BSc Thesis Applied Mathematics

Predicting New Web Pages on the World Wide Web Using Topological Features

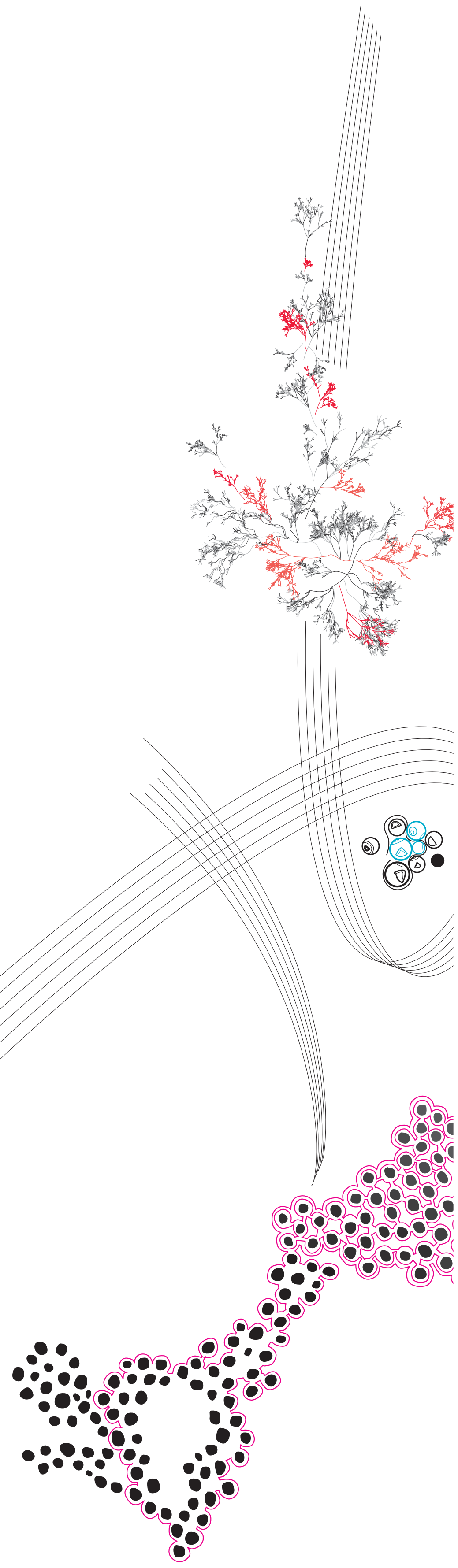
J.J. Sustronk

Supervisor: N. Litvak & D. Bucur

July, 2019

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.



Preface

This thesis is being written as part of the Bachelor program of Applied Mathematics and Technical Computer Science at the University of Twente.

A little over 3 months ago, I started this research with little knowledge about machine learning and the structure of networks. I received the very broad task of finding new information on the World Wide Web. At the time, I had no clue how to tackle this seemingly enormous problem. Step by step, I gained more knowledge and the research got more structured. Now, the research is finished, and the results are better than I ever expected. On top of that, and maybe even more important, I gained an incredible amount of new knowledge about how to conduct research, machine learning, big data processing and the way the Internet behaves. None of this would have been possible without my marvelous support group.

First and foremost, I would like to thank my amazing supervisors N. Litvak and D. Bucur. They offered me valuable advice whenever I needed it and shared my joy and excitement when new milestones were achieved. Furthermore, a special thanks to G. Pitel of eXenSa who contributed to this research by sharing his data and helpful insights. Lastly, I would like to thank my family and friends, who supported me through thick and thin while completing my bachelors.

I hope you enjoy reading this final work of my bachelor program.

Kind regards,

Jasper Sustronk

July the 5th, 2019

Predicting New Web Pages on the World Wide Web Using Topological Features

J.J. Sustronk

July, 2019

Abstract

Every day, numerous new web pages are created on the World Wide Web. These new web pages may contain information relevant to many people. However, finding these new pages is a non-trivial task. This research focuses on finding a function based on topological features that predicts where in the World Wide Web new pages can be found. Using various supervised Machine Learning techniques we show that we can predict with an accuracy and precision of over 90% where new pages can be found. The Random Forest algorithm obtained the highest performance.

Keywords: Web Crawling, Web Change Prediction, Machine Learning

1 Introduction

Since the start of the World Wide Web, the number of web pages has been increasing tremendously. With this growth of pages, new information gets released continuously. However, finding these new web pages is a non-trivial task. Traditionally, the World Wide Web is being scraped by web crawlers to extract information. However, scraping (large portions of the) internet is expensive and thus not suitable for finding just new pages.

In recent years, researchers have conducted work in the field of focused crawlers [1, 14, 5]. These crawlers utilize an evaluation function to predict where relevant information is stored in the web. The function allows a crawler to crawl only the pages that the function classifies as interesting. Research shows that this method can save a considerable amount of resources.

Focused crawlers attempt to find pages relevant to a specific topic. However, in this paper we are interested in finding *new* pages on the web. We attempt to find a similar evaluation function that can predict which pages have a high probability of linking to a new page. This function will be created using several topological features.

The remainder of this paper is structured as follows: Section 2 displays the research questions formulated for this research. In section 3 more information about the World Wide Web, Crawlers, Machine Learning, and the data set is given. Section 4 highlights the work that has already been done in this field and section 5 elaborates upon the notations used in this research. Section 6 gives an overview of how the change rate of pages can be predicted. Section 7 shows the approach of this research, and section 8 the results. Lastly, the results are discussed in section 9 and 10, and guidance for future work is given in section 11.

2 Research Questions

To guide our research, the following research questions have been defined. The first question is the umbrella question, the others act as subquestions.

- **RQ 1**: How can we predict where new pages will be created on the World Wide Web?
  - **RQ 1.1** Which Machine Learning models can be used to predict where new pages will be created on the World Wide Web?
  - **RQ 1.2**: Which topological features will contribute the most to finding new web pages on the World Wide Web?

3 Background

3.1 World Wide Web

We consider the World Wide Web to be a simple graph $G = (V, E)$. Each vertex $v \in V$ corresponds to a web page. Edges $e \in E$ are directed and correspond to the hyperlinks between the web pages. If there are multiple hyperlinks from one page to another, only one edge is present in G. A website is considered a collection of web pages. These websites are often identified with their domain, the first part of the URL of the website after the scheme (excluding 'www'). For instance, the domain of the web page `https://www.utwente.nl` is `utwente.nl`. Moreover, domains themselves can be divided in sub-domains, for instance `inter-actief.utwente.nl`.

