluvial plot of the external out-links change behaviours in Figure 5.7, and the pie charts of the averaged proportions of both change behaviours in Figure 5.8. Nevertheless, in the remainder of this section we will focus on the change behaviour of the page text feature.

Change behaviour per week In Figure 5.9, we show the fractions of each of the change behaviours for all full weeks of the crawls. Again, it becomes apparent here that in each of the weeks, the *0*-behaviour is by far the most common change behaviour. In addition, in the earlier weeks (26-32) the fraction of pages that does not change appears to be increasing. However, in later weeks (33-35) this fraction is decreasing again. This phenomenon might be explained by the fact that these weeks overlap with the national summer holidays in the Netherlands (during which most primary and high school pupils have summer vacation), causing pages to be updated less often during this period. There is, however, one change behaviour that appears to be increasing all throughout our crawls: the *7*-behaviour. This means that the number of pages that change every day in a given week increases every week (although not by very much). This seems counter-intuitive, as one might expect that pages are more likely to be abandoned the older they are. A possible explanation for this increasing fraction could be that the rate

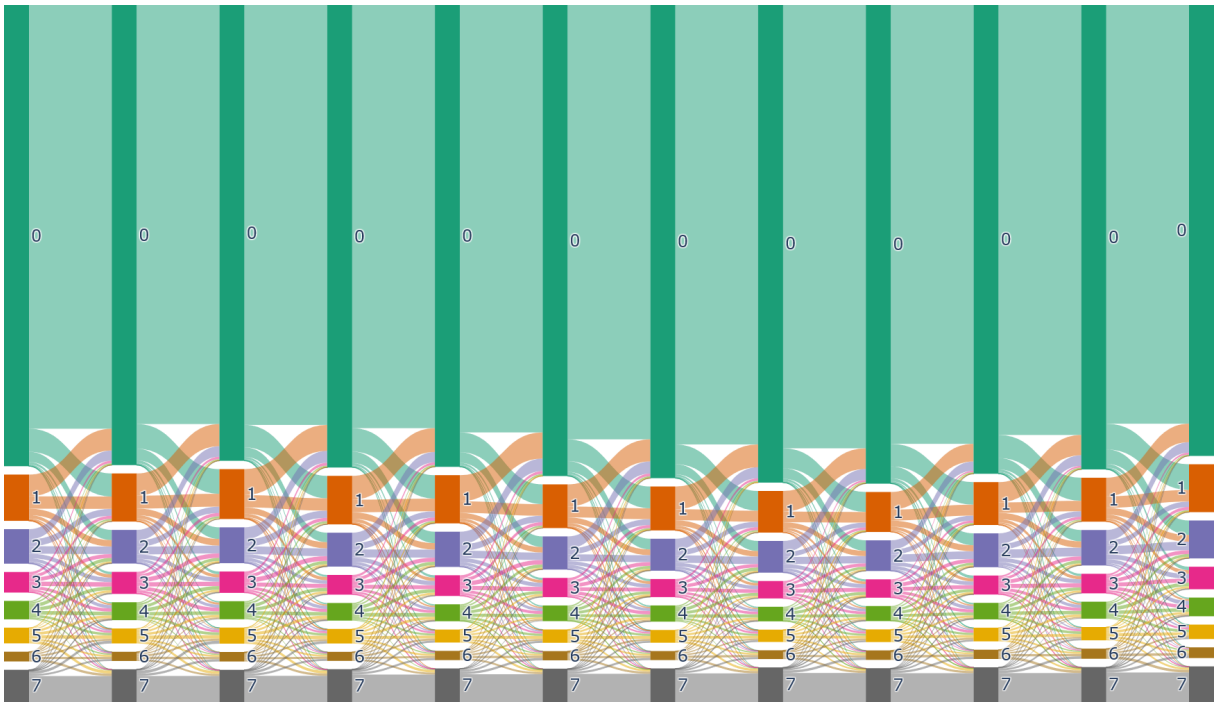

Figure 5.6: Alluvial plot displaying the number of days that internal out-links on pages in the Dutch Web change per week

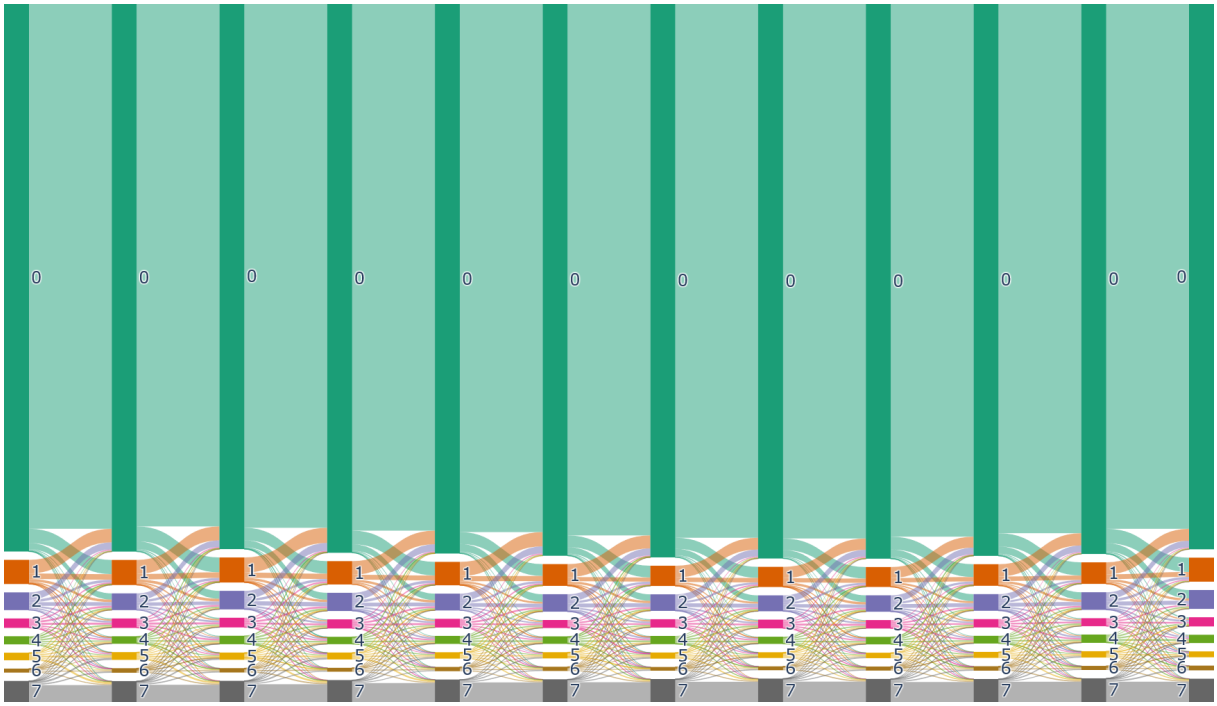

Figure 5.7: Alluvial plot displaying the number of days that external out-links on pages in the Dutch Web change per week

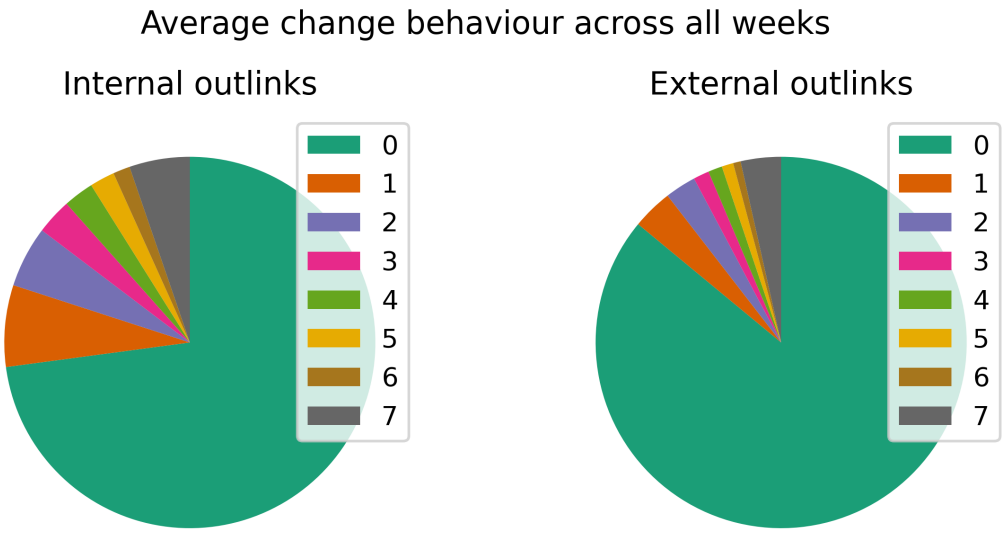

Figure 5.8: Average change behaviour across all weeks (page out-links)

at which third-party embedded page elements are updated speeds up ever so slightly as time goes on. For the other change behaviours, we do not detect any noticeable patterns.

Continuation of change behaviours Next, we are interested in the development of change behaviours over time. For this, we plot in Figure 5.10 for each of the change behaviours the fraction of pages that continue to exhibit that change behaviour in the following week. For example, out of all the pages that do not change in week 24, just below 90% of them also do not change in week 25.

Change behaviour stability From Figure 5.10, we can also derive the stability of change behaviours. We notice that change behaviours zero and seven are the most stable, while the others are not very stable (disregarding the change behaviour "0-6" for now, which will be explained later in this chapter). This can be interpreted in the following way: if a page changes every day or not at all in a given week, the page is very likely to continue exhibiting this behaviour in the following week. But if a page changes a different number of times (somewhere in between zero and seven times), the probability is high that the number of changes in the next week will be different.

Combined continuation of change behaviours In Figure 5.10, we looked at the continuation of change behaviours for every change behaviour individually. But by combining the different change behaviours (as is done in Figure 5.11), we can get an idea about the overall stability of change behaviours. In this graph, we can observe that on average the change behaviours exhibit (slightly) increasing continuation fractions in the summer holiday period (weeks 26-31). This is an interesting finding, because it tells us that not only does the fraction of pages that do not change in a given week increase

