TÉCNICO
LISBOA

2Gather4Health: Web Crawling and Indexing System Implementation

João Nuno Martins de Almeida

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. João Paulo Baptista de Carvalho
Prof. Fernando Manuel Marques Batista

Examination Committee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Advisor: Prof. João Paulo Baptista de Carvalho
Members of the Committee: Prof. Ricardo Daniel Santos Faro Marques Ribeiro

October 2018

Acknowledgments

I would like to thank my supervisors for helping me with my dissertation.

I would like to thank my friends, my family and specially my girlfriend, Isabel Antunes, for all the emotional support, good times, and for helping me stay positive throughout my university life. I could not have accomplished this stage of my life without them.

I would also like to thank my cousin, João Pedro Graça, for always being available to answer my questions about this study programme and for all the study resources he gave me.

Lastly, but the biggest thanks of all, goes to my parents, Ricardo de Almeida and Elisabete Martins, for all the support they gave me throughout my entire educational path. For always supporting my choices, both in education and in my personal life. For always being available to talk with me about my problems. For always being patient with me, specially when I get easily annoyed for being under stress. And for giving me the opportunity of having a great student life, one that I can be proud of. Thank you.

Abstract

In healthcare, patients tend to be more aware of their needs than market producers. So, it is only normal to start seeing innovative behavior emerging from them, or from their caregivers, to help them cope with their health conditions, before producers.

Today, it is believed that these users share their innovative health-related solutions, also called patient-driven solutions, on the Internet. However, the size of the Internet makes it hard to efficiently manually browse for these solutions.

A focused crawler is a system that automatically browses the Web, focusing its search on a topic of interest. This Master thesis proposes a focused crawler that searches the Web for patient-driven solutions, storing and indexing them to ease a further medical screening. To perform the focusing task, it was developed a distiller and a classifier. The distiller ranks the URLs to visit, sorting them by a given score. This is a linear combination of three components, that depend on the content, URL context, or scores of the pages where the URL was found. The classifier automatically classifies visited webpages, verifying if they concern patient-driven solutions.

In this thesis, it is shown that the developed system outperforms a breadth-first crawling and common focused approaches on measures of harvest rate and target recall, while searching for patient-driven solutions. The proposed classifier's results on crawled data deviate from its validation results. However, it is proposed an approach to re-train the classifier with crawled data that improves its performance.

Keywords: focused crawler; Patient Innovation; web indexer; text classification

Resumo

Na área da saúde, os doentes tendem a ter mais atenção às suas necessidades do que os produtores de mercado. Então, é normal começar a ver um comportamento inovador vindo destes, ou dos seus cuidadores, que os ajude a lidar com as suas condições de saúde, antes dos produtores.

Hoje, acredita-se que estes utilizadores partilham as suas soluções inovadoras relacionadas com saúde, também chamadas soluções orientadas ao doente, na *Internet*. No entanto, o tamanho da *Internet* faz com que seja difícil e ineficiente procurar por estas soluções manualmente.

Um *crawler* focado é um sistema que navega pela *Web* automaticamente, focando a sua procura num tema de interesse. Esta tese de Mestrado propõe um *crawler* focado que procura por soluções inovadoras orientadas ao doente na *Web*, guardando e indexando-as para facilitar uma futura tiragem médica. Foram feitos um destilador e um classificador para desempenhar a tarefa de foco. O destilador ordena os *URLs* a visitar, segundo uma dada pontuação. Esta é uma combinação linear de três componentes, que dependem do conteúdo, do contexto do *URL* ou das pontuações das páginas onde o *URL* foi encontrado. O classificador classifica automaticamente as páginas *web* visitadas, verificando se se referem a soluções orientadas ao doente.

Através da comparação de medidas de *harvest rate* e *target recall*, mostra-se que o sistema desenvolvido supera um *crawl* de busca em largura e abordagens focadas comuns, quando se procura por soluções orientadas ao doente. Os resultados do classificador proposto em dados extraídos de um *crawl* desviam-se dos resultados durante a sua validação. No entanto, foi proposta uma abordagem para retreinar o classificador com dados provenientes de um *crawl* e os resultados mostram uma melhoria no desepenho.

Keywords: *crawler* focado; *Patient Innovation*; indexador de *web*; classificação de texto

Contents

List of Tables

List of Figures

Acronyms

2G4H 2Gather4Health.

DIY Do It Yourself.

DNS Domain Name System.

DOM Document Object Model.

DoS Denial of Service.

FFP Fuzzy Fingerprints.

FFP-C Fuzzy Fingerprint Classifier.

FN False Negative.

FP False Positive.

FTP File Transfer Protocol.

GUI Graphical User Interface.

HITS Hyperlink Induced Topic Search.

HMM Hidden Markov Model.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

ICF Inverse Class Frequency.

IDF Inverse Document Frequency.

ILSP Institute for Language and Speech Processing.

ILSP-FC Institute for Language and Speech Processing Focused Crawler.

IST Instituto Superior Técnico.

kNN k-Nearest Neighbors.

LR Logistic Regression.

LUGNET	Lego User Group Network.
MNB	Multinomial Näive Bayes.
NB	Näive Bayes.
NLP	Natural Language Processing.
NN	Neural Network.
PoS	Parts of Speech.
RDF	Resource Description Framework.
SVM	Support Vector Machine.
TF	Term Frequency.
TN	True Negative.
TP	True Positive.
UI	User Interface.
URL	Uniform Resource Locator.
XML	Extensible Markup Language.

Chapter 1

Introduction

1.1 User Innovation

Formerly, users just influenced the market reactively, by giving their feedback on market's products and services, allowing producers to adjust them according to the users' needs. But now, they have a more active behavior. Their minds are set to take the initiative and contribute with new products or product modifications themselves. They tend to be more aware of their needs than producers, therefore, they are prone to innovate. They innovate expecting to improve the benefit they acquire from a certain product or service. This type of innovation is classified as *user innovation*, a phenomenon studied by several socio-economic researchers.

In [6], Gault defines user innovators as users who change a good or a service to enhance the benefits it provides. In [7], the authors further divide these defined innovators into two classes, defining *user innovator* as a single firm or individual that creates an innovation in order to use it, and *producer innovator* as a person or firm that innovates for profit, getting no direct use-value from their innovations. In the scope of this dissertation, the latter definition of user innovator will be used. To insert the definition in the actual problem, it is considered that a patient who creates an innovation for its own use is a user innovator. The same happens if a caregiver, i.e. someone closely related to the patient, is the creator. Still, someone who creates an innovation for profit is not considered as user innovator.

The growth of user innovation causes a huge impact on the market, shifting the job of innovating to users and collaborations between users and producers. The motivation behind user innovation is pushing users to innovate more and more, outperforming market innovation processes. To give an example, around $19 - 36\%$ of the users of industrial products and $10 - 38\%$ of users of consumer products have been found to modify products [8]. Additionally, their inventions or modifications can show improvements on efficiency, usability, price and other market metrics over their parallel retail products [9], [10].

However, user innovations are not only valuable for the market expansion. People need pragmatic solutions for their problems, solutions that are not being covered on the market, either because they do not exist or the alternatives are too expensive. This drives users to invent the solutions themselves, to fulfill the needs the market is not addressing.

This behavior can be seen all across the market sectors. Users are reinventing sports trainings and products [11], [12], finding new ways to improve energy production [9], [1], and inventing new devices and ways to help them cope with a disease or condition [13], [14]. The latter behavior is the a main focus of this dissertation.

These factors emphasize the importance of information collection about user innovation. However, little is being done to find out pragmatic ways of identifying user innovation and propagate user innovative solutions.

1.1.1 Identifying user innovation

Identifying user innovation is a challenging task. First, there is the challenge of identifying good sources of information. User innovations tend to pass unnoticed on public media because they stay on the shadow of company innovations, those made by known companies that attract the media attention. Then, one has to verify if the found innovations satisfy the definition of user innovation. To illustrate this problem, Figure 1.1 depicts a flowchart explaining the criteria for identifying user innovations used in [1]. The portrayed process complies with this dissertation's definition of user innovation. Afterwards, it is common practice to have experts on the subject performing a final in-depth analysis to the innovations retrieved [1].