The web is evolving continuously. Consider web page 2 displayed in figure 1. Web page 2 is linked to by page 0 and 1 (these edges are called inlinks). Furthermore, web page 2 has multiple outlinks to other web pages. At a future point in time, the neighbourhood around web page 2 may change. The following possibilities exist (see figure 2):

1. An new edge can be added pointing to an existing page (3)
2. A vertex can be deleted from the graph (4)
3. An edge and vertex can remain in the graph (5)
4. An edge can be deleted (6)
5. A new vertex can be added and linked to by web page 2 (7)

In this research, we only focus on cases 1 and 5.

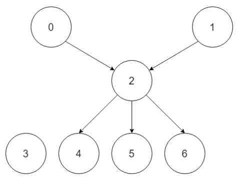


FIGURE 1: Graph at time t

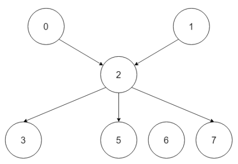


FIGURE 2: Graph at time $t' > t$

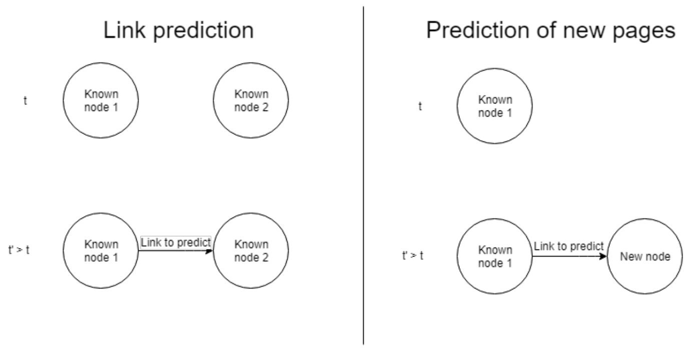


FIGURE 3: Comparison link prediction and prediction of new pages

3.2 Crawler

A crawler is a program that visits (web)pages in order to extract information. A crawl starts at the URL of the initial page p_0. This page is visited and extracts all the URLs present on that page and appends the yet unvisited pages to a queue Q. The crawler continues visiting the pages that are present in Q. All pages visited by the crawler are scanned and the relevant information is extracted. What information is relevant depends on the purpose of the crawler. For instance, a crawler can extract text, images, email addresses, files, etc.

Notice that a crawl can be started with a non-empty queue. Pages in the queue at the start of the crawl are often called seeds. These seeds allow crawling disconnected parts of the internet and/or finding relevant information earlier by starting with important domains.

3.3 Link Prediction

Link Prediction is a research area connected to social network analysis. The graph $G = (V, E, t)$ represents a social network at a particular time t in which each edge $e = \langle u, v \rangle \in E$ corresponds to a social interaction. Link prediction aims to predict new links between vertices in G at a future time $t' > t$. Examples of link prediction is suggested friends on social media websites and suggested products to buy in an online store.

Common features used in link prediction are based on the topology of the graph. For example, common neighbors and shortest path(s) between two nodes are often used.

Notice that there is a significant difference between link prediction problems and the problem this research tries to solve. In the case of Link prediction, the features are based on combination of known nodes. However, in this research, we focus in finding a link to a unknown page that might not even exist yet (see figure 3). Hence, features can only be calculated based on the properties of a single node.

That being said, there are striking similarities between link prediction and the prediction of new pages. Therefore, we take some inspiration from techniques used in link prediction and try to reuse these in this research.

3.4 Data

The data that will be used is provided by the French company eXenSa that focuses on big data analysis. The data contains basic meta-information of websites like URL, language, HTTP response status, size of the content and outgoing links. Furthermore, the contents of the web-pages are converted into semantic vectors and a content digest. The latter can be used to check whether or not the page has been modified. Lastly, of approximately 40% of the pages the Trustrank can be accessed. An example of a data-record is given in Appendix A.

The data set was gathered in a web crawl. One data set contains approximately 3 million pages. The total size of one (zipped) data set is approximately 6 GB.

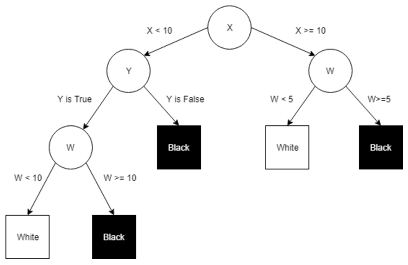


FIGURE 4: Decision Tree

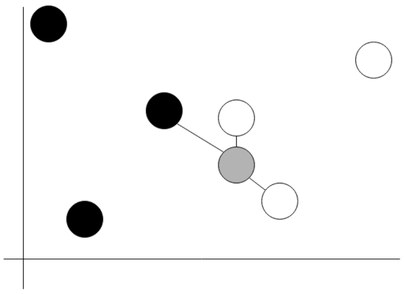


FIGURE 5: K Nearest Neighbors

3.5 Supervised Learning

Supervised learning is a machine learning task using historical labeled input and output variables. The algorithm tries to find relations between the input and output data. The goal is that the algorithm can predict what the output should be when new, unknown variables are presented. There are multiple, well-known supervised learning algorithms. In section 3.5.1 to 3.5.4 several are elaborated upon. Section 3.5.5 explains two normalisation methods and 3.5.6 explains how the performance of these algorithms can be compared.

3.5.1 Random Forest

The random forest algorithm is an extension to the decision tree classifier. A decision tree is a flowchart with several internal and leaf nodes. At each internal node, the algorithm must decide (based on one attribute of the input vector) which branch to take. Each leaf node corresponds to to a prediction class. See figure 4. Here, the circles represent the internal nodes and the squares the leaf nodes.

In the Random Forest algorithm, multiple different decision trees are generated based on random subsets of the training data. Each tree predicts to which class the input vector belongs. The class that gets the majority of 'votes' is chosen as the predicted class.

When using the Random Forest model, the number of trees should be chosen. Choosing this number too low, may lead to overfitting. However, using too many trees is computationally heavy and may not increase the performance.