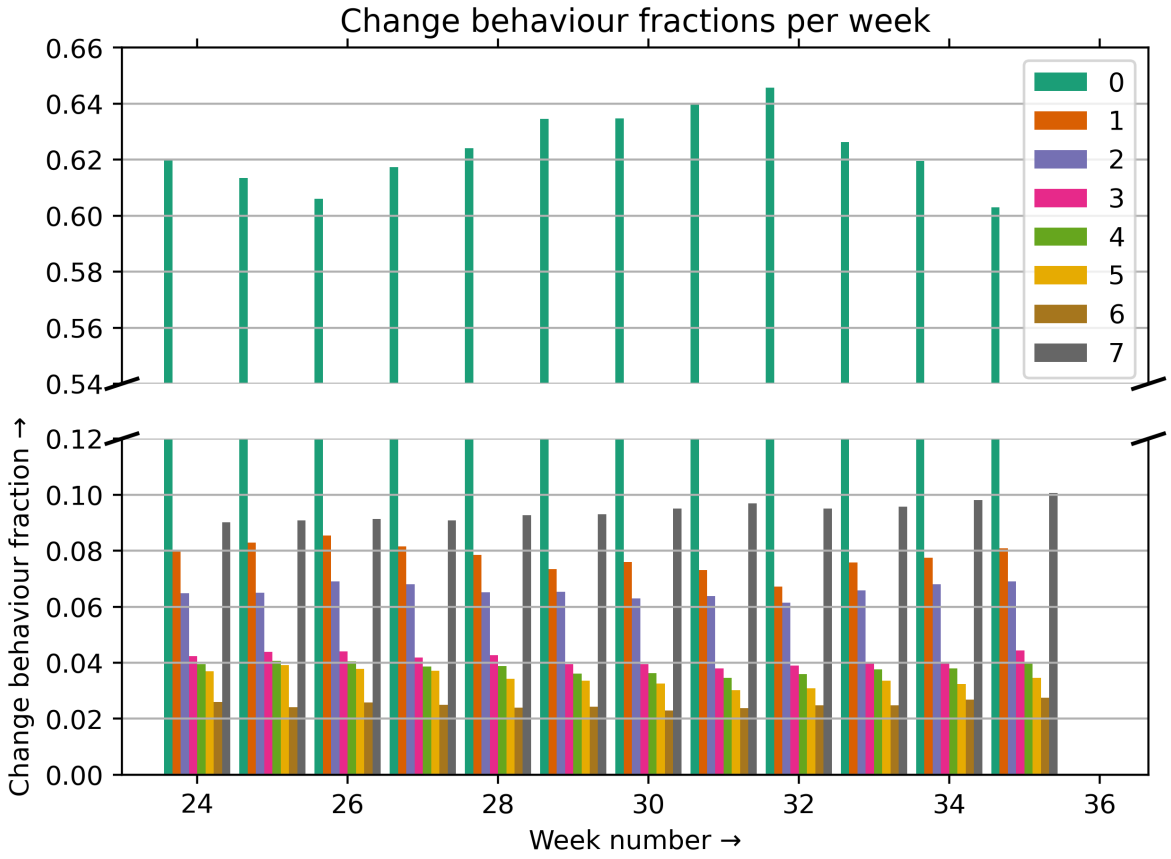

Figure 5.9: Change behaviour fractions per week

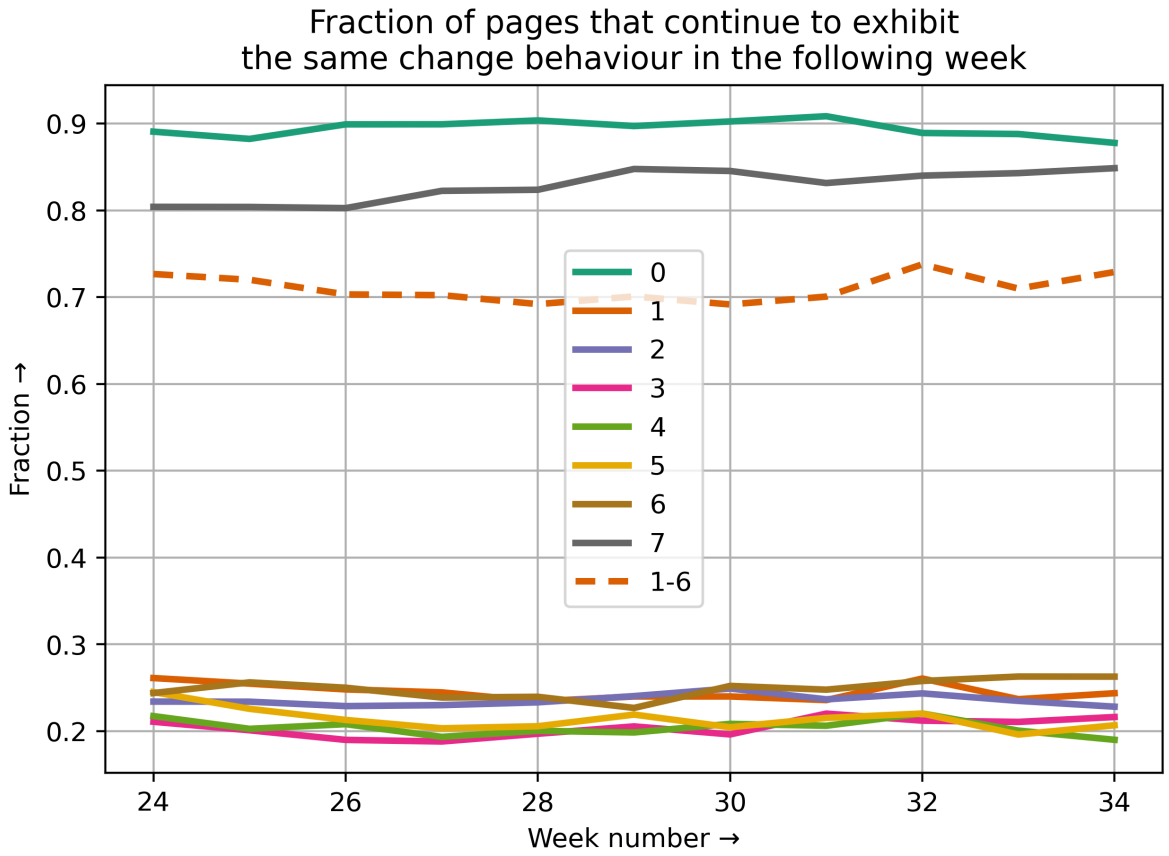

Figure 5.10: Fraction of pages that continue to exhibit the same change behaviour in the following week

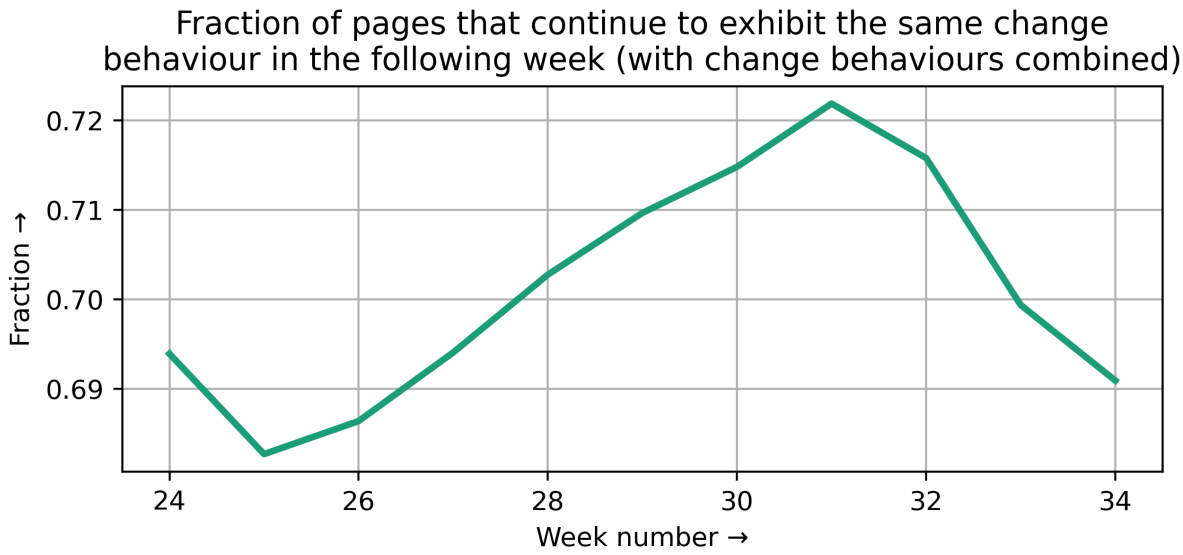

Figure 5.11: Fraction of pages that continue to exhibit the same change behaviour in the following week (with change behaviours combined)

in this period (see Figure 5.9), but also does the overall stability of change behaviours (slightly) increase.

Neighbouring change behaviours From Figure 5.10 it is obvious that change behaviours zero and seven are much more stable than the others. Since the *0*-behaviour and *7*-behaviour are extremes, it makes sense that they change less frequently than the others, for the reason that they can only be either increased or decreased, as opposed to the others which can be both increased and decreased. This begs the question by how much the others change. We answer this question by introducing the concept of "neighbouring behaviours", which we define for each change behaviour n as the fraction of pages that have either change behaviour $n-1$ or $n+1$ in the following week. We compute these values for all crawl weeks and plot the results in Figure 5.12. By inspecting this figure, we notice that for some of the change behaviours, the fraction going to neighbouring change behaviours in the following week is substantial. This is especially true for change behaviour one, and for a lesser degree for change behaviour six. We also note that change behaviours zero and seven have neighbouring fractions quite close to zero, which makes sense given that most of the pages exhibiting these change behaviours continue to do so in following weeks, which leaves not much occasion for these pages to go to neighbouring fractions.

Reduction to three classes It could be argued that the comparison in Figure 5.10 is not completely fair, as not all different change behaviour classes are equally different. Therefore, in addition to the analysis done above where we treat each of the eight classes separately, we also consider the situation where we have just three change classes: pages that never change (behaviour 0), pages that sometimes change (be-

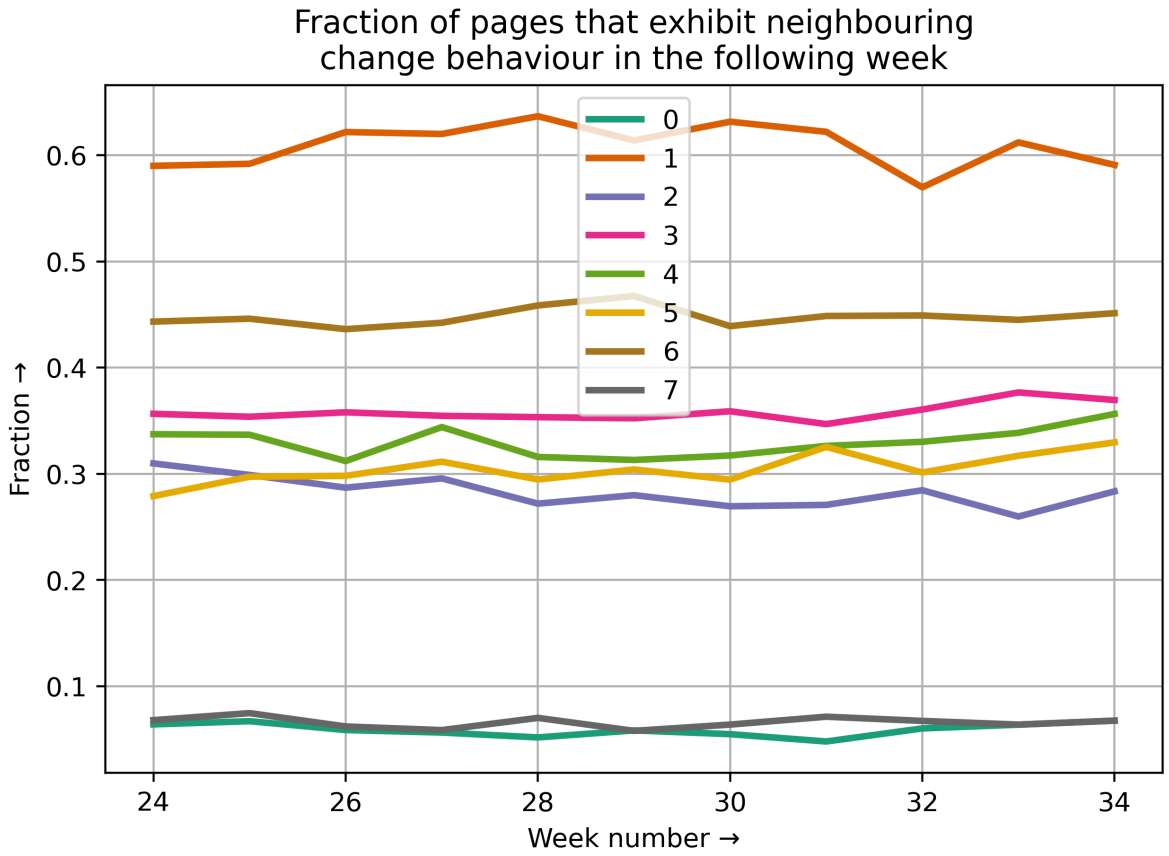

Figure 5.12: Fraction of pages that exhibit neighbouring change behaviour in the following week

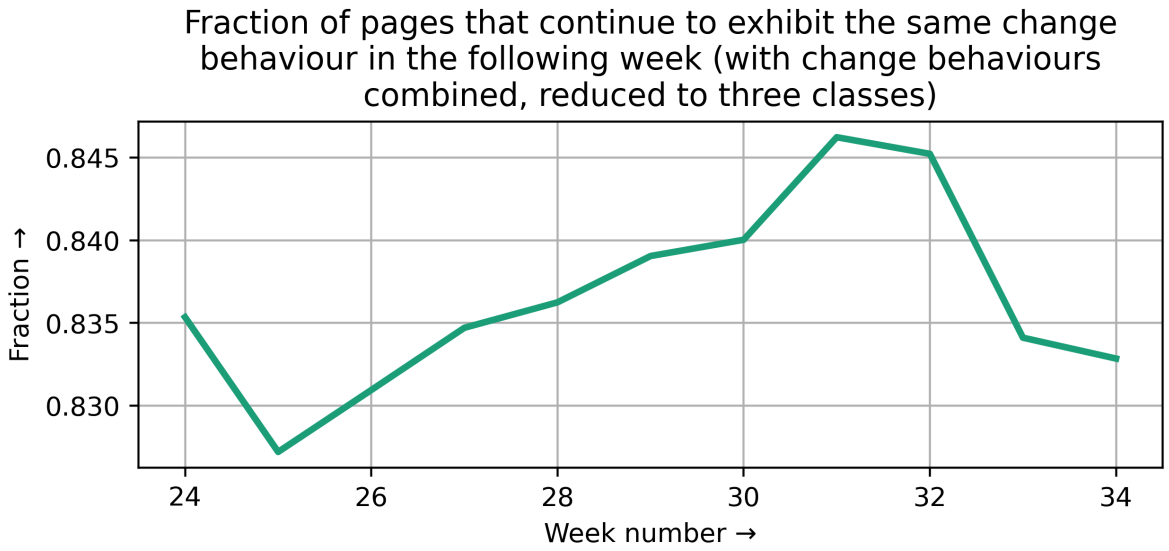

Figure 5.13: Fraction of pages that continue to exhibit the same change behaviour in the following week (with change behaviours combined, reduced to three classes)

haviours 1-6), and pages that always change (behaviour 7). The three-class counterpart of the individual change behaviours 1-6 in Figure 5.10 is plotted by the striped line in Figure 5.10. Obviously change behaviours zero and seven remain the same as before, but the new ("1-6") change behaviour is significantly above the bundle of lines that was displayed at the bottom of Figure 5.10. This indicates that a lot of the time if a page changes between one and six times in one week, it will be in that same range in the next. Still, the striped line in Figure 5.10 is substantially below the other two. For the combined continuation of change behaviours with three classes (see Figure 5.13), there is no prominent difference compared to Figure 5.11, except for the matter that the curve is moved upward (which makes sense, given the fact that also the bundle of lines at the bottom of Figure 5.10 is merged and moved upwards).