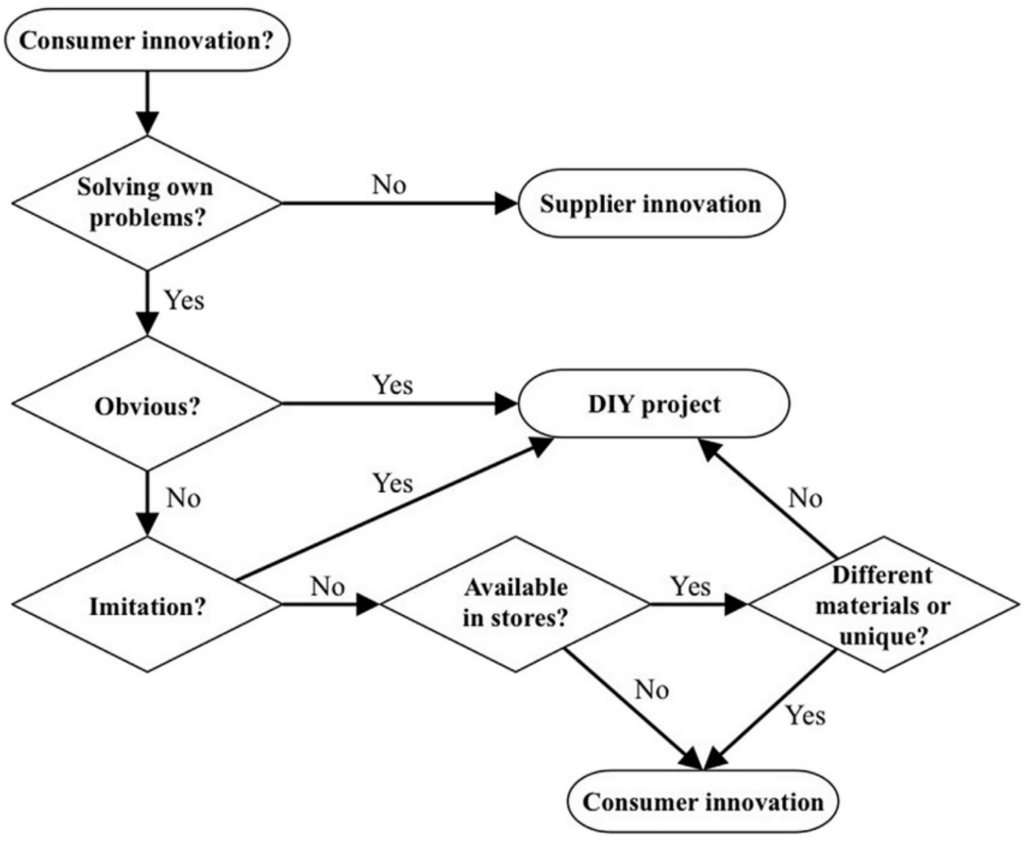

Figure 1.1: Criteria for consumer (user) innovation identification, extracted from [1]

Some examples of this process can be seen in [9] and in [1]. In the first, Hyysalo et al. analyzed Finish heat-pump and wood pellet Internet forums, searching for certain keywords and Do It Yourself (DIY) sections, to identify user inventions. They interviewed some of the its creators to gather more information on their inventions, so they could be further filtered on an assessing process made by experts on the subject. While in [1], it was used a combination of network search lead-user identification methods, explained on [15], to identify user innovations. They found out 300 potential consumer inventions or modifications that obeyed their definition of user innovation, which were then reduced to 213 by external experts.

1.1.2 User innovation diffusion

The propagation of user innovative ideas is also an important topic on the matter. It was previously said that a large percentage of users modify or invent new products, most of them enhancing in some aspects its retail version. Therefore, it exists the need of studying current ways and finding out new ones for user innovation dissemination.

For example, in [9], the experts assessed all identified user innovations on their potential for diffusion. On the research made in [1], it was analyzed how the diffusion of the selected innovations happened, and it was constructed a predictor for whether or not a given innovation should have diffused.

The efforts being made show that this is a major concern on this topic, and that there is a large demand for dedicated platforms for user innovation diffusion.

1.2 Motivation

As mentioned, users are more aware of their needs. In healthcare, the word ”need” is not an understatement. Some users (in this case patients) have been living with the same health condition for years, sometimes even their whole life. These individuals have to cope with daily life problems, imposed by their health condition, for which medicine does not provide a solution. This forces them to be constantly thinking of new ways to make their lives better, to approach new methods, to take some risks if that opens a whole new better life for them. Therefore, it is not a surprise that this area has a great potential for user innovation. Users are very concerned with their own well-being, more concerned than any producer in the healthcare industry. It is normal that they start to think of solutions, to help them or help others close to them coping with their health conditions, before professional innovators do. In fact, current literature [16] highlights the innovation capacity of patients and informal caregivers to develop solutions for health condition derived problems, which were not being addressed on the market.

Additionally, Internet, in its current state, is the largest source of knowledge in the world. This makes it easily accessible by anyone, which means people may learn about anything, anywhere. Furthermore, Internet eases the creation of global social communities. Users engage in this groups to share and acquire knowledge at the distance of a click. This environment drives user innovation appearance, making it publicly available for anyone to witness.

The user innovation being observed in the healthcare area and the noticeable presence of the Internet in today’s society prompts investigators to study the intersection of the two. For this purpose, Oliveira and Canhão created the Patient Innovation platform, an online open platform with a community of over 60.000 users and more than 800 health related innovative solutions developed by patients and informal caregivers from all over the world, shared and medically screened from 2014 to 2018. These solutions were found by manually browsing the Web, through search engines like Google and DuckDuckGo, searching for a combination of appropriate keywords.

However, the current searching method is not effective nor efficient, considering the amount of information currently available on the Web. Although the amount of patient-driven solutions is expected to be vast, they are hard to find. They may be posted on low-profile webpages, e.g., a personal blog, a personal profile of a social network, a local journal. These are hard to spot by manually searching the web, as they will hardly be at the top results of any search engine, even if they contain all the keywords that are being searched for. Furthermore, keywords alone do not cover the specificity of patient-driven solutions. A more focused model is needed.

Consequently, there is the need for a system that automates that Internet search, retrieving solutions in a much more effective and efficient manner. A focused web crawler is the ideal system for that task. It has the ability to quickly run through several webpages, avoiding pages concerning non-specified topics, while classifying the parsed pages' relevancy to the specified topic. This way, patient-driven solutions can be searched for in a much faster way than through manual search, leaving the human work only for medical screening. Furthermore, the crawler has the ability to reach subtopics in this area never addressed by the current analysts, as it can do a much more comprehensive search. A web indexer can index all the relevant information found, making it easily searchable, by relevancy score, topic, keywords, title and other metadata. This system can greatly increase the publication rate and diversity of health-related solutions in online user innovation diffusion platforms, like the Patient Innovation platform.

Although, there is already some research on how to identify and study the user innovation phenomenon over the Internet in some areas [11] [1] [9], an automatic system that crawls the Web searching for this type of information has not been done before.

1.3 Proposed Solution

This work implements a focused crawler, capable of efficiently searching for patient-driven solutions. The system is set to start from a set of hand picked Uniform Resource Locators (URLs), called seed URLs, which have the potential to be linked to other URLs about the target topic. To classify and set the fetching order of webpages, the crawler implements a *classifier* and a *distiller*, respectively. The classifier is an implementation of a 2-layered text classifier, trained with webpage samples of the DMOZ repository and with samples from the Patient-Innovation platform. It identifies parsed webpages concerning patient-driven solutions. The distiller sets the fetching order of newly discovered URLs, favoring those believed to relate to the solutions being searched for.

The crawler outputs relevant results to a web indexer, which organizes the information making it possible to be searched for. It is possible to query the index on fields of title, content, URL and others, and the query results can be sorted by their similarity score with the queried phrase and by their relevancy score to patient-driven solutions. This facilitates the medical screening of the solutions found, as the results obtained from the system are meant to be further analyzed by experts.

1.4 Objectives and Results

The main goal of this work is to automate the patient-driven solutions search, while doing an automatic maintenance of the webpages where they reside, in order to make the web search more efficient and comprehensive, keeping the information up to date. Another goal is to provide a system that efficiently indexes all the relevant information found, presenting also an interface to interact with the provided index. The final purpose of the proposed system (composed of the web crawler and the web indexer) is to collect webpages containing patient-driven solutions, meant to be further refined and processed by experts, so they can be published in an online user innovation diffusion platform, like the Patient Innovation platform.

The crawler's text classifier was tested on a crawled set of webpages, to evaluate its performance on crawled data. The proposed crawler was compared to a broad crawler and common focused crawling approaches on metrics of harvest rate and target recall.

The results show that the proposed approach outperforms the other comparing methods, being more efficient while searching for patient-driven solutions. Also, they show that incrementally training the custom classifier with crawled data can improve the solution search precision.

1.5 Achievements

As a result of this thesis, a paper has been accepted and presented at the 10th European Symposium on Computational Intelligence and Mathematics (ESCIM 2018) and an invitation has been received to extend the paper for publication as a book chapter in a Springer lecture notes volume.

1.6 Outline

Chapter 2 presents the related work required to understand the development and evaluation of this thesis' system. Chapter 3 describes the dataset used in this system development, as well as its cleaning process. Chapter 4 demonstrates the system architecture, describes its components and interactions and explains how each custom component was implemented. Chapter 5 describes the experiments done to evaluate the system's performance. It also shows the results obtained followed by observations and drawn conclusions. Finally, Chapter 6 concludes this Master thesis with a summary, final thoughts and future work.

Chapter 2

Related Work

This chapter describes the fundamental concepts that the reader needs to know in order to understand this Master thesis.

First, it gives an overview of the Text Classification concept, additionally describing some classification models used in this work. Furthermore, it describes the evaluation framework usually followed on this type of classification problems. The Web Crawling section describes the architecture and common processes undertaken in a crawler program, it gives a review of the types of crawlers and focused crawler guiding components that exist, describes the performance metrics used to evaluate a focused crawler, describes the common problems and limitations in a Web crawling process and analyzes current State-of-the-art open-source crawlers, stating their advantages and disadvantages with respect to the goal of this thesis. The Web Indexing section explains the web indexing process, highlighting the most common technique used to build web indexes and giving the example of Solr, the open-source web indexer and search platform used on this thesis development. Finally, the last section addresses previously done work concerning automatic crawling and classification of the health and user innovation subjects.

2.1 Text Classification

With nowadays' digital documents availability and mass production, automatic text classification is a crucial procedure for categorizing documents. Normally, this categorization is either topic-based (sports, science, arts, politics, etc) or genre-based (scientific articles, news reports, movie reviews, advertisements, etc) [17].

Text classification methods use Natural Language Processing (NLP) as a tool for text processing. NLP is the science that studies text analysis and processing. Documents are sets of unstructured data (text). These can be chaotic, presented in all shapes and sizes, so usually it is needed some pre-processing before documents are fed to classifiers for training and classification. It is through NLP methods that that processing is done.