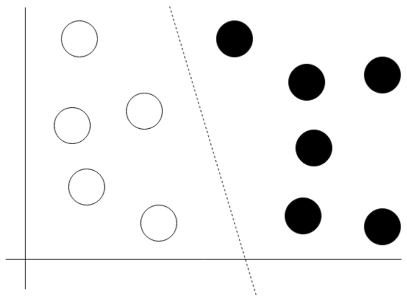


FIGURE 6: Support Vector Machine

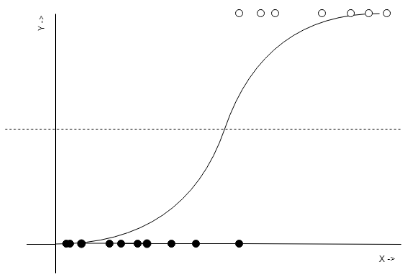


FIGURE 7: Logistic Regression

3.5.2 K Nearest Neighbors

In the K nearest neighbors algorithm, the output of a newly presented variable is based on the labels of the K closest data points. This algorithm for $K = 3$ is illustrated in figure 5. Suppose that we should predict the color of the gray circle. Since of its 3 closest neighbors, 2 are black, the dot will be classified as black. The Euclidian distance is often used to calculate the distance between two data points.

Obviously, an important feature of the K nearest neighbors model is the choice for K. Choosing a K that is too low may lead to overfitting. However, if we choose K too large, the prediction may be too general and heavily dependent on the label that occurs the most in the dataset. It is customary to choose an K that is odd, to make sure that ties are not possible.

3.5.3 Support Vector Machine

The support vector machine algorithm tries to find an optimal hyperplane that separates the classes. The hyperplane is considered optimal if the margin between the plane and the closest nodes of both classes is as large as possible. In figure 6 an illustration is given. Here, the striped line is the hyperplane (which in the 2D space is a line). Each new input on the right side is classified as black, on the left as white. Obviously, this figure pictures an ideal case. In real world examples, the hyperplane may not be linear or the data points may overlap.

3.5.4 Logistic Regression

Logistic regression is a statistical method that returns the probability that a certain input belongs to a class. It uses the following function (called Sigmoid function) to predict to which class an input vector belongs

$$f(t) = \frac{1}{1 + e^{-}(\beta_0 + \beta_1 x_1 + ...)} \tag{1}$$

Here, x_i denotes the i^{th} attribute of the input vector. The coefficient β_i is called a weight. Using the training data, the weight are adjusted such that the algorithm behaves optimally. See figure 7.

3.5.5 Normalisation

For the Machine Learning models K Nearest Neighbors and Support Vector Machine, it is important that the sizes of the domains of the features are similar. Both of these algorithms are based on calculating distances between nodes. Consequently, if the sizes of the domains differ significantly, the algorithms may be biased in favor of smaller domains (the distances are shorter in those cases). This leads to incorrect predictions.

To prevent incorrect predictions, the data needs to be normalised. There are several means of normalising data and in this paper we will use 2: normalising by maximum value and by rank. More precisely, consider the vector $\vec{v} = (v_1, v_2, ..., v_{n-1} v_n)$ such that $v_1 \leq v_2 \leq v_3 \leq ... \leq v_{n-1} \leq v_n$. Vector $\vec{u}$ is the normalised vector $\vec{v}$ by maximum value if

$$\vec{u} = \frac{1}{v_n} \cdot \vec{v} \tag{2}$$

Vector $\vec{w}$ is the normalised vector $\vec{v}$ by rank, if

$$\vec{w} = (\frac{g(v_1)}{n}, \frac{g(v_2)}{n}, ..., \frac{g(v_{n-1})}{n}, \frac{g(v_n)}{n}) \tag{3}$$

where

$$g(v_i) = \begin{cases} g(v_{i-1}) & v_i = v_{i-1} \\ i & otherwise \end{cases} \tag{4}$$

The main advantage of normalizing by maximum value, is that the relative distances between the nodes remain the same. On the other hand, the main advantage of normalizing by rank, is that the result values are spread across the entire domain, even if the original data is skewed.

3.5.6 Evaluation Metrics

The performance of the machine learning algorithms is measured using data with known labels. It is important that this test data is not used in the training part, since the results will then be biased. Once the model has evaluated the test data, the predicted labels can be compared to the actual labels in a confusion matrix (table 1). The correctly predicted values are called true positives (TP) or true negatives (TN), whereas the incorrectly values are named false positives (FP) and false negatives (FN). Using the confusion matrix, the following evaluation metrics can be calculated:

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP} \tag{5}$$

$$\text{Precision} = \frac{TP}{FP + TP} \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{8}$$

To summarize, accuracy measures the ratio of correct predictions to the total number of input samples. Precision is the number of samples correctly classified as positive, divided

Table 1: Confusion Matrix

	Predicted: 0	**Predicted: 1**
Actual: 0	True negatives (TN)	False positives (FP)
Actual: 1	False negatives (FN)	True positives (TP)

by all classified positives. This is an indication of how many selected items are relevant. Recall is the ratio of correctly predicted positives to all the input samples that should have been classified as positive. This is an indication of how many relevant items are selected. The F1-score is a metric calculated using precision and recall. It is an indication of how precise and robust the classifier is. Overall, the closer a metric is to 1, the better.

Lastly, models like Random Forest and Logistic Regression can give an indication which features are most important when predicting the labels.

4 Related work

4.1 Crawlers

Research on the effectiveness of crawlers has been done since the 1990s. Cho et al. [4] studies several importance metrics and ordering schemes in order to make sure that important pages are found earlier. They conclude that PageRank [11], a metric using the importance of pages that link to the concerned page, is an optimal ordering metric when searching for important pages.

In later research, it is concluded that websites use the PageRank algorithm to increase their position on search engines. For example, they place (hidden) links or text on their pages to get a higher rank. To combat these 'spam sites' a new metric called TrustRank was developed by GyÃűngyi et al. [6]. This method, based on PageRank, uses the links to and from known reputable sites in order to calculate the probability that a site is spam or not.

Chakrabarti et al.[1] elaborated upon focused crawling. Using this technique, a crawler does not visit all the pages stored in the queue. Instead, it only visits the pages that seem interesting based on the search query. Firstly, a classifier is trained based on seed URLs. Afterwards, this classifier is used in the crawl process by evaluating the visited pages. Only pages that are classified as relevant are further investigated. Using this method, they found that relevant pages were found more consistently.