Three-class neighbouring change behaviours We also plot the counterpart of Figure 5.12 with three classes in Figure 5.14. Here, we notice that now the curves representing change behaviours zero and seven increase significantly as compared to Figure 5.12, since the list of neighbouring classes is expanded (they now have 1-6 as neighbouring classes instead of just one or six). In addition, the "sometimes" curve in Figure 5.14 is exactly equal to the inverse (one minus the other) of the "1-6" striped curve in Figure 5.10, since the set of pages that change 1-6 times in one week and go to a neighbouring change behaviour (0 or 7 times) in the following week is equal to the set of pages that in the following week do *not* exhibit the same change behaviours (because the set of neighbouring change behaviours comprise all other change behaviours in this case).

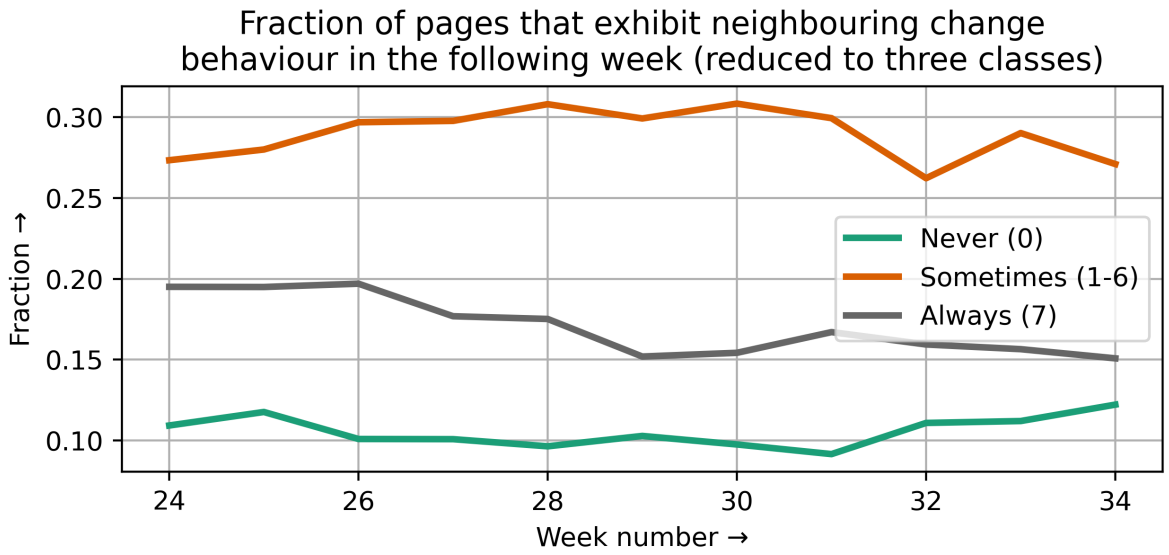

Figure 5.14: Fraction of pages that exhibit neighbouring change behaviour in the following week (reduced to three classes)

Summary Using the definition that a page has changed whenever a given target feature changed[2], we reconfirm that the majority of pages change either often (every day of the week) or rarely (not at all in a week). This is in line with the work of Santos et al. [40], in whose work we can find that the majority of pages (approximately 85%, depending on page topic) changes either often (at least 80% of the days) or rarely (at most 20% of the days). Moreover, this effect is especially great for the "out-links" target features. Over the Dutch national summer holiday period, this effect is even greater, with a higher fraction of pages that change not at all in a given week.

Next, the distribution over change behaviours does not change much over the weeks, neither does the distribution over the development (continuation) of change behaviours. This is true for all three target features we considered. Fetterly et al. found something similar: they established that the change rate is very stable for most pages, meaning that week-to-week change is predictable, and that past changes of Web pages are a good predictor for future changes [18]. Still, we did find some variation in the stability of change behaviours: change behaviour is more stable for pages with extreme change behaviours (that change either every day or not at all in a given week), and during the Dutch national summer holiday period.

[2]Possible target features are page text, internal out-links, and external out-links.

Chapter 6

Prediction using Static Features

In this chapter, we investigate our third research question:

> **RQ3** Using a collection of static page features, how well can we train a Machine Learning model to predict whether the text on pages in the Dutch Web will change?
>
> - **RQ3.1** What type of Machine Learning model is appropriate for this case?
> - **RQ3.2** How can we optimise the performance of our selected Machine Learning model?
> - **RQ3.3** Which features are most important for training the Machine Learning model?
> - **RQ3.4** How well does the trained Machine Learning model perform?
> - **RQ3.5** How does the trained Machine Learning model make predictions?
> - **RQ3.6** What are the limitations of our Machine Learning methodology?
> - **RQ3.7** How can the errors made by the trained Machine Learning model be explained?

The goal of this chapter is to discover how well we can predict whether the text on a page in the Dutch Web will change in the coming day using only information about the current page version.

6.1 Methodology

Random Forest model To investigate the research question, we train two different types of Machine Learning models. First, we train a Random Forest model, since such models usually perform very well [27]. We chose to train a classic Random Forest model

as opposed to a Gradient Boosting Machine (GBM) for the reasons that GBMs are prone to overfitting, and that they require more working memory [29]. Due to the size of the training data, we already run into memory limitations using a classic Random Forest model (as described in Section 6.2), making it hard to meet the additional memory requirements of GBMs.

Logistic Regression model The decisions made by Random Forests tend to be hard to explain, since the decision boundaries cannot be expressed by some formula (the decisions are made by selecting the most-voted candidate by all Decision Trees in the Random Forest [27]). As such, we also train a better explainable model type: the Logistic Regression model [35], whose decision boundaries can be expressed by a logistic expression.

Spark In practice, we train the aforementioned two Machine Learning models using the Spark ML package for Python[1]. Using the Spark ML toolkit for this project was an obvious choice, because our crawls already resided on a Hadoop cluster, and because the size of the dataset is quite big (over 11.5 million datapoints). Using Spark, we can effectively distribute (and hence speed up) the computations over hundreds of cores.

Training pairs To train and evaluate the Machine Learning models, they take as input a collection of training pairs. A training pair is defined every time that a page is successfully crawled for two consecutive days, and corresponds to those two page crawls. It consists of a set of features computed from the first of the two days, and a target value which is computed by comparing the two crawled versions. Since the Dutch Web was crawled for ninety days, a specific page can at most result in 89 training pairs (in case the page was successfully crawled all of the ninety days). In total, $11.622.331$ training pairs are extracted from the crawls.

Target and features In our case, we aim to predict whether the text on a Web page will change. As such, we define the target value for a given day pair as either "change" or "no change", depending on whether or not the text found in the second crawl of the page is different compared to the text of the first crawl of the page. To facilitate the Machine Learning models in learning this target value, we take the nine static features as described in Chapter 3, and convert them into numbers that the models can train on. This results in the following nine features: the number of lines of page text (not counting blank lines), the number of scripts, the number of internal out-links, the number of external out-links, the number of email addresses, the number of images, the number of tables, the number of meta elements, and the number of HTML tags.

[1]We use version 2.4.7 of the Spark library. The documentation of the ML package can be found at `https://spark.apache.org/docs/2.4.7/api/python/pyspark.ml.html`.

Train-test split To be able to objectively evaluate the performance of the Machine Learning models, we split the data into two distinct subsets: a training set and a testing set. The training set (80% of the full dataset) is used to train the models, while the testing set (20% of the full dataset) is used to evaluate the performance of the models. One possibility to make this split would be to set aside a random sample out of the full dataset for verification. However, in that case the resulting performance metrics would not give a good representation of the performance, since a part of the training data would be "leaked" into the testing data. This is caused by the fact that a given page can be in the training data on some days, while being in the testing data on others.

To prevent this, we chose to make the split such that a 20% random sample of the *pages* is set aside for evaluation, while the other 80% is used for training. This ensures that a page is either always in the training data, or always in the testing data. However, note that there is still some indirect (although limited) data leakage: for some domains multiple pages are crawled, and different pages from the same domain are not necessarily all in the training set or all in the testing set. Still, this form of data leakage is limited, because for the majority of domains (67.24%) only one page is crawled, and there is no guarantee that different pages from the same domain behave similarly.

Undersampling The two possible target classes ("no change" and "change") are not equally likely. In fact, 71.50% of the training pairs have the target "no change". A Machine Learning model can pick up on this property, and draw the conclusion that the performance is best when it (almost) always predicts that the text on a page does not change (which in this case would result in an accuracy of 71.50%). This is not desired, as we would like the model to discover other relations hidden within the data. As such, we balance the training data by undersampling all classes other than the target class with the least number of datapoints, such that all have an equal number of datapoints. This is similar to the approach of Barbosa et al., who also undersampled their data such that each of their four target classes ("change clusters") had equal cardinalities [4]. In our case (with two classes), we take a random sample from the "no change" class, such that the two classes have the same number of datapoints in the training data.

Dataset shift Note that by undersampling the "no change" class in the training data, we purposefully introduce dataset shift [36], since it causes the distributions of target classes in the training data to be different than the distribution of target classes in the testing data. However, we do this on purpose, in order not to give either of the two classes an advantage: we want to train the Machine Learning model with an equal number of datapoints per target class, in order to make the prior probability of the classes equal. Still, in doing so, we bias the model against the "no change" class compared to the real-life situation.

Evaluation After training the two types of Machine Learning models using the undersampled training data, we evaluate the performance of the models by computing several

metrics. The main evaluation metric we use is the *minimum recall*, which we calculate using *confusion matrices*. These matrices, in turn, are computed by using the models to make predictions for the datapoints in the testing dataset, and comparing those predictions to the actual target values. In addition to calculating the minimum recall, we also use the confusion matrices to compute class-specific precision values [44], class-specific recall values [44], and balanced accuracy [8, 10].

Minimum recall As mentioned above, using our undersampling approach only the target classes in the training dataset are balanced, while the testing dataset is left untouched. Therefore, if for some reason the Machine Learning model would predict that the text on all pages does not change, the accuracy of the predictions would still be 71.50%, which is not an accurate representation of the quality of the model. One option to prevent this would be to also balance the testing dataset. However, doing so would give an unrealistic view of the testing data, since the testing data is supposed to represent the real-life situation as well as possible.

Instead, a better option would be to use a different performance metric that better represents the quality of the trained model. We selected to use as performance metric the *minimum recall*. In Machine Learning, the *recall* [44] of a given class is defined as the number of datapoints that were correctly classified as that class, divided by the total number of datapoints that are in that class. Using this definition, we define minimum recall as the minimum of such recall values across all (in our case, two) classes.