Common text pre-processing steps are:

- Normalization - the text is lower cased, textual accents are removed, and numbers, dates, acronyms and abbreviations are written in a canonical form or removed.
- Tokenization - single or multiple-word terms are separated, to form tokens.
- Stopword removal - common words, like ”to”, ”and”, ”or” in the English language, are removed.
- Lemmatization - words are transformed into their lemma.

- Stemming - words are transformed into their stem.

Between stemming and lemmatization only one process is chosen. These are two different forms of mapping different words to the same term, when they are derived or inflected from that term.

The most common approaches for text classification tasks include [17]: Näive Bayes (NB) variants, k-Nearest Neighbors (kNN), Logistic Regression or Maximum Entropy, Decision Trees, Neural Network (NN), and Support Vector Machine (SVM). These usually use stylometric and syntactic features, representing each document or class as an unordered set of terms. To each term is associated a presence or a frequency value, and no semantic meaning is kept. This is called a bag-of-words model.

The most common approaches use variations and combinations of Term Frequency (TF) and Inverse Document Frequency (IDF) transformations as feature values, showed on equations 2.1 and 2.2 in their most simplistic forms.

These approaches demonstrate great results and they are the most used ones [18] [19]. However, it is a very restrictive technique for document representation as it fully removes the semantic relations within the document. Thus, the research community realized that using semantic features for text classification can achieve better performances. Hence, text classification is gradually evolving from syntax-based to semantically-driven approaches [18].

$$tf_i = n_i, \qquad \text{where } n_i \text{ is the number of occurrences of term } i \tag{2.1}$$

$$idf_i = \frac{N}{N_i}, \qquad \text{where } N \text{ is the total number of documents and } N_i \text{ is the number of documents where feature } i \text{ appears} \tag{2.2}$$

In text classification problems, using terms as features can result in a very large feature space. One approach that reduces the feature space uses pre-trained *Word2Vec* [20] models to transform each word into a vector of features. Each word vector representation is obtained by training a Neural Network model using a very large dataset. This model obtains each word vector representation by analysis of each word's neighbor words. This representation holds some semantic value, as words with similar meanings will have similar vector representations, because they will have similar neighbor words. This presents some additional value over the TF-IDF approach, that just uses syntactic knowledge. All the word vectors of a document can then be averaged to make a document vector representation. This approach often reduces the feature space and it has proven to perform better than a regular TF-IDF approach when combined with it [21].

One alternative classification approach is the Fuzzy Fingerprint Classifier (FFP-C), which was first used in [22] for authorship identification. This method consists of applying a fuzzy function to the top most frequent terms in each class, which is called the class fingerprint. The same is done for each document to be classified, and its fingerprint is then compared to the classes' fingerprints to see to what class the document belongs to.

In the following Sections, it will be briefly described the Multinomial Näive Bayes (MNB),SVM and Logistic Regression (LR) methods. Also, a more detailed description of the FFP-C method will be given. These are important approaches used in this thesis.

2.1.1 Multinomial Naive Bayes

Naive Bayes classification is a simple, yet powerful, approach for classification problems. It is a fast algorithm and it has demonstrated to be an effective approach for text classification problems [23].

A Naive Bayes classifier is a probabilistic classifier that uses the Bayes theorem as its main tool for class prediction. The main idea behind it is to find the most probable class to which a sample belongs, given the sample. Using the Bayes rule, this can be mathematically translated to

$$\underset{0\leq i<N}{\arg\max} P(C_i|x) = \underset{0\leq i<N}{\arg\max} \frac{P(C_i)P(x|C_i)}{P(x)} = \underset{0\leq i<N}{\arg\max} P(C_i)P(x|C_i) = \underset{0\leq i<N}{\arg\max} P(C_i, x) \tag{2.3}$$

Where C_i is the class i, on a total of N classes, and x is the sample being classified. x is represented by the feature vector $x = (x_0, x_1, \ldots, x_{n-1})$, where n is the number of features.The denominator $P(x)$ can be dropped on equation 2.3, because it is equal for every class.

Using the definition of conditional probability one has

$$\underset{0\leq i<N}{\arg\max} P(C_i)P(x|C_i) = \underset{0\leq i<N}{\arg\max} P(C_i, x) \tag{2.4}$$

This naive approach assumes that each feature is conditional independent from every other feature. Using the chain rule, this leads to

$$\underset{0\leq i<N}{\arg\max} P(C_i, x) = \underset{0\leq i<N}{\arg\max} P(C_i) \prod_{k}^{n} P(x_k|C_i) \tag{2.5}$$

In Multinomial Naive Bayes classification, each feature value of each sample is considered to be drawn from a multinomial distribution. This means that $P(x_k|C_i)$ follows a multinomial distribution. In text classification, this events can be represented by word occurrences in a document. In [24], McCallum and Nigam give an in depth explanation of how to use the Multinomial Naive Bayes model for text classification.

2.1.2 Support Vector Machine

SVMs are one of the most used machine learning approaches for text classification. A current study shows that the use of SVMs for this task is the most popular approach [19] between a large number of collected researches and, in some contexts, it outperforms other text classification methods [25]. Even if it is not the choice for a system classifier, SVMs are very often used to compare performances with other State-of-the-art approaches.

A SVM is a discriminative classifier formally defined by a separating hyperplane. Given labeled training data, the algorithm produces a hyperplane which gives an optimal division of the training data, with the goal of optimal categorization of new samples. This hyperplane is obtained by an optimization problem which tries to maximize the classification margins (represented by dashed lines in Figure 2.1), the distances from the closest points of the training data to the hyperplane to the hyperplane itself. The solution to the problem finds these points, called *support vectors*, and sets the margins according to them. Lecture notes [2] can be read for further theoretical explanation and demonstration.

For ease of comprehension one can imagine an SVM problem in two dimensional space. On this, a hyperplane is a line dividing a plane into two parts, where classes lay on separate sides. Figure 2.1 shows an abstract example of labeled data separated by a hyperplane with its margins set, obtained by the SVM algorithm.

In the space showed in Figure 2.1 the data is linearly separable, but with real world data full separation of data using a linear hyperplane is often an impossible task. Therefore, most implementations of SVMs include three main parameters to tune a SVM classifier. These are the kernel, the regularization parameter and the gamma parameter.

The kernel specifies a function which will transform the feature space into a possibly larger space,

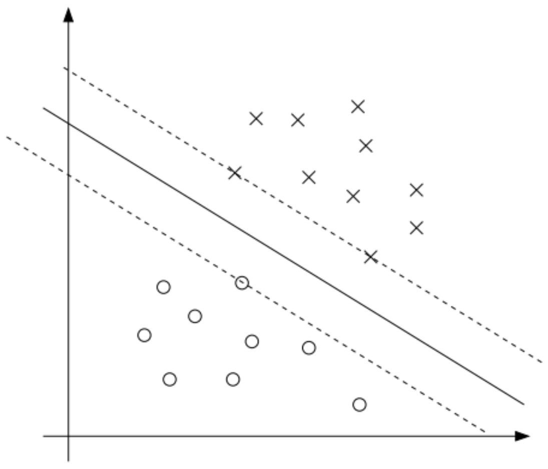

Figure 2.1: SVM example, extracted from [2]

where the data can be linearly separable. Non-linear kernels are used when the data is not linearly separable in the feature space, that usually happens when the feature space is too small and/or there is a lot of data and it is mixed.

The regularization parameter determines the cost of missclassification of training samples. A higher regularization value will make the SVM prioritize separability, even if that means very narrow margins, because the cost of missclassification is high. A lower value of regularization will make the SVM look for wider margins, prioritizing the margin for error, even if that means some missclassification occurrences, because the cost of missclassification is low. Summarizing, this parameter controls the trade-off between margins width, which represent the flexibility of the classifier, and missclassification rate, which represent the base accuracy of the classifier.

The gamma parameter controls how many training point influence the calculation of the separation hyperplane. Higher values will make only the closest point to the plane influence its calculation, while lower values will include further points into its calculation.

In text classification problems, the most common approaches use terms TF-IDF values as features, making the feature space dimension as large as the number of terms. This results in a very large feature space, which means most of the times a linear kernel is enough for data separation. The use of non-linear kernels is unnecessary in this cases and makes the system much less time efficient. Using SVMs with Word2Vec features most of the times reduces the dimensions of the feature space, and it can get better results than with TF-IDF values.

2.1.3 Logistic Regression

Like Naive Bayes, Logistic Regression is also a probabilistic model.

Logistic Regression uses the logistic function, also called sigmoid function, as probabilistic model to predict a dependent unknown variable. The sigmoid function is presented on equation 2.6

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.6}$$

In a binary classification problem, we have the variable $y \in \{-1; 1\}$ that we want to predict. Sample $x = (x_0, x_1, \ldots, x_{n-1})$ is linearly combined with a set of parameters $w = (w_o, w_1, \ldots, w_n)$ to form a linear model. Logistic Regression uses this model to obtain its probability model, which is showed on equation

2.7

$$P(y|x) = \frac{1}{1 + e^{-yw^T x}} \tag{2.7}$$

Logistic Regression use the maximum-likelyhood estimation method to obtain the w coefficients. This involves a minimization problem that focuses on minimizing the error in the probabilities predicted by the model when it takes as input the training data. For example, for every training sample x labeled with $y = 1$, we would want to have $P(y = 1|x)$ as close to 1 as possible.

Although fundamentally different models, both Logistic Regression and SVM build their model by resolving an optimization problem. For a linear SVM, both approaches solve a similar optimization problem [26]. *LibLINEAR* [26] is a software library that takes advantage of this and implements both models in a single linear classification package. This library is used in the development of this thesis.