Although the general findings in the findings above still hold, it must be considered that most research is conducted around the year 2000. As a consequence, some of the considerations made in the research do not hold anymore (e.g. the internet is nowadays larger than 1.5TB).

4.2 Evolving Web

Ntoulas et al. [10] conducted research on the evolution of 153 popular sites. Their findings indicate that new pages are created at the rate of 8% a week. However, a lot of these pages contain content that was available before. The rate of new pages is estimated at

5%. Furthermore, they claim that the link structure of the web is changing significantly. Every week, approximately 25% new links are created. After a year, 80% of the links are replaced. One last interesting finding, is that the frequency of change does not seem to be a good predictor of change in the future.

Cho et al. [3] defines the freshness and age of a local database. Moreover, they research how often a page should be crawled in order to keep the local information up-to-date. They argue that it is reasonable to assume that pages on the web are updated independently and according to a Poisson process. Surprisingly, they find that pages that change more often, should not be crawled more in order to keep the freshness of the local copy above a certain threshold. Instead, it is better to visit all pages at the same rate, regardless how often they are changed.

In a follow-up paper, the researchers address the freshness of an incremental crawler [2]. This type of crawler continuously updates its local database instead of updating it by periodically restarting the crawl process.

Radinsky et al. [12] investigate how to predict significant changes in content on the internet. One of their conclusions is that the accuracy of the prediction increases when the relations to other pages are taken into account.

4.3 Link Prediction

Since the rise of social media websites, more research has been devoted on Link Prediction. Several techniques have been developed, all with mixing results. Some of the methods are based on the assumption that related pages lie topologically close in the network. To predict future links, neighbours and (shortest) paths between the nodes are compared [9, 13].

Other research has been done by Zhu et al. [15] on using Markov chains to model user's navigation on the world wide web. A transition matrix is computed based on user data. This matrix is compressed by clustering nodes that have similar transition behavior. Using this transition matrix, one can predict to which page a user will probably go next.

Al Hasan et al. [7] incorporate supervised learning in their link prediction tools. They consider the problem as classifying positive classes (a new link) in the data set; a problem that can be solved by a binary classifier. Furthermore, different features are compared and ranked based on their ability to correctly predict links. 'Sum of neighbours' and 'keyword match count seem' to perform well.

In recent years, deep learning has been used in link prediction. For instance, a Link Prediction Framework called DeepLink has been developed by Keikha et al. [8]. DeepLink automatically selects the best features of a network that can be used in the link prediction. Moreover, DeepLink uses structural and content feature vectors. This framework performs better than often used methods for link prediction like Node2Vec and DeepWalk.

5 Notations

Before we state the methodology, experiments and results, we need to clarify what notations will be used in the rest of the paper.

- In total, 3 datasets will be used (see section 7.1). All the datasets will be given an index $i \in \{1, 2, 3\}$. The indices of the datasets are ordered based on the sequence in which they will be crawled.
- The i^{th} dataset will be denoted with $G_i(V_i, E_i)$. Here, V_i and E_i are the vertices (web pages) and directed edges (hyperlinks) of the i^{th} dataset respectively. Each vertex is uniquely identified by the url of the web page. For simplicity, we will assume that G_i is a simple directed graph, meaning that there are no loops and/or double edges.
- The neighborhood of a node v in the i^{th} dataset will be denoted with $\Gamma_i(v)$ ($\Gamma_i(v) \in V_i$). Notice that $v \notin \Gamma_i(v)$.
- The set of vertices in the i^{th} dataset that have edges pointing to v will be denoted with $\Gamma_i^-(v)$. More formally, $\Gamma_i^-(v) = \{u | u \in V_i, (u, v) \in E_i\}$. This is often called the in-neighborhood. The edges to v are called inlinks.
- The set of vertices that v points to in the i^{th} dataset are denoted with $\Gamma_i^+(v)$. To be precise, $\Gamma_i^+(v) = \{u | u \in V_i, (v, u) \in E_i\}$. This set is called the out-neighborhood. The edges are called the outlinks. Notice that $\Gamma_i^-(v) \cup \Gamma_i^+(v) = \Gamma_i(v)$.
- Since G_i is a simple directed graph, the number of inlinks is the same as the size of the in-neighborhood. We will often refer to $|\Gamma_i^-(v)|$ as the number of inlinks of v. The same holds for the outlinks and out-neighborhood.
- The subset $U_i(v) \subseteq V_i$ will contain all the vertices that are in the same domain as vertex v. Note that $v \in U_i(v)$ and that $|U_i(v)|$ is the number of pages in the domain of vertex v.
- Since it may be possible that not each page of a certain domain is crawled, we cannot determine the value of $|U_i(v)|$ precisely. Therefore, we estimate the value of $|U_i(v)|$ by a lower and upper bound. The lower bound $|U_i^-(v)|$ is based on the number of pages that is actually crawled. The upper bound $|U_i^+(v)|$ is based on the number of pages crawled and pages that are not crawled, but are referenced to (i.e. links to pages in that domain). By default, the upper bound is taken.

6 Estimation Rate of New Outlinks

To study the problem of finding new pages, we first need to formalise how the links are created over time. In this section, we elaborate on that issue. The basis of this estimation is closely related to the "Freshness" of a database, proposed by J. Cho et al. [3]

We assume that for each page $v \in V_i$ links to new pages are created according to a Poisson process. Such a process is regularly used to describe a sequence of events that happen at random and independently, with a fixed rate. If a sequence of events follow this Poisson process, the time to the next event is exponentially distributed. Specifically, if we let T be the time that the next event occurs in a Poisson process with rate λ, then the probability density function of T is as follows:

$$f_T(t) = \begin{cases} \lambda e^{-\lambda t} & t > 0 \\ 0 & t \leq 0 \end{cases} \tag{9}$$

Notice that each individual page has its own change rate λ, since one page may have new outlinks daily, whereas this only happens once a month for another page. By integrating (9), we obtain the probability that v has new outgoing links in the time interval $(0, t]$:

$$P(T \leq t) = \int_0^t \lambda e^{-\lambda t} dt = 1 - e^{-\lambda t} \tag{10}$$

As explained in section 3.5.4, Logistic Regression gives also the probability that v has new outgoing links in a certain time interval. By equating these probabilities, we can calculate an estimation for the change rate of a page based on its features:

$$\lambda = -1 \cdot \frac{\ln[1 - P(T \leq t)]}{t} \tag{11}$$

However, one should be very careful when using this estimation. Firstly, since logistic regression is a machine learning tool, it does not return the precise probability that the next event will happen. Secondly, it only indicates whether or not a new outlink will appear. It does not take into account *how many* outlinks may appear. This will skew the results.