With the selection of this metric, we treat both classes as equally important. We do this for the reason that we want the model to be unbiased with respect to the target classes: given a page, without looking at the page features we want the probabilities of both classes to be equal. Hence also the undersampling of the training data such that the prior probabilities of the classes are equal in the training data.

Precision, recall, balanced accuracy In addition to calculating the minimum recall, we also compute class-specific precision values, class-specific recall values, and balanced accuracy. For a given class, class-specific precision is defined as the number of correctly classified datapoints of that class divided by the total number of datapoints classified as that class [44]. Next, class-specific recall is defined as the number of correctly classified datapoints of that class divided by the total number of datapoints that are in the class [44]. Finally, balanced accuracy is defined as the average of the class-specific recall values [10].

Hyperparameter tuning As mentioned before, we train two different types of Machine Learning models on our training data. However, even within these model types, there are different possibilities to train them by changing the model's hyperparameters. Examples of hyperparameters are the number of Decision Trees in a Random Forest, or the regularisation penalty in Logistic Regression. By default, Spark ML fixes all hyper-

parameters to certain default values, but using them will not necessarily yield the best model performance. To find out which hyperparameters yield the best performance, for both model types we create a parameter grid with varying hyperparameters and perform *k*-fold Cross Validation [38].

Discovering important features Apart from evaluating the performance of the trained Machine Learning models, we would also like to explain the decisions made by the models, and explain why errors are made. However, since the models are trained using nine features and the predictions can be made using any combination of these nine features, this is complex. To make the act of explanation easier, we attempt to find the features that are most important for making predictions, and use them to illustrate the decisions and errors made by the models. One option to discover the most important features are to use the model-specific feature importances as given by the trained Random Forest model. However, due to the nature of how Random Forests work [27] (multiple trees using different subsets of features), feature importances will be "spread out" over almost all the features. A feature that carries no importance at all will not be affected much by this phenomenon, but a feature that carries *some* importance yet offers no additional information compared to some other feature will still be assigned a significant portion of the importance. Therefore, instead we attempt to discover the most important features by performing Recursive Feature Elimination [21].

RFE Given that there are n features in total, Recursive Feature Elimination (RFE) [21] works by training a Machine Learning model from scratch using all features except one ($n - 1$ features). This is done exactly n times, with every time a different feature being excluded from the training. After the training of the n models is completed, the feature that was excluded in the model with the highest performance is denoted the "least important feature" and is dropped for further iterations. Subsequently, the RFE procedure is invoked recursively on the subset of $n - 1$ features to retrieve the "second least important feature", which is again dropped for future iterations. The recursive invocations continue until there is only one feature left, and hence we have a complete ranking of feature importances. In our case, we perform RFE on a single day-pair worth of datapoints (to make the procedure more computationally feasible) using the Random Forest model type (since this model type yields the best performance).

Using the results of RFE, we train several new Machine Learning models, every time using a different pair of features out of the four most important features. In total, there are six of such pairs, resulting in six newly trained models per model type. The predictions made by these models are an approximation of the predictions made by the original model, but are less expressive, for the reason that they are only trained on two features each. Nevertheless, these approximate models can serve to provide insights into how the original model makes decisions.

Classification maps The main way in which we use the approximate models to explain decisions made by the original model is by generating classification maps (see Figure 6.4 and Figure 6.5). A classification map is a two-dimensional "heatmap" over the feature spaces of two features, which displays for every combination of the two features the prediction that the approximate model would have made for a datapoint having those two features values. In our case, we take as feature space for a given feature all the values in between the 2.5th and 97.5th percentile of values that we encountered for that feature (in order to account for outliers). Furthermore, to give an idea of how the feature spaces relate to the actual datapoints, and to give an indication of how well the approximate models can make predictions, we overlay the classification maps with a sample of the datapoints (whose colours represent the actual targets of the datapoints).

Summary Using over 11.5 million datapoints, we train two types of Machine Learning models to predict whether the text on pages in the Dutch Web will change: a Random Forest model and a Logistic Regression model. The Random Forest model is chosen for its high performance [27], the Logistic Regression model for its explainability. A $80 - 20$ train-test split is made in such a way that data leakage is limited. In addition, the "no change" target class is undersampled in the training data such that the prior probabilities of both classes in the training data are equal (similar to the approach by Barbosa et al. [4]).

The hyperparameters of both model types are optimised using *k*-fold Cross Validation [38]. After having trained both model types, their performance is evaluated by computing several metrics (the main metric being *minimum recall*). Furthermore, classification maps are generated to explain how the models make decisions.

6.2 Results

Random Forest tuning For the Random Forest model type, we create a grid with three hyperparameters:

1. The number of Decision Trees in the Random Forest (either 20 or 50),
2. The maximum depth of the Decision Trees (either 5, 8, 10, 12, or 15), and
3. The minimum number of instances each child node in a Decision Tree must have after a split (either 1, 0.001% of total instances, or 0.01% of the total instances).

Using k-fold Cross Validation with $k = 5$, we find that the model performs best with the maximum number of trees (50), the maximum depth of the trees (15), and the least number of required instances a child node must have after a split (1)[2], with a minimum

[2]Note that although hyperparameter tuning shows us that the model performance is highest with the least number of required instances for each child node, we actually override this value to be higher (as is explained in the next paragraph).

recall on the test set of 78.58%. Model performance might increase a bit further if we were to further increase the number of trees in the forest or the maximum depth of the trees, but in our current setup this is not possible due to memory constraints (the memory requirements scale linearly with the number of trees, and roughly double every time the depth of the trees increase by one).

We note that a minimum required datapoints per node value of one makes the model less generalisable. This is because the hyperparameter tuning does not account for the fact that a small part of the training data may be leaked into the testing data, for the reason that for some domains multiple pages are crawled, which are treated as independent datapoints for the train-test split (as described in Section 6.1). In the worst case, 50 different pages are crawled for a single domain (as is the case for 1.39% of the domains). To make sure that predictions by the model are made for *at least* two domains, we choose to set the minimum required datapoints per node in the trees to 100. This only slightly decreases the model performance to a minimum recall of 78.48%.

Logistic Regression tuning Similarly as for the Random Forest model type, we create a grid with three hyperparameters for Logistic Regression hyperparameter tuning:

1. The solver family used (either binomial or multinomial),
2. The type of applied regularisation penalty (either L_2 [5] or L_1 [26]), and
3. The strength of the regularisation penalty (either 0, 0.01, 0.1, 1, or 10).

Again using k-fold Cross Validation with $k = 5$, we find that the Logistic Regression model performs best using a binomial solver (binomial regression), without applying regularisation (i.e., a regularisation penalty of 0). This happens to be the default configuration of the model set by Spark ML.

Confusion matrices For both the trained Random Forest model and the Logistic Regression model, we use the testing dataset to evaluate the performance. In Table 6.1 and Table 6.2 respectively, we show the confusion matrices of the Random Forest and Logistic Regression models evaluated using the testing data.

Table 6.1: Confusion matrix of Random Forest model

Predicted \ Actual	**no change**	**change**
no change	1.300.926	129.629
change	356.732	532.982

Table 6.2: Confusion matrix of Logistic Regression model

Predicted \ Actual	**no change**	**change**
no change	1.209.769	236.458
change	447.889	426.153

From these matrices, we can compute the precision and recall values per class, as well as the minimum recall and balanced accuracy. In the descriptions below, we will interpret these metrics for both of the models, as well as compare the metrics between the models. Note that the goal of these comparisons is not to discover which of the two models performs better (a Random Forest model will almost always outperform a Logistic Regression model, as the former allows for far more variance), but to place the metrics into context.

Class-specific precision For the Random Forest model, the precision values of the classes "no change" and "change" are respectively 90.94% and 59.90%, and for the Logistic Regression model, the precision values are respectively 83.65% and 48.76%. The first thing we observe, is that for both classes the Random Forest classifier results in a higher precision. In addition, we find that for both models, the "no change" class has a significantly higher precision than the "change" class. This effect is caused by the class imbalance between the two classes, and entails that the probability that a page is falsely classified as "no change" given that it actually changes (false negative rate) is lower than the probability that a page is falsely classified as "change" given that is does not change (false positive rate). In fact, the false positive rate of the Logistic Regression model is even above 50% (since the precision of the "change" class is below 50%), which means that given that the Logistic Regression model predicts that a page will change, the actual probability of the page changing is only 48.76%. In other words: even if the Logistic Regression model predicts that a page will change, it is still the most likely outcome that the page will not change. The phenomenon of a high false positive rate also occurs in other studies, especially when the class imbalance is high, such as with COVID-19 PCR tests [6].

Class-specific recall For the Random Forest model, the recall values of the classes "no change" and "change" are respectively 78.48% and 80.44%, and for the Logistic Regression model, the recall values are respectively 72.98% and 64.31%. Again, we observe that for both classes the Random Forest classifier results in a higher recall. However, in contrast to what was the case with the precision values, none of the classes always (i.e., in both models) has a higher recall value than the other class. Additionally, we find that the recall values of the Random Forest model are much closer to each other than the recall values of the Logistic Regression model, indicating that the Random Forest performs more consistently.

Minimum recall From the class-specific recall values, we can compute our main evaluation metric: the minimum recall. For the Random Forest model, the minimum recall equals 78.48%, while for the Logistic Regression model, the minimum recall is 64.31%. From this, we draw the conclusion that the Random Forest model performs better than the Logistic Regression model. This is in line with our previously calculated performance values, where the values calculated for the Random Forest were consistently

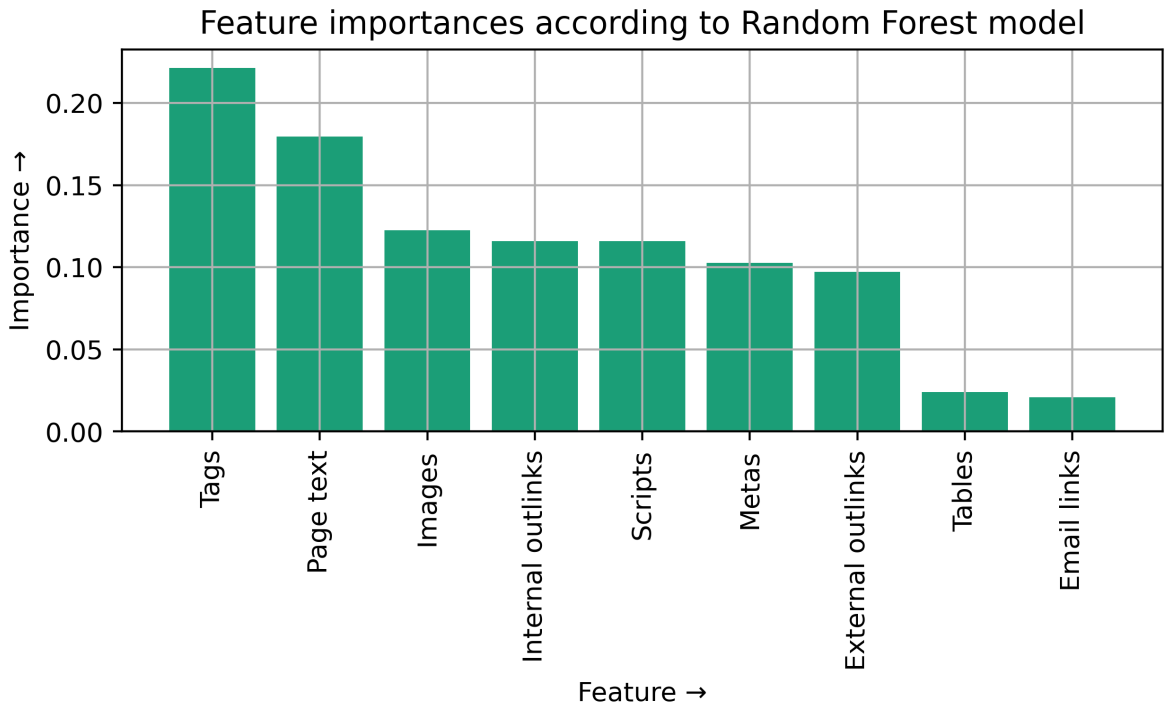

Figure 6.1: Feature importances according to Random Forest model

higher than those calculated for Logistic Regression.