2.1.4 Fuzzy Fingerprints Classifier

The FFP-C has been an emerging technique for text classification. Studies show it can successfully categorize text documents, ranging from a dual-classification problem [27] [28] to one with dozens of classes [22] [29], while outperforming common approaches such as NBs and SVMs when applied to certain problems.

In computer science, fingerprinting a resource, such as a document or any computer file, consists of mapping the resource's data, which can be arbitrarily large, into what is called a fingerprint, which is of fixed size and uniquely identifies the resource [29]. This compression makes resource comparison more efficient and avoids having to read the full content to uniquely identify a resource.

The FFP-C takes advantage of this technique, creating compact fingerprints of documents for text classification purposes. Each class fingerprint comprises the top-k most frequent tokens, usually distinct words or stylometric features, in the documents of the class, with k being an adjustable parameter. Formally, the fingerprint of class j is a list of k tuples $\{v_i, n_i\}$, where v_i is the i-th most frequent token and n_i its corresponding count, ordered by n_i in descending order, and $n_i \in F_j$ which is the set of tokens of class j [29]. To this ordered list is applied a fuzzy function, to assign a membership value to each feature based only on its rank in the list [29]. This is a strictly decreasing function, with maximum equal to 1. In its most simplistic form, a fuzzy function can be of the form:

$$\mu(i) = \frac{k - i}{k}, \qquad i = 0, \ldots, k - 1 \tag{2.8}$$

Occasionally, after getting the top-k tokens but before the application of the fuzzy function, in addition to counting the frequency of each token, a value called Inverse Class Frequency (ICF) is multiplied to each token frequency, generating the final value over which the tokens will be ordered. This ICF is and adaption of the IDF, but classes are used instead of documents to identify common tokens between classes. Formally,

$$icf_v = \frac{J}{J_v}, \tag{2.9}$$

where J is the total number of classes and J_v is the number of classes where token v is present [29]. This approach is used when documents are too small, and so tokens have a high probability of having similar frequencies within a class.

To find the belonging class of an unseen document D, its fingerprint is calculated the same way a class fingerprint is, without the ICF term as that is only applied to classes. Having Φ_D, the fingerprint of D, and considering Φ_j the fingerprint of class j, document D is assigned to the class j resulting from the formula:

$$\underset{0 \leq j < J}{\arg\max} \quad sim_D(j) \tag{2.10}$$

where

$$sim_D(j) = \sum \frac{\min(\mu_v(\Phi_D), \mu_v(\Phi_j))}{k} \tag{2.11}$$

where $\mu_v(\Phi_x)$ is the membership value associated with the rank of the token v in fingerprint Φ_x.

A threshold can also be set, to assign candidates to a ”negative” class if the similarity score of equation 2.11 is below that threshold, for every class.

2.1.5 Classification performance evaluation

It is common to evaluate a text classifier's performance with metrics of accuracy, recall (or true positive rate), precision and F_1-score (also called F-measure). To define these, it is useful to first know the definition of a confusion matrix.

A confusion matrix is the cross-referencing of the classification results with the actual class memberships of the classified samples. To give an example, in a binary classification problem there exists two classes, the positive class and the negative class. Therefore, there are four possible outcomes when cross-referencing the resulting labels with the actual ones: a True Positive (TP), when a positive sample is classified as positive; a True Negative (TN), when a negative sample is classified as negative; a False Positive (FP), when a negative sample is incorrectly classified as positive; a False Negative (FN), when a positive sample is incorrectly classified as negative. Table 2.1 shows how these occurrences are normally displayed in a confusion matrix. In each cross-referenced cell should be the frequency of each outcome.

		Predicted class	
		Positive	Negative
Actual class	Positive	TP	FN
	Negative	FP	TN

Table 2.1: Confusion matrix of a binary classification problem.

All classification performance measures can be derived from the confusion matrix. Equations 2.12, 2.13, 2.14 and 2.15 show how the measures of accuracy, recall, precision and F_1-score can be calculated, respectively.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.12}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.13}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.14}$$

$$F_1 score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.15}$$

Accuracy is the fraction of correctly classified samples. Recall is the proportion of positive samples classified as so. Precision measures the fraction of positive predicted samples that are in fact positive samples. F_1-score is the harmonic mean between precision and recall.

This can also be applied to multi-class classification problems. One has just to consider one class as positive and the rest as negative, when he wants to apply this framework to the considered class. However, a common approach is to do a weighted average of all metrics between all classes to get an overall performance. For example, equation 2.16 shows how to calculate the weighted precision of a classification problem with N classes. Here, n_i is the total amount of samples whose actual class is i and n_T is the total amount of samples.

$$Precision = \frac{\sum_i^N Precision_i \times n_i}{n_T} \tag{2.16}$$

2.2 Web Crawling

A web crawler is one of the principal systems of modern information retrieval. Web crawling is the act of going through a web graph, that represents a network where each node represents a resource (usually a document), gathering data at each node according to its needs. According to [30], web crawling dates back to 1993, when one was made with the purpose of generating statistical information about the web growth.

The need of web crawling rose with the large increase of web resources available on the Internet. Through the analysis of very well known search engines like Google[1] and Bing[2], Kunder created a website [31] which estimates that, at the time of writing, the indexed Web contains at least 4.58 billion pages. Additionally, there is the hidden web, that is estimated to be around 70% of the total Web. This makes the Internet the largest and most accessible information resource in the world. However, such amount of information cannot possibly be processed manually, only computers have the processing power to do so. That is why the crawler is the main component of search engines, nowadays.

Nonetheless, it has many more applications. It can be used for web data mining, for example to compare shopping prices or sentiment analysis. It can be used to collect bilingual corpus for translation purposes [32], to build an archive [33], or to gather local news of one's interest. The list of applications is endless, it all depends on the users' information need.

2.2.1 Basic Architecture

A web crawler always begins its activity from a set of chosen URLs, called seed URLs. From them it goes through a series of steps, that can be followed analyzing its architecture, depicted by Figure 2.2. Its composition is made of:

1. URL frontier - a component that stores, usually in a strict order, the URLs to be fetched.
2. Fetcher - it fetches the URLs fed by the frontier, following communication protocols like HTTP, HTTPS and FTP to interact with web resources.
3. Domain Name System (DNS) resolver - it interacts with the fetcher for the purposes of web resource name resolution.
4. Parser - it extracts the information on the web resources pointed by the fetched URLs and extracts all the URLs inside those resources.
5. Duplicate content detection module - it interacts with a storing system containing the fingerprints (also called signatures) of already fetched resources (documents) to detect repeated content.

[1]https://www.google.com/
[2]https://www.bing.com/

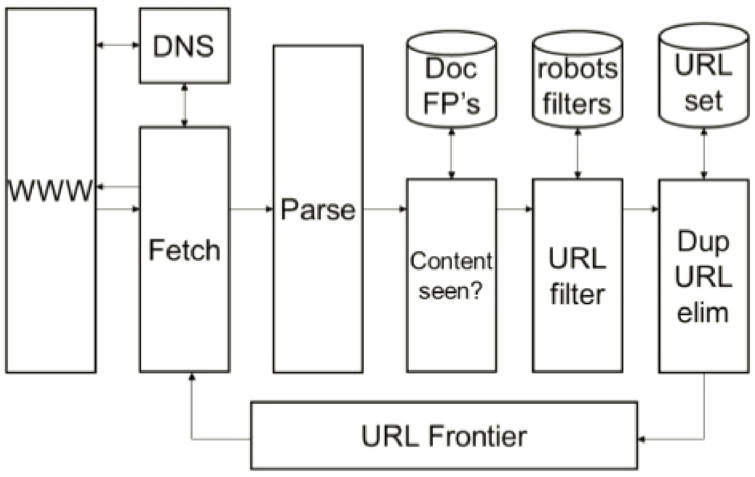

Figure 2.2: Basic architecture of a web crawler, extracted from [3]

These fingerprints are the results of hash functions applied to the documents' content, and they are made to avoid having to compare the full content of documents.

6. URL filter module - a module for filtering the URLs of a website, respecting the robots.txt file (described in Section 2.2.5), which sets constraints on some resources of a website.

7. URL duplicates elimination module - a module which prevents URLs from being added to the frontier if they are already in it or if they were already crawled.

2.2.2 Types of web crawlers

Although crawlers began as simple programs whose goal was to crawl the entire web, gathering as much information as they could, as time advanced, user's needs and the Internet structure evolved as well, together with the growth of available information. Thus, different types of crawlers began to appear. In [34], a very detailed State-of-the-art review about the different types of crawlers, pointing out their use and their potential for future areas of research. Other relevant references include [35], [36] and [30].

The following Sections describe the basic types of web crawlers, highlighting the types to which the crawler system implemented on this thesis belongs. Additionally, state-of-the-art examples of each type will be given.

Broad and Focused crawlers

Concerning the target results of a crawl, we can differentiate crawlers into two categories, broad and focused crawlers.

- **Broad crawlers**, as every crawler, begin from a limited set of URLs, and from them they aim to collect all the webpages whose URLs they encounter. They are the backbone of a general search engine, like Google or Yahoo[3]. However, they can be also used on an intranet, to later index all resources in it. These crawlers usually follow a breadth-first approach in URL visiting, following the URL extraction order, or a link analysis based approach, where URLs are fetched according to a score that is given to them based on their connections to other websites, i.e. their outlinks to and inlinks from other websites. Famous link analysis algorithms are PageRank [37] and HITS [38].