7 Methodology

In this section, we elaborate upon the general approach to this research. We will explain what data we use, how it is preprocessed and what features are extracted. Lastly, we will explain the details of the experiments that are conducted.

7.1 Approach

To answer the main research questions (RQ 1), we have to find a function $f : V_3 \to \{0, 1\}$ that predicts whether or not a vertex $v \in V_3$ will have any new outgoing links. More specifically, if $f(v) = 0$, the function predicts that v will have no new outgoing links and when $f(v) = 1$ it will. We consider outlinks to both new web-pages and existing ones (see case 1 and 5 in Section 3.1).

To create this function, we acquire three datasets of roughly the same part of the World Wide Web crawled at different points in time. The data will be preprocessed and sanitized if needed (see section 7.2). Afterwards, we use G_1 and G_2 to extract topological features (section 7.3) and we will use G_3 to label these features based on whether new outlinks are present. The labeled features will be used to train four different machine learning algorithms. In the last step, the performance of the algorithms is tested and displayed.

7.2 Data and Preprocessing

In total, three datasets have been acquired. The crawl were started on the 5th, 14th and 19th of June 2019 respectively. All crawls are instantiated with the same initial seeds, to make sure that roughly the same part of the Internet was crawled.

Before the features can be extracted, the data needs preprocessing. Firstly, some web-pages returned HTTP-error codes upon crawling. Most of them were in the 400 (client error response) or 500 range (Server error response). These pages often contain no useful information, and therefore only the pages with a successful HTTP response (200 range) were considered. Secondly, some of the features can only be calculated if the page is crawled in all datasets. Therefore, pages that were deleted in-between G_1 and G_3 are removed as well.

7.3 Features

The following static features are calculated for any node $v_1 \in V_2$. Most of the features are based on the neighborhoods of the vertex itself. Below all the features are listed, together with an reasoning why these features may be useful.

1. **Number of inlinks of v_1**

   $$|\Gamma_2^-(v_1)| \tag{12}$$

   If v_1 has many inlinks, it may contain information regarded as important by the other pages. Hence, this page might be well maintained and regularly updated. Therefore, it might link to new pages in the future.

2. **Number of outlinks of v_1**

   $$|\Gamma_2^+(v_1)| \tag{13}$$

   If v_1 is pointing to a lot of other pages, it may point to even more pages in the future. This might not be the case for a page that points to no other pages.

3. **Number of inlinks in the same domain**

   $$|\Gamma_2^-(v_1) \cap U_2(v_1)| \tag{14}$$

   If a lot of pages in the domain point to the same page, this page contains probably important information in the domain. Therefore, this page might be well maintained and regularly updated. New pages might occur close to this page.
   Notice that

4. **Fraction number of inlinks in the same domain**

   $$\frac{|\Gamma_2^-(v_1) \cap U_2(v_1)|}{|U_2(v_1)|} \tag{15}$$

   Similar reasoning as feature 3, only now divided by the number of pages in the domain. This will take smaller websites better into account compared to larger websites.

5. **Number of outlinks in the same domain**

   $$|\Gamma_2^+(v_1) \cap U_2(v_1)| \tag{16}$$

   If v_1 points to many pages in the same domain, it might be an indication that once another page is created in that domain, v_1 will also point to it. An example can be a homepage of a website.

6. **Fraction number of outlinks in the same domain**

   $$\frac{|\Gamma_2^+(v_1) \cap U_2(v_1)|}{|U_2(v_1)|} \tag{17}$$

   Similar reasoning as feature 5, only now divided by the number of pages in the domain. This will take smaller sites better into account compared to larger sites.

7. **Lower Bound Pages in Domain**

$$|U_2^-(v_1)| \tag{18}$$

If the domain of the page is large, the page might be part of a website that is changing regularly. Hence, new outlinks may occur often.

8. **Upper Bound Pages in Domain**

$$|U_2^+(v_1)| \tag{19}$$

Similar reasoning as feature 7.

9. **Content of the page has changed**

$$g(v_1) = \begin{cases} 1 & \text{The content of } v_1 \text{ has changed between } G_1 \text{ and } G_2 \\ 0 & \text{The content of } v_1 \text{ remained the same between } G_1 \text{ and } G_2 \end{cases} \tag{20}$$

If the content of v_1 has changed, it is likely that the page is being maintained by someone. Therefore, it may be likely that the page will be changed in the future and thus have new outlinks.

10. **Number of new outlinks**

$$|\Gamma_2^+(v_1) \setminus \Gamma_1^+(v_1)| \tag{21}$$

If a page has a high number of new outgoing links, it may be the case that the number of outgoing links will keep increasing in the future.

11. **Number of new inlinks**

$$|\Gamma_2^-(v_1) \setminus \Gamma_1^-(v_1)| \tag{22}$$

If a page has many new inlinks, it might indicate that the web page is in a neighborhood of the World Wide Web that is changing rapidly. Hence, this feature might be an indicator that new pages will appear close to it.

12. **Fraction of new inlinks**

$$\frac{|\Gamma_2^-(v_1) \setminus \Gamma_1^-(v_1)|}{|\Gamma_2^-(v_1)|} \tag{23}$$

Similar reasoning as feature 11, only now divided by the total number of inlinks of v_1. This might give an advantage to nodes with less inlinks.

13. **Number of neighbors with changed content**

$$\sum_{v \in \Gamma_2(v_1)} g(v), g(v) = \begin{cases} 1 & \text{The content of } v \text{ has changed between } G_1 \text{ and } G_2 \\ 0 & \text{The content of } v \text{ remained the same between } G_1 \text{ and } G_2 \end{cases} \tag{24}$$

Similar reasoning as feature 11.