Balanced accuracy The balanced accuracies of the Random Forest and Logistic Regression models are respectively 79.46% and 68.65%. We observe that although both models have a significantly higher balanced accuracy than making random predictions (in which case the balanced accuracy would be around 50%), the Random Forest model has a higher balanced accuracy than Logistic Regression. In addition, we notice that the balanced accuracy (i.e., the average recall) and minimum recall of the Random Forest model are closer to each other than the balanced accuracy and minimum recall of the Logistic Regression model, again indicating that the Random Forest model performs more consistently across classes.

Feature importances From the trained Random Forest model, we can directly extract the feature importances, as is done in Figure 6.1. However, as explained in Section 6.1, due to the nature of how Random Forests work, feature importances will be "spread out" over almost all the features. Still, from the extracted feature importances, we can at least derive the most and least important features for making predictions. From Figure 6.1, we determine that the most important features are HTML tags and page text, and the least important features are email links and tables.

Regression coefficients From the trained Logistic Regression model, we cannot directly extract feature importances. However, we can extract the regression coefficients,

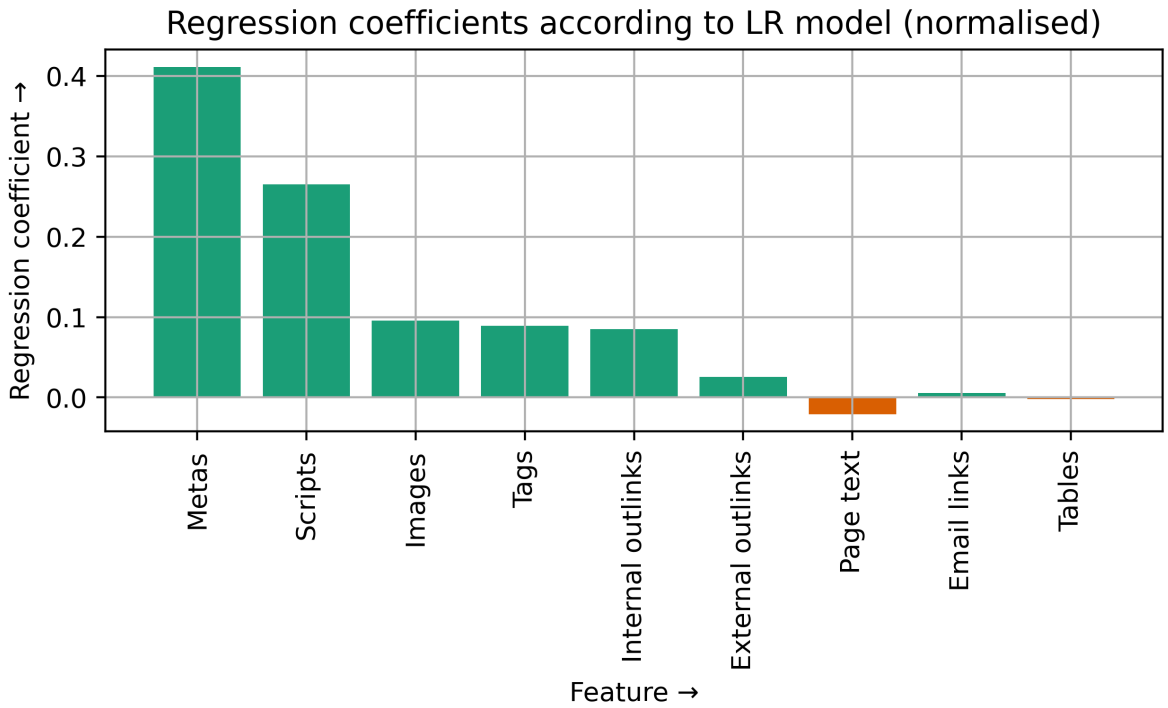

Figure 6.2: Regression coefficients according to Logistic Regression model (normalised)

which make up the decision boundary of the model. The normalised regression coefficients are shown in Figure 6.2. The first thing we note, is that the coefficients can be negative, as is the case for page text and tables. However, to determine the importance of the features in making predictions, the thing we are interested in is the magnitude (absolute value) of the coefficients.

In addition, we observe that the order of the features of the regression coefficients in Figure 6.2 is very different than the order of the feature importances in Figure 6.1. Most surprisingly, whereas page text was one of the most important features for making predictions in the Random Forest model, the fact that its coefficient magnitude is one of the lowest in the Logistic Regression model seems to indicate otherwise. This shows us that the Logistic Regression model makes predictions in a very different manner (utilising different features) than the Random Forest model.

RFE feature rankings Using the Recursive Feature Elimination approach as described in Section 6.1, we find that the feature importance order is the following:

> HTML tags > Page text > Internal outlinks > Scripts > Images > External outlinks > Meta features > Email links > Tables

This order more or less matches the order of feature importances as given by the trained Random Forest model. However, the main thing we are now interested in is by how much the performance of the fully trained model decreases as we drop features. The results of these experiments are plotted in Figure 6.3. As we can see in the figure, the gain in model performance for each subsequent feature added decreases after the

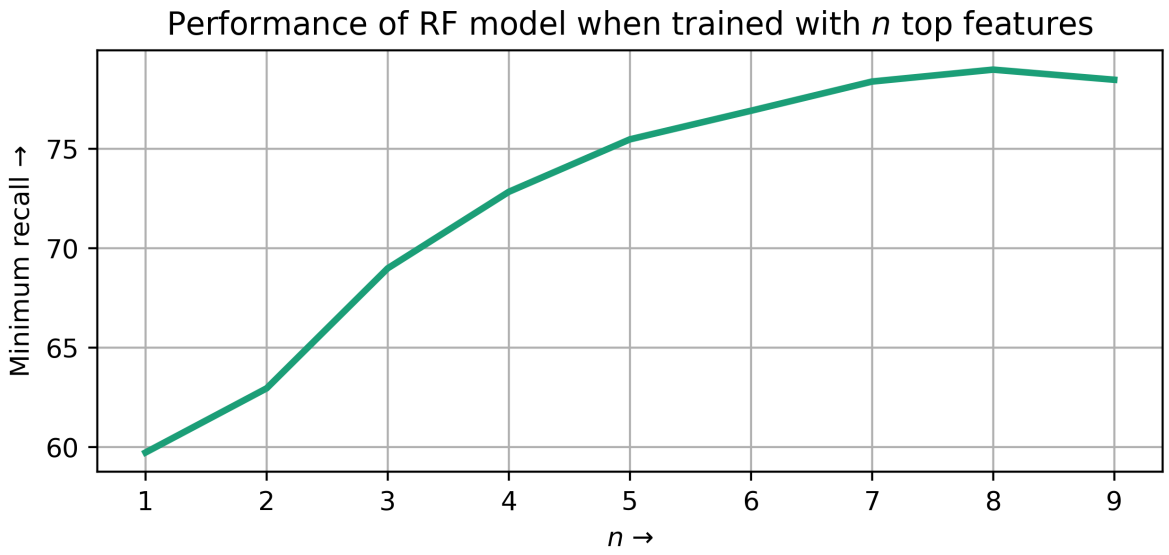

Figure 6.3: Performance of Random Forest model when trained with n top features

second most important feature is added. In fact, there is even a tiny decrease in model performance visible once the ninth feature (tables) is added.

Summary The trained Random Forest model has a significantly higher minimum recall (78.48%) than the Logistic Regression model (64.31%). Also for the other evaluation metrics the Random Forest model consistently outperforms Logistic Regression.

The feature importance order of the trained Random Forest model is more or less the same as the feature rankings according to the Recursive Feature Elimination method [21]. Additionally, both methods agree that "HTML tags" and "page text" are the most important features for making predictions, and that "tables" and "email addresses" are the least important features. Still, model performance decreases once features are dropped from the dataset, with the marginal decrease increasing with every additional feature that is dropped.

6.3 Classification Maps

Feature pair models For the four most important features according to the RFE approach, we train six new Machine Learning models, every time using only two of these four features. Each of these models serves as an approximation of the original model, and are used in an attempt to explain predictions made by the original model. Then, for each of these models, we generate a classification map (see explanation in Section 6.1). For the approximate Random Forest models, the classification maps are shown in Figure 6.4, and for the approximate Logistic Regression models, the classification maps are shown in Figure 6.5.

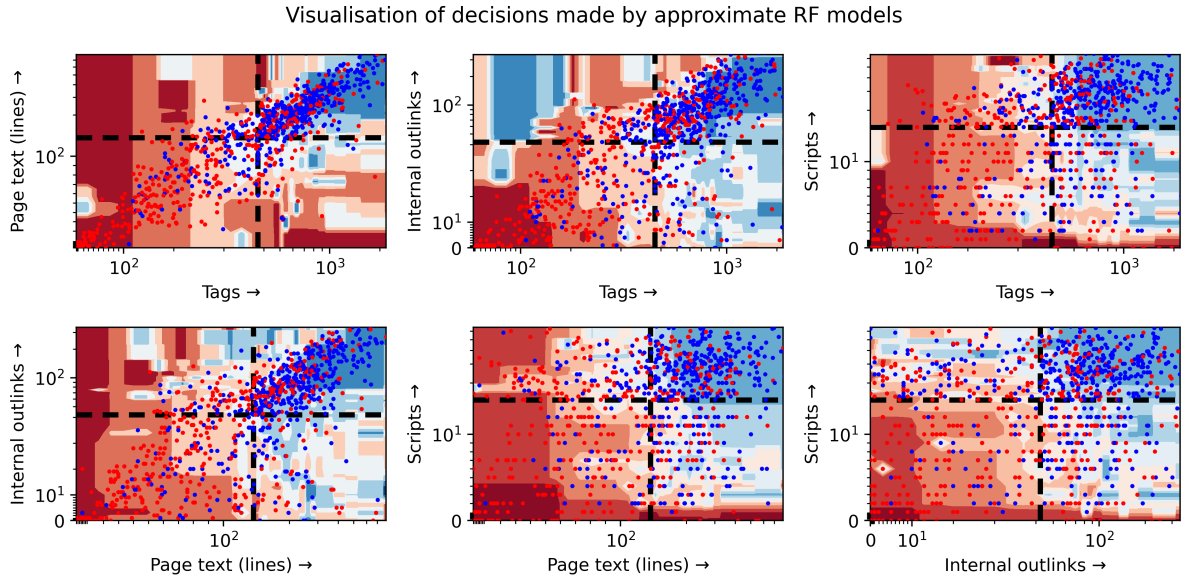

Figure 6.4: Visualisation of decisions made by approximate Random Forest models. A red background indicates that the model predicts that a page does not change for that feature combination, a blue background indicates that it does. The darker the background, the more certain the model is of its prediction. The plots are overlaid with a sample (1000) of the datapoints, with the colour of the dots indicating whether the page which the datapoint represents does change (again, red indicates no change, blue indicates change). Finally, the plots make use of a "symlog" scale: below and left of the black striped lines, a linear scale is used, whereas on the other sides of the black striped lines, a logarithmic scale is used.

Interpretation of classification maps In these plots, the colour of the background indicates the prediction that the model would make for that combination of features: red means that the page does not change, blue means that it does. The darker the background, the more certain the model is of its prediction. In addition, the plots are overlaid with a sample (1000) of the datapoints, with the colour of the dots indicating whether the page which the datapoint represents does change (again, red means no change, and blue means change).

Symlog scale We note that because datapoints tend to be concentrated in the bottom-left corner of the classification maps, these classification plots make use of a "symlog" scale (a mixed linear-logarithmic scale). The switch from linear to logarithmic scale happens whenever a feature reaches the median value we encountered for that feature, and is indicated in the classification maps by black striped lines.