[3]https://www.yahoo.com/

- **Focused, or preferential, crawlers** are only interested in a subset of all the URLs they extract. They have an extra component, the guiding component, that filters and/or orders the URLs for fetching based on the domain they are part of, on the content and/or structure of the webpage where they were found and/or on the geographic location of the server hosting the webpage. This component takes care of classifying the relevancy of webpages with respect to a specific topic, analyzing its parsed content, domain, location, surrounding text of its URL. This last feature can range from just the anchor text of the URL, which is the hyperlinked text associated with the URL on the webpage it was found, to the full content of the webpage. Newly discovered URLs are sorted in an order influenced by the results of the classification process. This type of crawler will be the main focus of this Master thesis, with the crawler's guiding component being a combination of a text classifier and a distiller (a URL classifier).

Continuous crawlers

Nowadays information is entering in the Web at a fast pace. Websites keep changing and updating their pages to cope with this evolution. Therefore, depending on the kind of application a crawler is supporting, there might be the need of keeping up with the webpages updates. Regarding this we can distinguish between two types of crawlers: snapshot and incremental crawlers.

- **Snapshot crawlers** crawl each webpage only once. They retrieve a snapshot of a webpage at a certain time and usually stop the crawl when they reach the target number of pages desired. Snapshot crawling is useful for large scale crawls, as it can be concerned only with coverage, minimizing the amount of state information it has to store for each webpage. Because it does not need to revisit webpages, it just needs the URI (or a fingerprint of it) to check if certain resource was already fetched or not [39]. However, for the majority of applications, an up-to-date index of webpages is needed. One could think webpage maintenance could be achieved with this approach, doing periodic re-crawls starting from the same seed URLs, but this most certainly would bring inconsistency issues because the content of some webpages would change, referencing different pages from the last time. Thus, the crawl would reach out to different pages and possibly miss old ones. Additionally, one would fetch also the unchanged documents, resulting in unnecessary effort.

- **Incremental, or continuous, crawlers** try to always have the last image of a webpage. They appeared as a response to the snapshot crawler problems. While crawling the Web, they make sequential scheduled fetches to a same webpage, to examine their updating rate, and adjust their schedule depending on whether or not and when it has been modified. If it changed, they discard the old information and parse the webpage again to store the updated one, or keep an additional snapshot of that webpage, for example, to keep track of the evolution of the Web[40]. Downsides of this style of crawling are the need of storage of more URI state information, like the last time it was fetched and last modified dates, and the time consumption spent in revisiting, rather than expanding the webpage index. Nonetheless, this is the most used approach of the two, specially in focused crawlers, considering that applications using these gain more in having a selective up-to-date index rather than a very complete and extensive one, which is most likely out-of-date.

The work presented in this Master thesis follows an incremental crawling approach, since keeping the information regarding user innovative solutions up-to-date is a natural requirement. It takes advantage of the Nutch Adaptative Fetch Schedule [41] functionality, which has a default and bounded re-fetch interval, that is decreased or increased by a configurable factor, depending on whether a webpage has been modified or not, respectively. It can even include the last time a page was modified in a formula to adjust that factor. Other State-of-the-art approach to incremental crawling is the one presented by Singhal et al. in [42], where each URL starts with the same refresh rate, that is adjusted according to a formula that depends on the probability of change of a webpage. This probability is an estimation calculated through the frequency of changes on the page after a number of hits. Furthermore, URLs are distributed per queues refreshing time wise, i.e., URLs with lower refreshing times are put in lower indexed queues, which are hit more often than URLs in higher indexed queues. Additionally, within each queue, URLs are sorted by PageRank score, to also add a component of relevance to this crawler. A more complex example is shown in [43], where the authors claim that it is not possible to check if pages have been modified or not without actually crawling them. Therefore, they propose a method for predicting lately modified and newly added pages, based on a parameter they call modifying probability, that depends on a page structure pattern and on the hierarchical structure of websites. The time interval set to re-crawl a webpage is calculated through that page's modifying probability and the portion of candidate pages to re-crawl are set according to the pages' structure patterns.

Parallel crawlers

The need to parallelize the process of gathering web documents emerged with the rapid expansion of the Web. Therefore, researchers started to create strategies to parallelize crawling processes. This led to the appearance of techniques for centralized and distributed crawling, as opposed to single-process crawling.

- **Single-process crawling**, as the name implies, does all the processes of a crawler in a sequential manner. This way of crawling has many limitation. The main one is the fact that all processes run sequentially, although many crawling processes can be run independently. For example, all the processes concerning different websites can be parallelized. Other limitation is the fact that a website can block a machine from accessing its content if it detects a large amount of accesses in a short amount of time from that same machine. This can be avoided if these accesses are distributed amongst different machines.
- **Centralized parallel crawling** is the most used crawling technique nowadays. It is usually referred to as distributed crawling, although there is a difference between this and a fully distributed approach. It usually employs a master-slave strategy, where the master takes the control of the main flow of the crawl (e.g.: initialization, data storage, process creation and termination) and distributes the URLs between the available slaves, for them to fetch, parse and/or index. This distribution strategy can be domain-based, geographically-based, topic-based, amongst others.
- **Fully distributed crawling** is the same as centralized crawling, but the control of the program does not reside just on one machine. This avoids having a single point of failure and has a distributed file system where distributed storage and replication may occur. It has higher availability and fault tolerance than a centralized approach but it is often harder to manage and monitor, as there is no central point of control.

[44] is an example of a fully distributed crawler. Here Boldi et al. decided to build a crawler with complete decentralization of every task. This crawler presents a fault tolerant architecture where all the components, except for one, work in an asynchronous fashion, providing the highest degree of parallelization possible. It also assigns URLs to available agents using consistent hashing, a hashing technique originally made for distribution of Web caches, to reduce the number of remapped hosts over the agents.

The Nutch approach for distributed crawling is one of the State-of-the-art approaches. It runs on top of Hadoop[4], a framework which enables distributed processing and storage through simple programming

[4]http://hadoop.apache.org/

models. This alone takes care of data replication and fully distribution, making the system fault tolerant. Even though the program has just one master machine, if it fails all its data can be recovered and the program can still go on, delegating other machine of the cluster as master. Regarding the distribution strategy, Nutch has a queue per unique host and the number of threads per queue can be configured. This allows for parallel processing per host, although an interval per request to host is set for politeness reasons.

On the other hand, the approach in [45] does its distribution of URLs amongst machines based on an ontology. The system is based on a master-slave approach, where each slave has its own crawler, targeting pages which belong to a subtopic of the target ontology by starting from seed URLs chosen by the master. The master keeps a pool of jobs in a shared memory space to distribute amongst the slaves. In this shared space, the local results of each slave crawler are also stored, to avoid page duplication amongst slaves.

In this work, it was decided to follow a single machine approach, using its multi-core capabilities to parallelize tasks. At the time of writing, the post-processing of the relevant pages resulted by the crawler (summarizing for posting on the final platform, background check, medical screening) is still planned to be done by a few amount of people. Therefore, there is no need to overwhelm these analysts with results. Thus, it was decided to work on the effectiveness of the crawler, trying to provide relevant results with good precision and recall, rather than on its efficiency, which would focus on delivering more results in less time. Nonetheless, as this project's crawler uses Nutch, which runs on top of Hadoop, inter-machine parallelization is easy to configure. So this crawler has the capability to deliver at a faster pace when needed.

Other crawlers

There are other types of crawler that fall off of the scope of this thesis, but for the sake of State-of-the-art review they will be briefly described.

A **mobile crawler** is one of them. It is mobile in a sense that it does not run on a predefined set of machines. Instead, the processing is done on the server side, i.e., on the host machine where the target documents reside. This reduces the bottleneck effect that the network requests have on the crawler, by retrieving just the processed information of a webpage to the local database, instead of downloading the entire page. Moreover, it also reduces the computational load at the client side, because most processing is done on the server side. An example of this type is the one on [46]. Here, the authors proposed a crawler which implements a master-slave design pattern. The master node runs a search engine locally and uploads the slaves to the server side. They do all the content processing at the host, whilst building their index. This process reduces the network load by solely sending the index to the search-engine (master node). Obviously, mobile crawlers come with some deployment issues, as not every host on the Web will let a crawler deploy a program on its server. But it is safe to assume that mobile crawlers can be of more use on intranets.

There is also the **hidden web crawler**. The hidden web is the part of the web that is not indexed by traditional crawlers, because there are no hyperlinks to this hidden information on indexed webpages. This information is hidden behind forms or query interfaces, therefore, a hidden web crawler has to implement different techniques from traditional crawlers. These include building meaningful queries to access information through query interfaces, identifying searchable forms which hide information, incrementally revisit hidden databases, extracting and study labels from forms and their respective values to gain access to hidden data and determining the topic of information behind hidden resources forms [34].

2.2.3 The Focused Crawler Guiding Component

In the core of a focused crawler lies its guiding component. It is the component that navigates the crawler through the Web, according to the user's information needs. [34] presents a very complete State-of-the-art review about the different subcategories of focused crawlers, each category approaches one or more types of guiding components.

In the majority of cases, a guiding component can follow webpage relevancy through four main approaches: URL based, content and structure based, topically link based and machine learning based approaches. These tackle different aspects of the webpages being visited, however, they can be combined. In fact, machine learning based approaches come hand-in-hand with one or more of the other three.

URL based

URL based approaches conduct the crawler to follow URLs which are of a specific host or domain or which contain a specific word or expression within a selected dictionary. The use of regular expression filters is a flexible and intuitive technique for this kind of approach. One advantage of this approach is that it decides whether a webpage is relevant or not without having to download it beforehand. It is probably the most efficient approach, on the other hand, it doesn't offer much flexibility and accuracy while looking for topic-specific content.