14. **Fraction of neighbors with changed content**

$$\frac{1}{|\Gamma_2(v_1)|}\sum_{v\in\Gamma_2(v_1)} g(v), g(v) = \begin{cases} 1 & \text{The content of } v \text{ has changed between } G_1 \text{ and } G_2 \\ 0 & \text{The content of } v \text{ remained the same between } G_1 \text{ and } G_2 \end{cases} \tag{25}$$

Similar reasoning to feature 13, only now divided by the number of neighbors that v_1 has. This way, smaller sites are also taken into account.

15. **TrustRank [6]**
TrustRank is calculated using a list of trusted hosts (sites that are not spam) containing 170.000 pages. The TrustRank (TR) is calculated using the following formula:

$$TR(v_1) = \sum_{v\in\Gamma_2^-(v_1)} 0,85\cdot TR(v)+0,15\cdot h(v), h(v) = \begin{cases} 1 & v \text{ is a trusted host} \\ 0 & v \text{ is no trusted host} \end{cases} \tag{26}$$

A web page with a high TrustRank is assumed to have importance on the web. Therefore, we may assume that the web page is well maintained and has a higher probability of pointing to new web pages.

The features of v_1 will be labeled based on whether v_1 has new outgoing links in G_3.

7.4 Experiments

The labeled features will be randomly split into training and test data. The training data will be used to train the following four machine learning algorithms: Random Forest, K Nearest Neighbors, Support Vector Machine, Logistic Regression. The training will be done using 10-fold cross validation. Using the test data, the performance of the models will be validated. Furthermore, we will check which features are most important when predicting new pages on the Internet.

8 Results

The first, second and third dataset contain 2.944.955, 2.964.550 and 2.964.439 URLs respectively. Of those URLs respectively 1.805.757, 1.774.199 and 1.729.548 return a HTTP status code of 200. Lastly, a total of 1.693.126 URLs are present in all 3 datasets.

In total 32,4% of the pages changed its content between dataset 1 and 2. Of those pages, 69% contained new outgoing links. Since the time between dataset 2 and 3 was shorter, only 25% of the pages was changed between those crawls, of which 70% contained new outlinks.

Before the data can be used, it needs preprocessing as described in section 7.2. After preprocessing, 1.693.126 pages remain. Of only 843.362 of these pages was the TrustRank known. Lastly, of these remaining pages, only 20,3% contained new outlinks. To make sure that the machine learning algorithms can not achieve high performance because of this inbalance, we randomly remove some of the sites that did not contain new outgoing links. In total 339.078 pages remain (see figure 8).

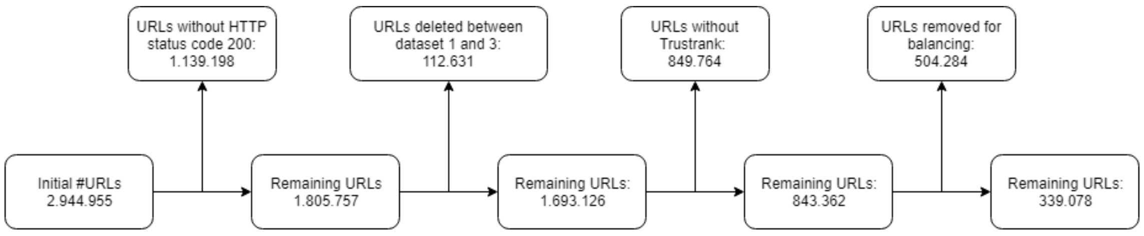

URLs without HTTP status code 200: 1.139.198
URLs deleted between dataset 1 and 3: 112.631
URLs without Trustrank: 849.764
URLs removed for balancing: 504.284
Initial #URLs 2.944.955
Remaining URLs 1.805.757
Remaining URLs: 1.693.126
Remaining URLs: 843.362
Remaining URLs: 339.078

FIGURE 8: Number of URLs during preprocessing

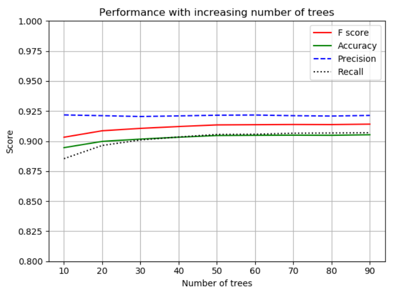


FIGURE 9: Performance with increasing number of trees

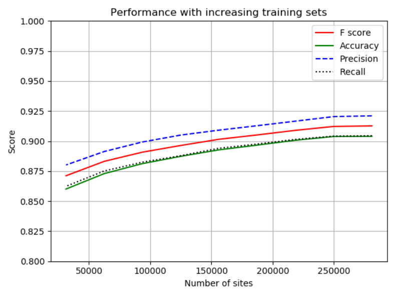


FIGURE 10: Performance with increasing training sets

8.1 Random Forest

Below, the results of the experiments using the Random Forest will be shown.

8.1.1 Number of trees

To figure out what the optimal number of trees is, we plot the performance of the algorithm against the number of trees used. Here, we use all features and 85% of the data will be used as training set (this ratio will later be explained). The results can be found in figure 9. As can be seen, the performance does not increase when more than 50 trees are used. Hence, this number will be used in the remainder of the section.

8.1.2 Training size

To test whether the algorithm keeps learning when data is added, we plot the performance against the size of the training set. The results are shown in figure 10. An increasing trend is clearly visible over the entire x-axis. This indicates that we should use a training set as large as possible (and we even could use more data). Therefore, 85% of the data will be used as training set and 15% as test set.

8.1.3 Performance and Vector Importance

When training the algorithm with all the features, we obtain an F1-score of 0,901. The importance of each feature is shown in figure 11. When we look at these results, it is clear that the most important features are the number of new outlinks in dataset 2 and the number of neighbors changed. Furthermore, almost all the other features seem to have the same level of importance. Striking, is that the (fraction of) new inlinks do not contribute at all.