Correct classifications The first thing we observe in the classification plots, is that more than half of the datapoints are correctly classified (i.e., the colour of a dot is the same colour as the background at its location). For the Random Forest classification plots, on average 61.6% of the sample datapoints are correctly classified, and for the

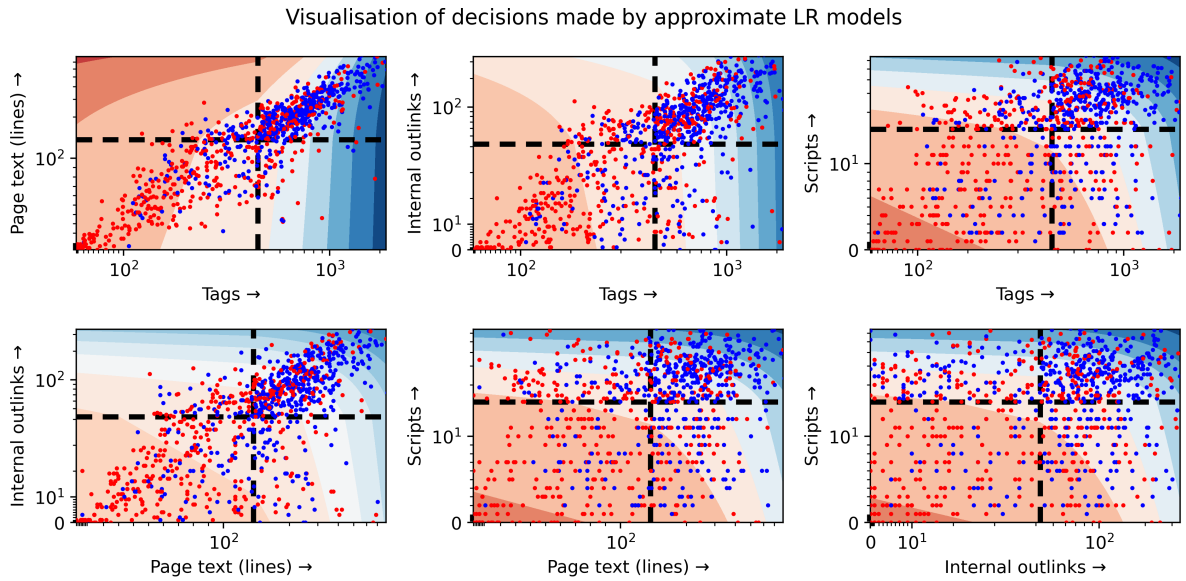

Figure 6.5: Visualisation of decisions made by approximate Linear Regression models. A red background indicates that the model predicts that a page does not change for that feature combination, a blue background indicates that it does. The darker the background, the more certain the model is of its prediction. The plots are overlaid with a sample (1000) of the datapoints, with the colour of the dots indicating whether the page which the datapoint represents does change (again, red indicates no change, blue indicates change). Finally, the plots make use of a "symlog" scale: below and left of the black striped lines, a linear scale is used, whereas on the other sides of the black striped lines, a logarithmic scale is used.

Logistic Regression plots, the fraction of correctly classified sample datapoints equals 62.4%. Surprisingly, the fraction of correctly classified sample datapoints is slightly higher for the Logistic Regression classification plots, indicating that with these feature subset pairs, Logistic Regression is better able to make better predictions. This also highlights the shortcomings of classification plots using only two features each: the models become less expressive (also see the results in Figure 6.3). Still, using these approximate models we can get some insights into how the original models make predictions.

Clearer decision boundaries Second, a striking difference between the Random Forest classification maps (Figure 6.4) and the Logistic Regression classification maps (Figure 6.5) is that the decision boundaries of the Logistic Regression model are much more straightforward than the decision boundaries of the Random Forest. This is due to the nature of how Random Forests work (a prediction is made based on the winner of a majority-vote between different Decision Trees [27]), allowing "pockets" of different prediction areas to be scattered around the map. This is in contrast to how decisions are made by the Logistic Regression model, where the decision boundaries can be expressed by a single curve [35]. This property makes the Logistic Regression better explainable, but at the cost of expressiveness of the model. We witness this trade-off in the performance of the models, with the Random Forest clearly outperforming Logistic Regression (in the case that the model is trained with all features).

Confidence in predictions Third, we recognise from the classification maps that the approximate Random Forest models are overall more certain of its predictions, shown by the darker-coloured backgrounds in the maps. As we would expect, a higher certainty goes hand-in-hand with a higher model performance, which we already observed previously in the performance metric values of the two model types.

Approximate models Fourth, we stress that these classification maps are generated using approximate models (trained using two features each), and hence only display an approximation of the predictions made by the original model. This means that a certain-coloured background at a certain position in the classification map does not guarantee that the original model makes the same prediction at that combination of features. However, the classification maps do give the best approximation of the predictions made by the original model that can be made using only two features.

Concrete insights Finally, we extract insights from these classification maps to give an approximate explanation about how the original models make predictions. As mentioned above, the Logistic Regression classification maps (Figure 6.5) have clear decision boundaries, which means they are well interpretable. As such, we use these maps to provide explanations as to how the models make predictions.

From the classification maps, we make the following observations. Note that these observations are in the order from most significant to less significant, which is consistent

with the RFE feature ranking shown above. Furthermore, we say that a page is *probable* to change whenever the probability of change (within a 24-hour period) is over 50%.

1. The more HTML tags a page has, the more likely it is to change.
   (a) If a page has fewer than ~ 400 HTML tags, then it is not probable to change.
   (b) If a page has in between ~ 400 and ~ 600 HTML tags, then it is probable to change only if it has fewer than ~ 150 lines of page text.
   (c) If a page has more than ~ 600 HTML tags, then it is probable to change, except if it has fewer than ~ 30 scripts.
2. The more lines of page text a page has, the more likely it is to change.
   (a) If a page has fewer than ~ 150 lines of page text, it is not probable to change, except if it has more than ~ 400 HTML tags.
   (b) If a page has in between ~ 150 and ~ 200 lines of page text, it is not probable to change, except if it has more than ~ 600 HTML tags, more than ~ 100 internal outlinks, or more than ~ 30 scripts.
   (c) If a page has more than ~ 200 lines of page text, it is probable to change.
3. The more internal outlinks a page has, the more likely it is to change.
   (a) If a page has fewer than ~ 100 internal outlinks, it is not probable to change, except if it has more than ~ 150 lines of page text, or more than ~ 30 scripts.
   (b) If a page has more than ~ 100 internal outlinks, it is probable to change.
4. The more scripts a page has, the more likely it is to change.
   (a) If a page has fewer than ~ 30 scripts, it is not probable to change.
   (b) If a page has more than ~ 30 scripts, it is probable to change.

From these observations, a clear trend is visible: the higher a feature value, the more likely it is that the page text will change in the coming 24 hours. And since higher feature values in general imply a larger page, we can relate this trend to the observations of Fetterly et al.: they discovered that larger pages change more often than smaller ones [18].

Summary Classification maps are generated in an attempt to explain the predictions made by the Random Forest model. These maps use approximate models trained using only two features each, and hence do not always correctly reflect the predictions made by the original models. Still, we discover a clear trend: the higher a feature value, the more likely it is that the page text will change in the coming 24 hours.

6.4 Limitations

Misclassified datapoints The trained Machine Learning models do not always make the correct predictions. For the Random Forest model, the balanced accuracy of the predictions is 79.46% (meaning that for the two classes on average 20.54% of the datapoints is misclassified), and for the Logistic Regression model, the balanced accuracy of the predictions is 68.65% (meaning that for the two classes on average 31.35% of the datapoints is misclassified). In the classification maps shown above, we can clearly see some datapoints being misclassified. This is visible in both the Random Forest and Logistic Regression maps (Figure 6.4 and Figure 6.5 respectively), where a number of blue dots are on top of a red background, and vice versa. To get perfect predictions by these (approximate) models, the colour of every individual dot would have to match the colour of the background at its location.

Decision boundary freedom However, as expected, using only a single decision boundary (as is the case for Logistic Regression) in a two-dimensional feature space is not enough to achieve proper predictions. On the other hand, a Random Forest model has more freedom in making predictions in the sense that it is not constrained to a single decision boundary. It allows for "pockets" of different coloured prediction areas to be scattered around in otherwise mono-coloured areas in the classification maps (see Figure 6.4). However, even with this additional freedom, a two-dimensional feature space is not enough to make proper predictions.

Approximate maps In addition, we again stress that these classification maps are generated using approximate models (trained using only two features), and hence do not fully reflect the predictions made by the original models. As such, it is more probable that a datapoint is misclassified according to one of these classification maps, than it is according to the original model. Still, even the predictions made by the original models are not perfect.

The original models have nine-dimensional feature spaces, as compared to the two-dimensional feature spaces in the approximate models. But even using all the features is not enough to make (near) perfect predictions, for neither of the two models. For the worst-performing model of the two (Logistic Regression), it appears that the low variance decision boundary is still not expressive enough to properly take into account all the characteristics hidden within the dataset.

Overfitting and underfitting For the Random Forest model, on the other hand, the performance is considerably higher, indicating that this model is able to better account for the traits hidden within the dataset. Nevertheless, the performance is far from perfect, with an average misclassification rate across the two classes of 20.54%. A part of these misclassifications may be attributed to the fact that the model can overfit on parts of the data. We attempted to counteract this issue by requiring every node in

the Decision Trees to have at least 100 instances. But given the fact that there is a possibility that a single page is in the training data 89 times (since the crawls span a period of ninety days), the existence of two "outlier" pages can still cause the model to draw invalid (read: overfitted) conclusions.

Nonetheless, setting the "minimum required instances per node" requirement to a hundred instances may also cause underfitting, since it may prevent the model from picking up on certain rare but repeating patterns hidden within the dataset. Hence, setting this requirement is a trade-off between overfitting and underfitting. Given our reasoning near the beginning of Section 6.2, we believe we made a proper compromise.

Feature expressiveness Moreover, the cause of the imperfect predictions does not only lie in the limited expressiveness of the Machine Learning models, but also in the limited expressiveness of the features. As described in Section 6.1, we only use nine static features extracted from single versions of a given page to attempt predicting whether the text on that page will change. However, it may be the case that this is not enough information to make a proper prediction. Prediction performance may increase if we were to use additional static features, or even dynamic features (meaning that multiple versions of the same page are used to predict whether the text on the page will change).

Human element Finally, there is the human element. Even if we were to have all the possible information that there ever was about a given page, we may still be unable to always make the right prediction. This is because a significant portion of the pages are owned and moderated by people, and people do sometimes diverge from patterns.

Summary Even though the decision boundaries of the Random Forest model allow for more freedom in making predictions than the decision boundaries of Logistic Regression, a two-dimensional feature space (as is the case for the classification maps) is not enough to always make proper predictions. But even the original Random Forest model (with a balanced accuracy of 79.46%) does not make perfect predictions. Some of the prediction errors flow forth from the trade-off we made between overfitting and underfitting, others from the limited expressiveness of the features we use, or even from the fact that people (such as Web site moderators) sometimes diverge from patterns.

6.5 Error Analysis

Accuracy intervals We now focus on the prediction errors made by the original Random Forest model. As mentioned in Section 6.4, this model misclassifies an average of 20.54% of the datapoints for the two target classes. We are interested in whether these errors are spread out evenly across the possible feature values. To investigate this for a given feature, we sort the feature values in ascending order and divide them into ten

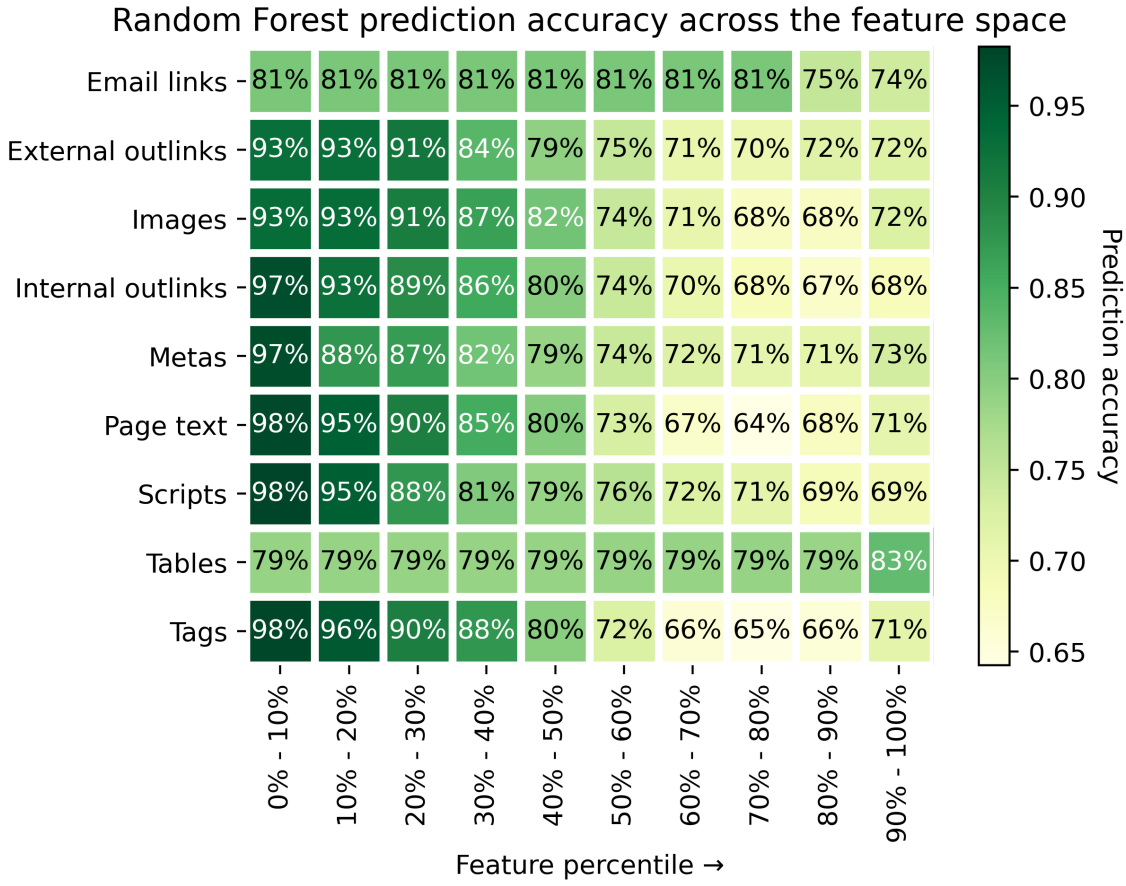

Figure 6.6: Random Forest prediction accuracy across the feature space

consecutive intervals. The first interval corresponds to the lowest 10% of that feature value, the second interval to the next lowest 10% of that feature value (so in between 10% and 20%), and so on. For each of these intervals, we then compute the prediction accuracy on the test data. We repeat this for every feature and show the results in Figure 6.6.