Content and structure based

Content and structure based approaches cover a wide variety of cases. Most known approaches fall on this category, even if not purely content based, most approaches must exploit a webpage content to test it for relevancy. These consist of using an established taxonomy, a set of keywords, a prepared set of documents or queries, to compare with the content of the webpages being visited. Approaches that fall solely on this category include: using a Vector Space Model, which is a bag-of-words model where documents are represented as vectors of terms, to represent the array of target words and the full textual content or URL anchor text of the document being analyzed, calculating its relevancy through scores of syntactic or semantic similarity [47] [48] [49]; using the webpage structure (specific parts of the Document Object Model (DOM), specific metadata) for keyword spotting or link-context selection, assigning different relevance levels according to the location [32] [50]; query search in integrated search interfaces [51]. The downside of this methods is that most of the times the web documents must be downloaded before they are checked for relevancy, contrary to the URL based approaches. This is not the case for methods that just use the URL anchor text to check for document relevancy, however, this approach tends to perform significantly worse than using the full (or most of the) content of the webpage the URL points to, as the latter offers a much complete description of the document being analyzed.

Topically link based

In topically link based approaches, a focused crawler exploits the topical connection between webpages to predict their relevancy to the topic of interest. Not to confound with regular link based approaches, that broad crawlers use. These link based approaches first calculate the similarity of a webpage to the relevant topic, usually by content based methods, and then decide whether or not to follow its outlinks, based not only on its topic similarity but also on its topical connections to other webpages. These connections can be expressed by context graphs [4] [52], which consist of a central node that represents the topic being perceived, connected to several nodes that represent topics related to the central one.

Nodes further away from the central node represent more general topics, with a wider topic definition and conceptually further away from the central topic, while the closest nodes have a narrower topic definition and are directly related to the topic of interest. Each node has a set of keywords or documents that represent the concerning topic. A context graph example can be seen in Figure 2.3. Other approach that takes advantage of topical connection without using a context graph is the one in [53], in which the authors apply a modification to the well known PageRank algorithm to express topical concern. This modification allows the crawler to stay on the topics of interest while considering webpages link scores.

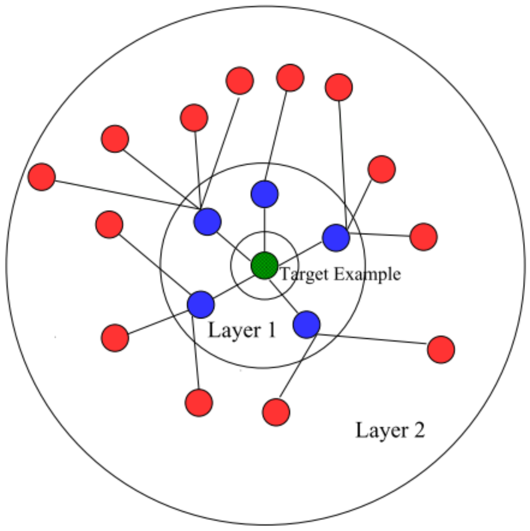

Figure 2.3: Context graph example, extracted from [4]

Machine learning based

Given the current popularity of machine learning, these approaches are probably the ones getting the most attention. A common practice is to use a text classifier to guide a focused crawl. It is very popular because of the variety of web documents currently available, that can be used as training data. Several popular text classifiers have been tested:NB,kNN, NN and SVM. [54] has done a variety of tests, crawling the web for topical information, to evaluate the performance of NBs, SVMs and NNs for focused crawling purposes, using a Vector Space Model with TF-IDF weights. They show that SVMs and NNs outperform NBs for focused crawl guidance, the first two having similar performances. Still, NBs classifiers are used due to their time efficiency and the fact that a likelihood threshold can be set to discard unseen negative examples when one lacks negative examples in the training data [4], amongst other reasons. [55] uses an ontology-driven approach to train a NN for crawl guidance. They demonstrate the superiority of NN based focused crawling, comparing with other non machine learning approaches, and combining it with an ontology definition makes the NN based approach reach higher performances.

Besides common supervised text classification methods, other machine learning techniques have been applied. The authors in [56] have applied a reinforcement learning algorithm to update the weights of entities within an ontology, which guides a focused crawl. In [57], the authors implemented a genetic algorithm to expand their initial set of keywords, that guided the focused crawl, to achieve higher precision and improve the crawl path. A Hidden Markov Model (HMM) was used in [58] to predict the links which would lead to relevant results. The model was built from a concept graph, originated from a manual browsing of the Internet. Each time the crawler reaches a page, it is assigned to a state of the HMM. Each state is characterized by the number of hops the crawler needs to get to the target topic pages.

This Master thesis crawler will have a guiding component that is machine learning and content based. It will use a two-layered text classifier to identify and score patient-driven solutions and a content based distiller to set the fetching importance of newly discovered URLs. Their implementation will be described in Section 4.

2.2.4 Performance Evaluation

For a focused crawler, we want to check if the pages it is retrieving are relevant for the user's needs and if it is doing so efficiently. Therefore, important indicators that test the focused crawler's performance are:

- *Harvest Rate*: This measure was first used in [59]. It determines, at several points of the crawl, the fraction of pages crawled by the system that are relevant to the focused topic. It can be seen as the precision measure of an information retrieval system. It examines all the crawled pages and, through a classification method, confirms if a page is relevant or not. Usually it is very laborious to manually classify the pages retrieved, therefore, in some cases, a classifier or a combination of these is used to label the pages [59], [60]. This component, usually called evaluation classifier, is trusted to present a fair classification because it is usually trained more throughly. It may use a larger training set [60] and more complex features thus being more trustworthy than the crawling classifier, as it is expected to hold more knowledge about the topic.

- *Target Recall*: This measure, throughly described in [61], estimates, during the crawl, the fraction of total relevant pages that are fetched by the crawler. It can be seen as an estimate of the recall measure of an information retrieval system. This estimate can be obtained by pre-defining a set of target pages, and taking, at several points of the crawl, the fraction of those pages that were already crawled.

The formulas for harvest rate and target recall can be seen on equations 2.17 and 2.18, respectively. Here, S_C^t is the set comprising all the crawled pages at time t, R is the set of relevant pages on the Web and $R_T \subset R$ is the set of target pages defined.

$$HarvestRate = \frac{|S_C^t \cap R|}{|S_C^t|} \tag{2.17}$$

$$TargetRecall = \frac{|S_C^t \cap R_T|}{|R_T|} \tag{2.18}$$

2.2.5 Problems and limitations

Most crawlers are built to run through the Internet, a public virtual place where there are no rules for how to create a webpage, and hosts expect to have their servers up and available for everyone to visit their websites. Therefore, crawlers must be alert for eventual webpages that may interfere with the crawling path, and they also must be wiling to respect hosts.

Concerning crawler politeness, a crawler is expected not to overload hosts with requests, creating an accidental Denial of Service (DoS) attack [62]. A solution for this problem is setting a maximum frequency of requests to the same host, and having separate queues of URLs, that handle separate hosts, alternating between queues to fetch URLs.

Another thing crawlers are expected to respect is *The Robots Exclusion Protocol* [63]. This standard consists of a file written by hosts, called *robots.txt*, which states some rules a crawler should follow, along with optional information about the website. These rules can prevent all or specified crawlers (user-agents) from accessing certain parts of a website or set a crawl delay for repeated accesses by a same user-agent. An example of a robots.txt file can be seen in Figure 2.4. The first rule prohibits every user-agent from accessing the folder ”directory_0”, while setting their repeated access delay to that website to 10 seconds, and the second rule prohibits only the agent named ”SampleCrawler” from accessing the folder ”directory_1”.

```
User-agent: *
Crawl-delay: 10
Disallow: /directory_0/

User-agent: SampleCrawler
Disallow: /directory_1/
```

Figure 2.4: Robots.txt example

There are no laws preventing a crawler’s bad behavior, however, these systems are expected to be polite and respect the robots.txt standard.

On the other hand, hosts are expected to be polite as well, unfortunately some hosts set traps for crawlers, intentionally or not. These include setting a crawl-delay to absurd time periods (that can go to days of waiting) and websites that dynamically create pages for a crawler to keep following indefinitely. Ways to avoid falling on these traps are: ignore websites that have a crawl-delay higher than a certain amount of time; establish a maximum number of URLs to fetch per host, in total and/or per crawler iteration; establish a maximum reachable path depth per website. Sometimes, these traps are purposely set to catch impolite crawlers, that do not respect/read the robots.txt file. In those cases, the hosts write a rule on the file to prevent the polite agents from falling into the trap, catching just the impolite robots that do not regard the file.

2.2.6 Analysis of State-of-the-art open-source crawlers

For this project, instead of developing a web crawler from scratch, which would end up being very inferior to the ones already available, it was decided to choose an open-source crawler and adjust it to fulfill the project’s goal. This Section describes some of the State-of-the-art crawlers that were taken into account, the ones that were explored more deeply. It gives an overview of them, their advantages and disadvantages, which one was chosen and why.

ILSP-FC

This crawler [32], [64] was originally made for gathering domain-specific monolingual and bilingual corpora. It was first released in October 2012 and its latest release was in June 2016.