To calculate the most optimal feature set, we systematically remove the least important

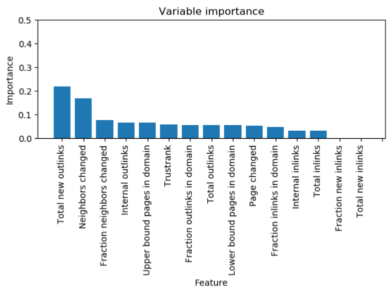


FIGURE 11: Variable importance Random Forest

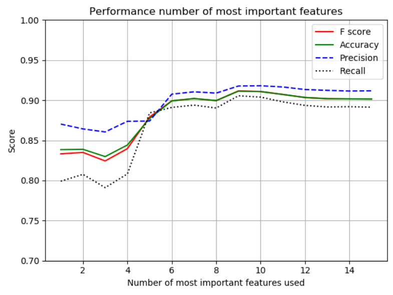


FIGURE 12: Performance Feature Sets Random Forest

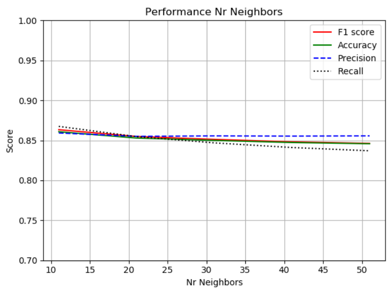


FIGURE 13: Performance Neighbors

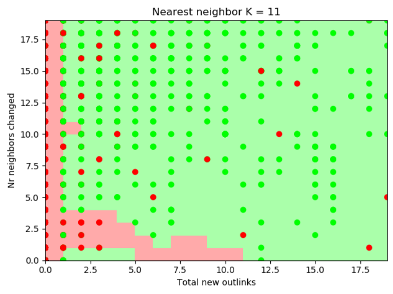


FIGURE 14: Neighborhood KNN

feature and compare the results (see figure 12). The performance is optimal when the top 9 features are used with an F1-score of 0,911.

8.2 K Nearest Neighbors

Below, the results of the experiments using the K Nearest Neighbors (KNN) algorithm are shown.

8.2.1 Number of Neighbors

As explained in section 3.5.2, choosing the number of neighbors that are used in the prediction is important. Figure 13 shows how the performance of the algorithm progresses when the number of neighbors is increased. A slightly decreasing trend is visible, meaning that with a low number of neighbors, the results are optimal. In figure 13, the data is normalised using the rank method. However, normalising using maximum value shows a similar trend.

8.2.2 Normalisation

Using $K = 11$, we can compare the performance of both normalisation methods discussed in section 3.5.5. Table 2 shows the performance of both methods. The normalising methods seem to have a similar performance and both outperform not using a normalising strategy. The optimal F1-score (0,863) is achieved using rank normalisation.

8.2.3 Neighborhood

It is interesting to visualise the working of the KNN network in the 2 dimensional space. In order to do this, we predict using only the two (most important) features 'Total new

Table 2: Performance Normalisation Methods KNN

Normalisation Method	**F1-score**	**Precision**	**Accuracy**	**Recall**
Maximum Value	0,847	0,830	0,842	0,864
Rank	0,863	0,859	0,861	0,867
No Normalisation	0,836	0,817	0,831	0,856

outlinks' and 'Number of neighbors changed' and use a K of 11. Since the sizes of the domains of these features are roughly the same, the data is not normalised. Figure 14 visualises how the KNN algorithm would classify the points. Data points in the green area would be classified as having new outgoing links and the contrary for points in the red area. The dots represent training data, where again the color represents whether or not a page contained new outgoing links.

Since only two features are used, this is only a simplified visualisation of the algorithm. However, there is a clear pattern to see in the figure. For low values of both features, the algorithm predicts that there are no new outgoing links.

8.3 Support Vector Machine

This section will elaborate upon the results of the support vector machine experiment. Since the training phase of the SVM is computationally heavy, all the experiments are conducted using just 20% of the test and training data.

8.3.1 Normalisation

In table 3 the performance of the different normalisation methods are displayed. Normalising by Rank seems to be the most optimal solution. Again, no normalisation gives the worst performance. The optimal achieved F1-score is 0,830.

Table 3: Performance Normalisation Methods SVC

Normalisation Method	**F1-score**	**Precision**	**Accuracy**	**Recall**
Maximum Value	0,793	0,764	0,784	0,826
Rank	0,830	0,871	0,836	0,793
No Normalisation	0,736	0,770	0,746	0,705

8.4 Logistic Regression

In this section, we discuss the results of the experiments using Logistic Regression.

8.4.1 Vector Importance

As explained in section 3.5.4, Logistic Regression is built upon the Sigmoid function. Comparing the weights in this function, gives us an idea of the importance of the features. The results are shown in figure 15. The two most important features seem to be the number of neighbors changed and the total number of new outlinks between dataset 1 and 2.

My removing less important features one at the time, we get an idea what the optimal feature set would be (see figure 16). Surprisingly, the performance seems almost constant for

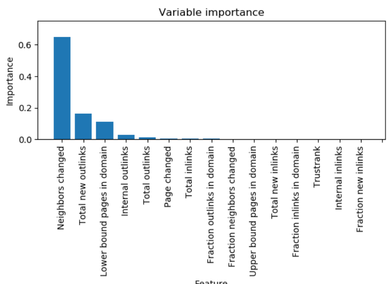


FIGURE 15: Variable Importance Logistic Regression

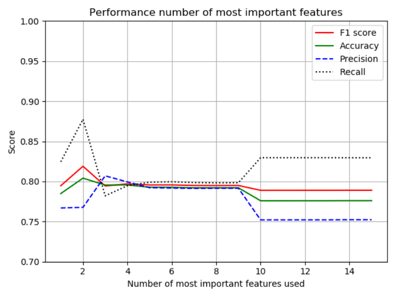


FIGURE 16: Performance Feature Sets Logistic Regression

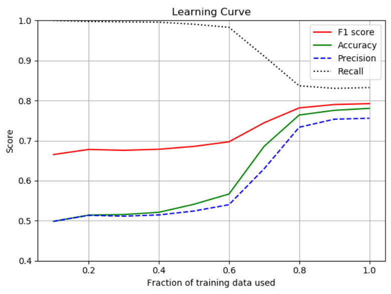


FIGURE 17: Learning Curve Logistic Regression

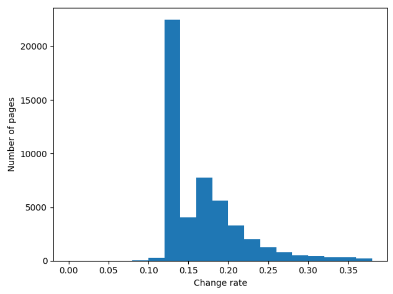


FIGURE 18: Change Rate Logistic Regression

any number of features used. Only if a few or almost all features are used, the performance changes.