Varying accuracy The first thing we notice, is that the misclassification rate is not the same across the feature space. For all features except email addresses and tables, the difference between the highest and lowest accuracy intervals is over 20 percent points.

Decreasing prediction accuracy Moreover, for all features except tables, the prediction accuracy decreases as the feature value increases. As such, we establish that the prediction accuracy is higher for lower feature values. We attribute this phenomenon to three factors:

1. The lower intervals are smaller in the range of features they cover (since the feature values are more densely distributed at lower values—for instance, there are many more datapoints with a certain feature in the interval $[0, 10)$ then there are in the interval $[100, 110)$—hence also the *symlog* scale in Figure 6.4). And since the higher intervals cover a larger range of features, there is a higher probability that

a few outliers in the training data generate faulty prediction pockets in the trained model. For the lower intervals, these outliers would be cancelled out more easily, since we would encounter more training datapoints having those exact same feature values, which is not the case for the higher intervals.

2. The features of the pages in the lower intervals are more similar, and hence the pages might also behave more similarly. The more the pages look like each other, the more they might behave like each other. In the later intervals, there is a larger difference between the features of a page (i.e., they are more dissimilar), and hence they might also behave more differently. The reason that tables (and to a certain degree email addresses) do not seem to exhibit this phenomenon is that the majority of pages do not have any tables or email addresses. Hence even in the higher intervals, the pages look—and as such also behave—similarly.
3. At low feature values, there is a high certainty that pages do not change. On the other hand, at higher feature values a page may either change or not change. In other words, at high feature values the variance of the target value is higher. This effect can to a certain degree be observed in the classification maps of Figure 6.4, where there are almost exclusively red dots in the bottom-left corners of the maps (low feature values), but there are a number of red dots scattered in the top-right corner of the maps (high feature values).

Higher accuracy at highest intervals Finally, for most features, there is a slight increase in prediction accuracy in the highest one or two intervals. This might be explained by the fact that although the feature values of these pages are quite dissimilar, they share something else: they are the pages with the highest feature values. And because they have such high feature values, they do look similar in some way, and hence also behave similarly.

In addition, these are the very highest feature values, meaning that the likelihood of change is the highest here (since larger pages are more likely to change [18]). This implies a lower variance, and hence an increase in prediction accuracy in these intervals.

Summary The prediction errors made by the trained Random Forest model are not spread out evenly across the feature space. In general, the lower a feature value, the better the model is able to make predictions. We attribute this to three reasons. First, at lower feature values the values are closer to each other, making the model less prone to overfitting in these regions. Second, since the feature values are closer to each other, the pages are more likely to be similar to each other as well—and as such also behave similarly. This also explains the slight jump in model performance at the highest feature values: these pages are slightly more similar to each other in the sense that they are all outliers. Finally, the variance of the target value is higher in the later intervals, making it harder for the model to make good predictions.

Chapter 7

Discussion

In our problem statement (see Chapter 1), we pointed out that search engines need an up-to-date knowledge base of Web pages in order to serve appropriate search results to a user, but that crawling resources are limited. Hence, not all pages can be crawled continuously, and decisions need to be made at what order and intervals to crawl Web pages [43]. This lead to the formulation of three research questions (see Chapter 1), each of which we dedicated a chapter to (see Chapters 4-6). In this chapter (Chapter 7), we assemble the results of the three preceding chapters and discuss the corresponding research questions.

Change Types

The first research question was defined as the following:

> **RQ1** What are the type of changes that pages on the Dutch Web undergo?

We answer this question by providing answers to its five sub-questions.

RQ1.1 *What is the fraction of pages on the Dutch Web that undergoes any change?*

Within a given 24-hour period, over two-thirds (68.7%) of the pages on the Dutch Web undergo some kind of change (see the beginning of Chapter 4).

RQ1.2 *For the pages on the Dutch Web that undergo any change, what is the distribution over change types? Possible change types are changes in page text, scripts, internal out-links, external out-links, email addresses, images, tables, meta elements, and HTML tags.*

Page scripts change most often (40% of the time), followed by text (26%), HTML tags (24%), images (22%), and internal out-links (19%). Email addresses change least often ($< 1\%$), followed by tables (4%), meta elements (5%), and external out-links (7%) (see Section 4.1).

RQ1.3 *Out of the pages on the Dutch Web that change, what fraction contains potentially informative changes?*

Just over half of the page changes are presumably informative (meaning that they add new content or links), the remainder are presumably not. This high portion of presumably uninformative changes matches the findings of Fetterly et al., who discovered that when Web pages change, they often only change in their mark-up or other trivial ways [18] (see Section 4.1).

RQ1.4 *Whenever the readable text on a page on the Dutch Web changes, in what way does it change?*

The readable text of Web pages can change in a lot of different ways. In our research, we encountered changes divided over a total of 23 different change categories. However, not all changes are equally useful. Only in about 36% of the cases, text changes reflect new content being purposefully added by a human being. The rest of the changes can be attributed to automatically generated content, such as visitor counters and advertisements (see Section 4.1).

RQ1.5 *Whenever a feature on a page on the Dutch Web changes, by how much does it change? Possible features are page text, scripts, internal out-links, external out-links, email addresses, images, tables, meta elements, and HTML tags.*

Whenever a page feature changes, how much it changes depends on the feature type. Although rare, when email addresses or tables change, they change a lot. On the other hand, when HTML tags, page text, scripts, or internal out-links change, they only tend to change in small amounts. Nevertheless, for all features, on average just as much is added to the feature as is removed from it. In fact, even the distributions of positive change amplitudes and negative change amplitudes are almost equal (see Section 4.2).

Takeaway By answering the first research question, we retrieved insights about the type of changes that pages on the Dutch Web undergo. In particular, we learnt that not all page changes are equally useful, that is, that they reflect new content or links being added to the page. This finding can help us in deciding the order and priority of page crawls. For instance, we can assign a lower crawl priority to pages that tend to change only in non-important ways.

Change Patterns

The second research question was defined as the following:

> **RQ2** What are the temporal change patterns of pages on the Dutch Web?

We answer this question by providing answers to its four sub-questions.

RQ2.1 *What are the daily page change frequencies of pages on the Dutch Web?*

Daily page change frequencies differ between the three features we consider. On a given day, page text changes on $18\% - 24\%$ of the pages, internal out-links change on $12\% - 16\%$ of the pages, and external out-links change on $6\% - 8\%$ of the pages (see Section 5.1).

RQ2.2 *Do the page change frequencies of pages on the Dutch Web follow specific temporal patterns?*

All three features we consider exhibit a repeating pattern every seven days, with page features changing more frequently on weekdays as compared to weekend days. The same correlation is found by Calzarossa and Tessera [9] (see Section 5.1).

RQ2.3 *To what degree do page change frequencies of pages on the Dutch Web differ across different days?*

The page change fractions stay mostly constant over our ninety-day crawl period (see Section 5.1). In addition, we found that the majority of pages change either very often (every day of the week) or very rarely (not at all in a given week), which is in line with the findings of Santos et al. [40]. Over the Dutch national summer holiday period, this effect is even greater, with a higher fraction of pages that do not change at all in a given week (see Section 5.2).

RQ2.4 *To what extent do pages on the Dutch Web keep exhibiting the same change behaviour?*

The distribution over change behaviours does not change much over the weeks, neither does the distribution over the development (continuation) of change behaviours. This is true for all three target features we considered. Fetterly et al. found something similar: they established that the change rate is very stable for most pages, meaning that week-to-week change is predictable, and that past changes of Web pages are a good predictor for future changes [18]. Still, we did find some variation in the stability of change behaviours: change behaviour is more stable for pages with extreme change behaviours (that change either every day or not at all in a given week), and during the Dutch national summer holiday period (see Section 5.2).

Takeaway By answering the second research question, we retrieved insights about the temporal change patterns of pages on the Dutch Web. In particular, we learnt that most pages tend to keep exhibiting the same change behaviour, and that pages change more frequently on weekdays as compared to weekend days. One possible way to use these insights would be to focus crawling resources on the frequently changing pages during weekdays, and during weekend days (when the frequently changing pages are less likely to change) catch up on crawling infrequently changing pages.

Prediction using Static Features

The third research question was defined as the following:

> **RQ3** Using a collection of static page features, how well can we train a Machine Learning model to predict whether the text on pages in the Dutch Web will change?

We answer this question by providing answers to its seven sub-questions.

RQ3.1 *What type of Machine Learning model is appropriate for this case?*

Using over 11.5 million datapoints, we train two types of Machine Learning models to predict whether the text on pages in the Dutch Web will change: a Random Forest model and a Logistic Regression model. The Random Forest model is chosen for its high performance [27], the Logistic Regression model for its explainability. The reason that we chose to train a classic Random Forest model as opposed to a Gradient Boosting Machine (GBM) is that GBMs are prone to overfitting, and that they require more working memory [29]. Due to the size of the training data, we already run into memory limitations using a classic Random Forest model, making it hard to meet the additional memory requirements of GBMs (see Section 6.1).

RQ3.2 *How can we optimise the performance of our selected Machine Learning model?*

We optimise the performance of both Machine Learning models by optimising their hyperparameters using *k*-fold Cross Validation [38]. In addition, a $80 - 20$ train-test split is made in such a way that data leakage is limited. Moreover, the "no change" target class is undersampled in the training data such that the prior probabilities of both classes in the training data are equal (similar to the approach by Barbosa et al. [4]) (see Section 6.1).

RQ3.3 *Which features are most important for training the Machine Learning model?*

The feature importance order of the trained Random Forest model is more or less the same as the feature rankings according to the Recursive Feature Elimination method [21]. Additionally, both methods agree that "HTML tags" and "page text" are the most important features for making predictions, and that "tables" and "email addresses" are the least important features. Still, model performance decreases once features are dropped from the dataset, with the marginal decrease increasing with every additional feature that is dropped (see Section 6.2).

RQ3.4 *How well does the trained Machine Learning model perform?*

After having trained both model types, their performance is evaluated by computing several metrics (the main metric being *minimum recall*). The trained Random Forest model

has a significantly higher performance (minimum recall of 78.48%) than the Logistic Regression model (minimum recall of 64.31%). Also for the other evaluation metrics the Random Forest model consistently outperforms Logistic Regression (see Section 6.2).

RQ3.5 *How does the trained Machine Learning model make predictions?*

Classification maps are generated in an attempt to explain the predictions made by the Random Forest model. These maps use approximate models trained using only two features each, and hence do not always correctly reflect the predictions made by the original models. Still, we discover a clear trend: the higher a feature value, the more likely it is that the page text will change in the coming 24 hours. This is related to the findings of Fetterly et al., who remark that larger pages are more likely to change [18] (see Section 6.3).