Like most crawlers, it receives as input a list of seed URLs. But additionally, a user has to input a list of keywords representing the domain the user wants to search for. Each keyword has a weight and optionally one or more subclasses of the domain that the list is characterizing. Its architecture includes modules for page fetching, web page normalization and cleaning (e.g.: metadata extraction, boilerplate removal), a text classifier which leads the crawl, a link extractor, a module for exporting a webpage to an XML file, which follows the cesDoc encoding standard, with the option of transforming it into an HTML file for easy corpus browsing, a page deduplicator and a detector of parallel documents, i.e., documents written in different languages that are supposed to have the same content. Usually one is the translation of the other or they are both different translations of a single document.

Its value for the project is in it's text classifier and link extractor. The classifier follows a content and structure based approach. It assigns a score to a webpage depending on the frequency of the terms that match the domain description (list of keywords), their weight and their location in the webpage. The link extractor assigns a score to each link of the webpage currently being analyzed, which depends once again on the keywords term frequency in the link's surrounding text and their weight, but also on the page score and the number of links in it. This score then serves to set the fetching order of the unseen links. These two components were seen as promising because we could define the domain as extensively as we want, configuring and testing with different weights, to achieve promising results. We could even do a linguistic analysis to the documents already posted on the Patient Innovation platform to find the keywords that better describe the domain we are looking for.

The main drawback of this crawler was the fact that it was not very extensible comparing to some of its competitors, so it would be hard to add new functionalities to it, including the indexer, and try to plug new classifiers. Although promising, the text classifier was also too simplistic, with limited margin for configuration, so it offered little flexibility to test out new approaches and in its form it could not capture the details of the topic being perceived. Also, this crawler is poorly documented, being hard to understand it and know how to adjust parameters and add functionalities.

Heritrix

This crawler [33], [65] was first released in December 2003 and it is still, at the time of writing, being maintained and updated by an open-source community. The development of Heritrix was started by the Internet Archive[5]'s team and its original purpose was to archive webpages, storing them as a snapshot on the date the webpage was visited.

This crawler can be used for broad crawls or focused crawls. Its architecture is composed of 3 main components: the Web Administrative Console, the Crawl Order and the Crawl Controller. The Crawl Order is a configuration object that represents a crawl and its parameters, amongst them there are the URLs seed list, URL filters and maximums and minimums (e.g.: bytes, threads, time interval) that can be set overall or per domain basis. This can be sent to the Crawl Controller to execute the crawl itself, which includes core modules for page fetching, link selection and content extraction (which is compatible with a large variety of MIME-types) in addition to other extra modules. The Web Administrative Console is a web-based user interface that lets the user interact with the other 2 components. The retrieved content is saved in a compressed file format, the Internet Archive's specific ARC file format. A focused crawl is guided by precedence policies that choose the next URL to follow based on the precedence value assigned to it. By default, these values are obtained considering only the URL, the number of hops or target pages reached. However, content-based precedence policies have been made by external entities and are freely available.

Some of its highlights are the Web Administrative Console, that lets the user intuitively configure a crawl. It lets him start/pause/stop a crawl and keep checking its status through logs and reports. A user can even adjust parameters mid-crawl through this interface. Heritix is already very complete, in addition, it is also very extensible and very well documented, backed by a large and active community.

The main drawback is the fact that as it was made for archiving, it is not programmed to do incremental crawling, although it was already done [39]. By default, it is set to store snapshots of webpages in compressed ARC files and keep appending files without deleting old ones, even if they contain (new or repeated) content of already visited websites, so it is hard to maintain and update specific website content. Also, although it is not hard to integrate an indexer with Heritrix, due to its extensibility, there are other open-source crawlers which already offer integrated indexers.

[5]https://archive.org/

Nutch

The Nutch [66], [67] was developed by *The Apache Software Foundation*, a very well known non-profit corporation due to its contribution to famous open-source projects. Its first release was in August 2005 and, like Heritrix, it is still maintained and updated at the time of writing. Its main goal was to offer more transparency and control, to users and developers, over Web search processes, as opposed to other commercial search engines, like *Google*, but still achieving the same performance.

Nutch has a very intuitive architecture, based on batch processing. Its batch operations, by sequence, are: *Inject*, *Generate*, *Fetch*, *Parse*, *UpdateDB*, *InvertLinks*, *Dedup*, *Index* and *Clean*. Like the crawlers mentioned above, it first starts from a list of URLs called the seed list. The *Inject* operation injects all these URLs in the *CrawlDB* (the crawl database), which will store information about the crawl (URLs fetched and to fetch, their status, fetching schedule, fetching order, etc). The *Generate* operation generates a Segment, used to store all the data about the URLs to fetch (the raw content of the webpages to which they point to, the parsed content, the outlinks found, the metadata, etc), and picks a batch of URLs to fetch. The *Fetch* and *Parse* operations then fetch those links and parse the content retrieved from them, respectively, and store it in the Segment generated. A score is assigned to every URL, that can be manipulated during most part of the Nutch crawl phases. That score will be used to set the fetching order of the URLs. With the data obtained from the previous steps, the crawl database is updated in the *UpdateDB* step, which adds new links for fetching and can rearrange the fetch list order. The *InvertLinks* operation builds the Web graph representing the URLs fetched so far (the *LinkDB*), which contains information about the inlinks and outlinks of each URL. That information can be used for link analysis, and even influence the score of each URL. The *Dedup* step marks URL duplicates, i.e., different URLs that point to webpages with identical content. It does so by comparing the recently fetched URLs' hash signatures with the ones already on the CrawlDB. Two URLs with the same signature point to identical content. The *Index* operation indexes the new information on an external web index. This last step is done by an external application, for example, by *Solr*[6], also maintained by *Apache*, which additionally offers, among other things, a Web UI for index managing and text searching purposes. Finally, the *Clean* task removes URLs marked as gone and duplicates from the index. Gone URLs are those whose fetching process received a 404 Hypertext Transfer Protocol (HTTP) response.

This crawler is very extensible through a system of plugins. A user can develop a custom plugin, which must implement at least one of a list of the available endpoints (plugin interfaces). Through this system one can implement a customized protocol, a URL, parse, index or a scoring filter, which can manipulate URL information according to the user's needs. Nutch is also highly configurable, through configuration files and pre-made tools. These features offer a lot of transparency, flexibility and control over the system. As already mentioned, Nutch comes with an indexer and search platform plugged into it, *Solr*. This platform provides an index for the uploaded documents, granting the ability to efficiently search through them. This index can also be integrated in third-party applications. Other highlights of Nutch are the huge community around it and the tools it comes with that let you interact with the crawl and link databases and with the Segments.

Some of its disadvantages can be the fact that it has a lot of features, which can be overwhelming for new users, and some of them are poorly or not at all documented. Although Nutch has very informative logs being shown during the crawl, it misses a GUI that other crawlers have to interact with the crawl and better visualize it.

[6]http://lucene.apache.org/solr/

Final decision

Initially, ILSP-FC was used as the crawler for this project. It was simple to start with and, like it was mentioned above, its domain definition through a list of keywords was a promising feature that could be put to good use on this project. However, eventually it was understood that this crawler was very limited, offered very little control over the crawler classifier model and was hardly extensible. Therefore, this project's crawler was moved to Nutch. Its extensibility and configurability was very encouraging, and although it was overwhelming, its batch processing configuration and logging feature helped understanding what was going on in each process and by exploring different configurations and community posts most of its functionalities were learned. Its plugin-based architecture allowed to implement our custom text classifier and distiller through extension of one filter. Only through a custom classifier the details of the data that this project's crawler seeks can be expressed. Nutch's pluginable system gave a lot of flexibility and control over the system implementation.

Table 2.2 summarizes the list of advantages and disadvantages of each crawler, for the purpose of the targeted system implementation.

Crawler	Advantages	Disadvantages
ILSP-FC	• Simple crawler, easy to study and comprehend its basic features. • Promising integrated text classifier.	• Poorly documented, hard to understand it in depth. • Although promising, the classifier was too simplistic. • Not easily extensible.
Heritrix	• Well documented and it has a large community around it. • Extensible architecture. • Its Web Administrative Console allows user-friendly crawl management.	• Made for snapshot crawling.
Nutch	• Well documented and it has a large community around it. • Very easily extensible architecture due to its plugin system. • A lot of utility tools that let a user interact with the information being crawled. • It comes with an integrated search platform and index, out-of-the.box.	• Hard to pick up, as it is overflowed with features, some poorly documented.

Table 2.2: Advantages and disadvantages of each analyzed crawler.

2.3 Web Indexing

Indexing documents makes documents easily searchable by associating them with a compact, browsable list of terms or sections [68]. Web indexing focuses on creating an index for the content of web documents. Large scale indexes are usually built by an automatic process and their indexed documents are normally given by web crawlers. Documents may be indexed by their full content and/or by their metadata (title, author, date of creation, topic, key terms, etc).

Web indexes may be displayed like A-Z indexes, also called back-of-the-book indexes (because it is the style of displaying that people often see on the final pages of a book), where entries are alphabetically ordered and each entry holds a hyperlink to the corresponding webpage. Each entry may be named with the title or topic of the webpage it points to. The index may be organized through a taxonomy, showing a hierarchy of topics for ease of searching. However, when the quantity of indexed documents gets too large, developers opt by making hidden indexes, associated with search query interfaces, to easily search through the index by expressions, keywords or metadata, without having to visually run through them.

Making an index avoids linearly scanning all documents while searching for queries. The objective of a web index is to make that search as efficient as possible. To do so, a technique called *inverted indexing* was created.

2.3.1 Inverted index

An inverted index can be seen as a table which has as primary keys the unique terms that occur in the collection of documents. To each term is associated all the documents within which that term occurs.