8.4.2 Learning Curve

The learning curve of the Logistic Regression model is displayed in figure 17. An upward trend is clearly visible, an indication that more data is appreciated. For small training sets, the recall value is very high. This is since the logistic regression model classifies almost any input vector as having new outgoing links. This results in a high recall, but a very low accuracy and precision.

8.4.3 Change rate

Using the method explained in section 6, we can predict the change rate of the pages using logistic regression (see figure 18). Pages with a rate lower than 0.14 are classified as not having new outgoing links.

9 Discussion

Before we can interpret the results of this research effectively, we should consider the decisions that are taken to siplify the problem. Firstly, in preprocessing the data, approximately 90% of the URLs were discarded. Most of these URLs are discarded since the data was not complete or because of balancing. Therefore, we should keep in mind that by throwing away these URLs, data is lost and the training set may not represent the World Wide Web correctly.

Secondly, the times at which the crawles were started, were not evenly spread. Between dataset 1 and 2 was a 9 day gap, between dataset 2 and 3 a 5 day gap. If the rate at which pages are created is consistent, this would mean that less pages were created between dataset 2 and 3 compared to dataset 1 and 2. This could have skewed the results.

Thirdly, we are dependent on the pages that are present in the crawl. By initializing each crawl with the same seeds, we make sure that roughly the same part of the World Wide Web is crawled each time. However, we cannot make sure that this is actually the case. Pages that were crawled before might be left out in later crawls and vice versa. Changes in the web can only be detected correctly if the same part of the web is crawled each time. Therefore, any deviations will have a negative effect on the results.

Lastly, we only consider new outgoing links. It may happen that a new outgoing link is pointing to a page that existed already. Hence, new outgoing links is not a perfect metric for determining where new pages are created.

Keeping this in mind, we do seem to get good results from the machine learning models. Especially the Random Forest excels with an accuracy and precision of over 90%. The most important features, seem to be the total new outgoing links between dataset 1 and 2, and the number of neighbors changed. The former is not surprising; if a page linked to newly created pages in the past, it will likely do so in the future as well (think about webshops or news sites). However, it is surprising that the number of changed neighbors has a high influence on the classifiers. Apparently, if a page is in a part of the Internet that is changing rapidly, the page itself will change as well. Furthermore, the number of (new) ingoing links have almost no importance. This could be an indication that often used metrics like PageRank (which are heavily dependent on inlinks) are also a bad indicator when predicting new outgoing links. Interestingly, the learning curves of the algorithms are all increasing. This means that the performance of the models may be increased by adding new data.

10 Conclusion

We have shown that topological features are a good indicator when predicting where new pages on the World Wide Web will appear. With an accuracy and precision of over 90%, the Random Forest model was able to predict where new outgoing links would occur on the Internet. Other Machine Learning algorithms, like Logistic Regression, Support Vector Machine and K Nearest Neighbors, achieved an accuracy and precision of 75% and higher. The most important indicators when predicting outgoing links, are how often a page had new outgoing links before and the number of its neighbors that were changed.

The solution proposed in this research shows promising results in solving the problem of finding new pages. However, there are also indicators that the results may improve when more data is used in the training process of the models.

11 Future work

Firstly, this research should be repeated using more (clean) data. As the learning curves of the different algorithms showed, more data is appreciated and could improve the performance.

Secondly, it would be interesting to implement a classifying function into a focused crawler. This focused crawler would give higher priority to pages that are likely to contain new outgoing links. It should be verified if this crawler indeed finds more new pages than a traditional crawler.

Lastly, in this research we did not include the quality of the outgoing links. For instance, a webshop creates dozens of new pages daily. However, these pages are very similar and will not all contain new relevant information. It would be interesting to incorporate the quality of the new web page, to make sure we only find interesting content.

References

[1] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623 – 1640, 1999.

[2] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. 2000.

[3] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(2):117–128, 2000.

[4] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. *Computer Networks*, 30(1-7):161–172, 1998.

[5] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C Lee Giles, Marco Gori, et al. Focused crawling using context graphs. In *VLDB*, pages 527–534, 2000.

[6] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 576–587. VLDB Endowment, 2004.

[7] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006.

[8] Mohammad Mehdi Keikha, Maseud Rahgozar, and Masoud Asadpour. Deeplink: A novel link prediction framework based on deep learning. *CoRR*, abs/1807.10494, 2018.

[9] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[10] 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 conference on World Wide Web*, pages 1–12. ACM, 2004.

[11] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

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

[13] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences*, 58(1):1–38, Jan 2015.

[14] Meiyappan Yuvarani, A Kannan, et al. Lscrawler: a framework for an enhanced focused web crawler based on link semantics. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 794–800. IEEE, 2006.

[15] Jianhan Zhu, Jun Hong, and John G. Hughes. Using markov chains for link prediction in adaptive web sites. In David Bustard, Weiru Liu, and Roy Sterritt, editors, *Soft-Ware 2002: Computing in an Imperfect World*, 2002.

A Data record

Below, a record of the dataset is displayed in JSON-format. Note that several keys are truncated for readability purposes.

```
{
   "url":"http://url.org/",
   "urlViewInfo":{
      "lastView":"2019-06-18 21:43",
      "lastSchedule":"2019-06-18 20:33",
      "lastFetch":"2019-06-18 20:33",
      "rank":{
         "elementValue":0.016857073,
         "trustValue":0.02436128
      },
      "numInLinksExt":"2",
      "numInLinksInt":"112"
   },
   "urlFetchInfo":{
      "language":"English",
      "textQuality":1.3409368,
      "contentLength":25679,
      "httpStatus":200,
      "title":"title",
      "fetchDuration":161,
      "fetchDate":"2019-06-18",
      "internalLinks":[
         {
            "targetUrl":"https://www.url_inside_domain.com",
            "linkInfo":{
               "linkType":"LINK",
               "linkRel":1,
               "linkQuality":0.9921029
            },
            "crawlerInfo":{

            }
         }, ...
      ],
      "externalLinks":[
         {
            "targetUrl":"https://www.url_outside_domain.com",
            "linkInfo":{
               "linkType":"A",
               "text":"Link",
               "linkQuality":0.9308616
            },
            "crawlerInfo":{

            }
            }, ...
        ],
        "contentDigest":"OHQGbGtSKcaLmEWzNybstg==",
        "semanticVector":{
            "values":[ ... ]
        },
        "textSize":106
    }
}
```