RQ3.6 *What are the limitations of our Machine Learning methodology?*

Even though the decision boundaries of the Random Forest model allow for more freedom in making predictions than the decision boundaries of Logistic Regression, a two-dimensional feature space (as is the case for the classification maps) is not enough to always make proper predictions. But even the original Random Forest model (with a balanced accuracy of 79.46%) does not make perfect predictions. Some of the prediction errors flow forth from the trade-off we made between overfitting and underfitting, others from the limited expressiveness of the features we use, or from the fact that people (such as Web site moderators) sometimes diverge from patterns (see Section 6.4).

RQ3.7 *How can the errors made by the trained Machine Learning model be explained?*

The prediction errors made by the trained Random Forest model are not spread out evenly across the feature space. In general, the lower a feature value, the better the model is able to make predictions. We attribute this to three reasons. First, at lower feature values the values are closer to each other, making the model less prone to overfitting in these regions. Second, since the feature values are closer to each other, the pages are more likely to be similar to each other as well—and as such also behave similarly. This also explains the slight jump in model performance at the highest feature values: these pages are slightly more similar to each other in the sense that they are all outliers. Finally, the variance of the target value is higher in the later intervals, making it harder for the model to make good predictions in these regions (see Section 6.5).

Takeaway As part of answering the third research question, we trained a Machine Learning model, evaluated the performance of the model, explained the predictions made by the model, and inspected the prediction errors. We can use the retrieved model and corresponding analysis to help us in deciding the order and priority of page crawls. For instance, the analysis gives us insights into the relationships between page characteristics and change likelihood, and as such we are able to focus crawling resources on

pages that tend to change frequently. Moreover, we can also use the retrieved model itself to guide the crawling processes of Web crawlers, by prioritising the crawling of pages for which the model predicts that the text will change.

Chapter 8

Conclusion

With this work, we performed daily crawls on a large number of pages representative of the Dutch Web, for a period of ninety days. We performed these crawls using our custom-built Web crawler that circumvents cookie walls. This resulted in a novel large high-quality dataset of the Dutch Web, which we used to perform a large-scale study on the evolution of the Dutch Web across three main strands.

Change types We discovered that over two-thirds of the pages on the Dutch Web undergo some kind of change in a given 24-hour period, and looked into the distribution of these changes over different page features. Next, we found that just over half of these changes are presumably informative (meaning that they add new content of links), and that text changes only reflect new content being purposefully added by a human being in approximately 36% of the cases. Finally, we found that also change amplitudes differ between features.

Change patterns We determined that change frequencies differ significantly between the three features we consider[1], but that they always followed a weekly repeating pattern (with pages changing more frequently on weekdays compared to weekend days). Moreover, we found that page change fractions remained mostly constant over the ninety-day crawl period, that the majority of pages changed either often or rarely, and that the change rates of most pages were stable. Finally, we found that pages changed slightly less frequently over the Dutch national summer holiday period.

Machine Learning We trained two types of Machine Learning models to predict whether the text on pages in the Dutch Web will change in the coming 24 hours. The highest performing one is the Random Forest model, whose hyperparameters were optimised using *k*-fold Cross Validation, resulting in a minimum recall of 78.48%. We also detected that not all features are equally important for making predictions, with the most important features being "HTML tags" and "page text". Finally, we investigated

[1]The three features we consider are *page text*, *internal out-links* and *external out-links*.

the prediction errors made by the model, and identified that the model is worse at making predictions for higher feature values.

Findings such as these give us insights into the relationships between page characteristics, change likelihood, and prediction accuracy. These insights can then be used by search providers to optimise their search engines, for example by focusing crawling resources on pages that tend to change often. The model trained as part of this research can itself also be used to guide the crawling processes of Web crawlers: Web crawlers can prioritise crawling the pages for which the model predicts that the text will change.

8.1 Future Work

Dynamic features The next logical step to improve the prediction performance of our Machine Learning model would be to also use dynamic features in addition to the static features already used. Examples of such features are the number of times that a given (static) feature changed in the past week. This makes the features more expressive, and hence might increase the prediction performance. Additionally, it might be interesting to then not only use dynamic features, but also try to predict a dynamic target value, such as the (local or global) change rate of a page.

Web semantics In a slightly different vein: useful insights may be found by investigating the semantics of Web pages. We have access to a high-quality corpus of over 11.5 million crawled Web pages, giving us more than enough data to work with. Furthermore, these extracted insights might then even be used as training features to increase the prediction performance of Machine Learning models.

Different locales Next, as is one of the main novelties of this work, this is the first large-scale study on the dynamics of the Dutch Web. However, as mentioned in Section 2.5, most studies so far use more generalised data sets (attempting to capture the dynamics of the Web as a whole). Still, it might be interesting to repeat our study for the Web of a different country, to discover to what degree the insights we found are the same across the world.

Change models A more theoretical branch of Web research focuses on modelling the changes of the Web as statistical distributions (see Section 2.2.1). In particular, the change frequency of Web pages is often modelled according to Poisson process models [12, 13, 24, 42]. It will be interesting to explore to what degree such models are applicable to the pages within our large high-quality dataset of the Dutch Web.

Custom crawlers Finally, one of the practical applications of our research is that it can be used to optimise the allocation of crawling resources. Future work could set out to create a custom Web crawler, that uses our insights to focus resources on pages that

change frequently, and then evaluate to what degree this increases the “freshness” [2] of the crawled pages.

Bibliography

[1] Eytan Adar, Jaime Teevan, Susan T Dumais, and Jonathan L Elsas. The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 282–291, 2009.

[2] Konstantin Avrachenkov, Kishor Patil, and Gugan Thoppe. Change rate estimation and optimal freshness in web page crawling. In *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 3–10, 2020.

[3] Ziv Bar-Yossef, Andrei Z Broder, Ravi Kumar, and Andrew Tomkins. Sic transit gloria telae: towards an understanding of the web's decay. In *Proceedings of the 13th international conference on World Wide Web*, pages 328–337, 2004.

[4] Luciano Barbosa, Ana Carolina Salgado, Francisco De Carvalho, Jacques Robin, and Juliana Freire. Looking at both the present and the past to efficiently update replicas of web content. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 75–80, 2005.

[5] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[6] Glenn D Braunstein, Lori Schwartz, Pamela Hymel, and Jonathan Fielding. False positive results with sars-cov-2 rt-pcr tests and how to evaluate a rt-pcr-positive test for the possibility of a false positive result. *Journal of Occupational and Environmental Medicine*, 63(3):e159, 2021.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[8] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *2010 20th international conference on pattern recognition*, pages 3121–3124. IEEE, 2010.

[9] Maria Carla Calzarossa and Daniele Tessera. Modeling and predicting temporal patterns of web content changes. *Journal of Network and Computer Applications*, 56:115–123, 2015.

[10] Henry Carrillo, Kay H Brodersen, and José A Castellanos. Probabilistic performance evaluation for multiclass classification using the posterior balanced accuracy. In *ROBOT2013: First Iberian Robotics Conference*, pages 347–361. Springer, 2014.

[11] CERN. A short history of the web. `https://home.cern/science/computing/birth-web/short-history-web`. Accessed: 2021-07-13.

[12] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems (TODS)*, 28(4):390–426, 2003.

[13] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)*, 3(3):256–290, 2003.

[14] McKinsey & Company. How covid-19 has pushed companies over the technology tipping point–and transformed business forever. `https://mck.co/3nlK8b2`, 2020. Accessed: 2021-11-02.

[15] Intersoft Consulting. General data protection regulation. `https://gdpr-info.eu`. Accessed: 2021-07-13.

[16] Statista Research Department. Share of households with internet access in the Netherlands from 2007 to 2020. `https://www.statista.com/statistics/377747/household-internet-access-in-the-netherlands`, 2021. Accessed: 2021-07-13.

[17] Jonathan L Elsas and Susan T Dumais. Leveraging temporal dynamics of document content in relevance ranking. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 1–10, 2010.

[18] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L Wiener. A large-scale study of the evolution of web pages. *Software: Practice and Experience*, 34(2):213–237, 2004.

[19] Carrie Grimes. Microscale evolution of web pages. In *Proceedings of the 17th international conference on World Wide Web*, pages 1149–1150, 2008.

[20] Kanik Gupta, Vishal Mittal, Bazir Bishnoi, Siddharth Maheshwari, and Dhaval Patel. Act: Accuracy-aware crawling techniques for cloud-crawler. *World Wide Web*, 19(1):69–88, 2016.

[21] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1):389–422, 2002.

[22] Caroline Haythornthwaite. Introduction: The internet in everyday life, 2001.

[23] Joseph Johnson. Global digital population as of January 2021. `https://www.statista.com/statistics/617136/digital-population-worldwide`, 2021. Accessed: 2021-07-13.

[24] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric Horvitz. Staying up to date with online content changes using reinforcement learning for scheduling. 2019.

[25] Eleni Kosta, Christos Kalloniatis, Lilian Mitrou, and Evangelia Kavakli. Search engines: gateway to a new "panopticon"? In *International Conference on Trust, Privacy and Security in Digital Business*, pages 11–21. Springer, 2009.

[26] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y Ng. Efficient l˜ 1 regularized logistic regression. In *Aaai*, volume 6, pages 401–408, 2006.

[27] Yanli Liu, Yourong Wang, and Jian Zhang. New machine learning algorithm: Random forest. In *International Conference on Information Computing and Applications*, pages 246–252. Springer, 2012.

[28] Lakmal Meegahapola, Vijini Mallawaarachchi, Roshan Alwis, Eranga Nimalarathna, Dulani Meedeniya, and Sampath Jayarathna. Random forest classifier based scheduler optimization for search engine web crawlers. In *Proceedings of the 2018 7th International on Software and Computer Applications*, pages 285–289, 2018.

[29] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[30] Norman H Nie and Lutz Erbring. Internet and society: A preliminary report. *IT & society*, 1(1):275–283, 2002.

[31] NIFO. Digital government factsheets - 2019. `https://joinup.ec.europa.eu/collection/nifo-national-interoperability-framework-observatory/digital-government-factsheets-2019`, 2019. Accessed: 2022-04-15.

[32] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What's new on the web? the evolution of the web from a search engine perspective. In *Proceedings of the 13th international on World Wide Web*, pages 1–12, 2004.

[33] Christopher Olston and Sandeep Pandey. Recrawl scheduling based on information longevity. In *Proceedings of the 17th international on World Wide Web*, pages 437–446, 2008.

[34] Gautam Pant and Padmini Srinivasan. Predicting web page status. *Information Systems Research*, 21(2):345–364, 2010.

[35] Chao-Ying Joanne Peng, Kuk Lida Lee, and Gary M Ingersoll. An introduction to logistic regression analysis and reporting. *The journal of educational research*, 96(1):3–14, 2002.

[36] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. Mit Press, 2008.

[37] Kira Radinsky and Paul N Bennett. Predicting content change on the web. In *Proceedings of the sixth ACM international on Web search and data mining*, pages 415–424, 2013.

[38] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. *Encyclopedia of database systems*, 5:532–538, 2009.

[39] Myriam Ben Saad and Stéphane Gançarski. Archiving the web using page changes patterns: a case study. *International Journal on Digital Libraries*, 13(1):33–49, 2012.

[40] Aécio Santos, Bruno Pasini, and Juliana Freire. A first study on temporal dynamics of topics on the web. In *Proceedings of the 25th International Companion on World Wide Web*, pages 849–854, 2016.

[41] Aécio SR Santos, Cristiano R de Carvalho, Jussara M Almeida, Edleno S de Moura, Altigran S da Silva, and Nivio Ziviani. A genetic programming framework to schedule webpage updates. *Information Retrieval Journal*, 18(1):73–94, 2015.

[42] Sanasam Ranbir Singh. Estimating the rate of web page updates. In *IJCAI*, volume 7, pages 2874–2879, 2007.

[43] Qingzhao Tan and Prasenjit Mitra. Clustering-based incremental web crawling. *ACM Transactions on Information Systems (TOIS)*, 28(4):1–27, 2010.

[44] Luis Torgo and Rita Ribeiro. Precision and recall for regression. In *International Conference on Discovery Science*, pages 332–346. Springer, 2009.