The basic steps for creating an inverted index are: pre-processing the documents, through the text processing phase described in Section 2.1; running through the entire collection, assembling term–document_ID pairs; sorting the pairs alphabetically, by their associated term; identifying pairs which have the same term, merging them in a list, called *postings list*, where each entry holds the document ID, the term frequency of the term in that document, possibly alongside other information (e.g.: positional information). In the end, one gets a dictionary of unique terms, each one having a postings list associated. Figure 2.5 depicts a basic example of an inverted index, where each square with a number in it represents an object which holds information about the document whose document ID is the number within.

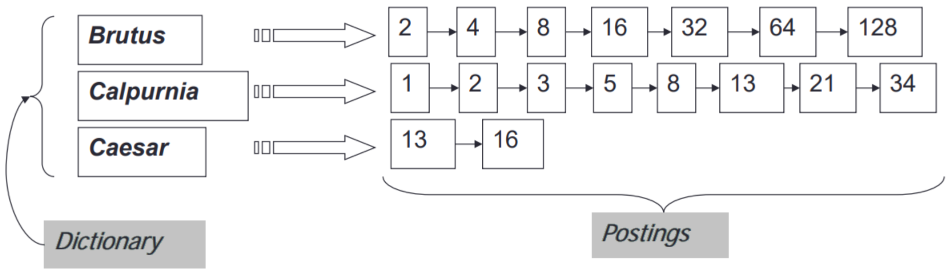

Figure 2.5: Inverted index example, extracted from [5]

It is possible to store inverted indexes in a variety of data structures. Usually one opts to use arrays or linked lists. Arrays are the best type of structure for this task, if one has a fixed collection of documents to index. They are faster than linked lists in searching operations and they are more space efficient, as they do not need space for pointer storage. However, they are not efficient when a changeable collection of documents has to be indexed, as array creation and copy needs to be done every time the index changes. In those cases a linked list approach is a better choice. Other sorts of data structures have been explored, like *B-trees* and *Tries* [69]. These prove to be faster while query searching than the basic approaches, on the other hand they have a high maintainability cost and occupy more space.

When searching for a query in a system with an inverted index, all the terms in the search query are matched with their corresponding postings list, after passing by the same text processing phase the documents did. Specific documents from the selected postings lists are then selected, depending on the type of query. For example, if it is a boolean query (a query joining the terms with logical expressions like AND, OR, NOT) the same logical operations are performed to the postings lists to achieve the final result. Other types of queries use statistical information, like TF, IDF or other weighting schemes, to rank the results and select an ordered subset of the postings lists. Techniques for query expansion, using semantic relations between words, are also used to improve the retrieval performance.

2.3.2 Solr

Solr [70] is a popular open source search platform, that can be integrated in all sorts of projects, to serve as a search interface over a collection of documents. It features a very fast searching ability, and that is only possible due to its highly efficient indexing. It is commonly used in conjunction with web crawlers to create search engines.

It is built on top of *Apache Lucene Core* project[7], a text search-engine library known for its high scalability, high-performance indexing and efficient text search capabilities. Solr can be deployed on a cluster, providing distributed indexing and searching, plus index replication, load balancing and sharding. It is highly customizable through its plugin architecture and it has a REST-like API which lets third-party programs interact with Solr's index and search engine through JSON, XML and CSV requests and responses over HTTP. It also comes with a built-in user interface for off-the-shelf index searching. These are just the main features of Solr, as it comes with many other extensions and functionalities.

Like most of web indexers for unstructured data (text), Solr uses an inverted index to index a collection of documents. The structure and content type of the index is managed through a *schema*, configured in the *schema.xml* file. This file states which fields (title, text, author, date, etc) should be indexed and/or stored, which field(s) is(are) the primary key, which is the default field for searching, which of them are required for a document to have and how to index and search for each type of field (through which type of text processing should it pass). Stored fields store the original content of a field, while indexed fields store the indexed content (pre-processed and indexed term-wise). The fast indexing process of Solr is due to the method it uses to store its index. Solr's index is stored in segments. Each segment indexes its unique set of documents, meaning each segment can be seen as an independent inverted index. When a document is deleted that document is flagged as so in the corresponding segment. When new documents are added, one or more new segments can be created to index these documents. Eventually, segments go through a merge operation, which will drop the deleted documents and compact the full index, making it smaller in size and making searches faster.

When a user searches a query in the Solr search engine, the system first emits a request, handled by the *Request Handler*, which will identify the type of request the user is sending (search query, insertion, deletion, etc). After, the query is passed to the *Query Parser*, which will check the syntax of the query and parses it to acquire the terms to search in each field of the index. It also passes through an *Analyzer* so it can be pre-processed (tokenized, normalized, stop words removal, etc) to match the index's terms. The index search is done and the results are retrieved. Before displaying them to the user, they first pass by any filters set by the user, to further narrow the results. The *Response Writer* then formats the results, displaying them to the user in one of three possible formats (JSON, XML or binary).

All this information was taken from Solr's and Lucene's documentations [71] [72].

2.4 Previous related work on health and user innovation

We can see the target topic as a combination of two concepts: health and user-innovation. Several systems to crawl for health information have been created. However, this Master thesis focuses on creating a system which searches for a specific fraction of that topic, it is not sufficient to implement a system with a broad approach of using just keywords or an ontology for an health topic specification.

To our knowledge, there has never been created a system for crawling the entire Web in search of user innovative health-related solutions. In spite of that, there are some works worth mentioning.

One of them is [73], in which the authors created the *ssmCredible*, a crawler for credible medical content, which also incorporates source and sentiment information. To check for credibility, candidate URLs are assigned a credibility score based on a two-tier graph (website-level graph, page-level graph) propagation algorithm that builds two graphs from online medical databases, that contain information about the trustworthiness of various forms of medical content. Credible URLs are then checked for rele-

[7]https://lucene.apache.org/core/

vancy, using their surrounding and anchor text found in parent pages. The relevance classifier integrates topic and sentimental relevance by combining text classifiers trained with n-grams and medical lexicons. Credible but not highly relevant URLs are not discarded. Depending on their context score, they may be used for tunneling purposes (they are not relevant, but may be connected to relevant webpages). Their system achieves better recall and precision results compared with many other methods, collecting the majority of relevant content early in the crawl.

This work does not consider user innovation, but it searches for a specific type of health information, which can relate to the problem of this thesis.

Another work worth to mention is [74], where Christensen et al. develop and thoroughly analyze the best configuration of a machine learning based system that can automatically identify ideas within an online community. The datasets used were extracted from a Lego online community, called Lego User Group Network (LUGNET). Two dedicated idea raters labeled the training data, identifying samples that were ideas and that were not ideas. It was just kept the samples on which both raters agreed. The same was done for the test set, but with five raters instead, and the samples kept were decided by majority voting. They experimented 252 classification models, comprised of different configurations of the training data and of the ensemble of linear SVMs used. In the end, they obtained satisfactory results both during the validation and testing.

Although this process does not involve identification of pure user innovations, an idea is defined as creative process that can result in a proposal, prototype or tangible product [74], and, according to Christensen et al, they have innovation potential.

Chapter 3

Dataset

This chapter describes the dataset used for the development and evaluation of this thesis' system, as well as its processing phase.

We start by introducing the type of information that can be found on the Patient Innovation platform. Then, we describe the dataset constitution and how it was acquired. Finally, we explain the dataset cleaning process and the different configurations kept.

3.1 The Patient Innovation platform

As mentioned before, Patient Innovation is an online open platform, created with the goal of sharing innovative health-related solutions developed by patients and informal caregivers. This thesis' system was developed with the purpose of finding these solutions automatically. So, its development was based on the knowledge that was already collected on the Patient Innovation platform. Therefore, it is useful to understand exactly what kind of information gets accepted and rejected on this platform.

The classifier used in this system will use the information on the Patient Innovation database as its source of patient-driven solution knowledge. Therefore, to clearly understand the type of knowledge existent on the platform, there are some definitions to consider. The hierarchy of the presented taxonomy is shown on Figure 3.1.

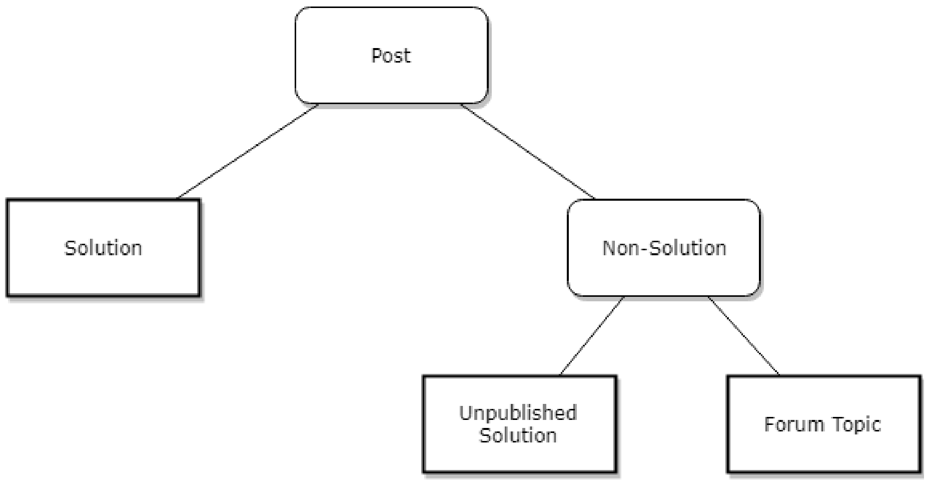

Figure 3.1: Hierarchy of post samples.

- **Post**: an accepted or denied publication on the Patient-Innovation platform; a patient-driven solution candidate.
- **Solution**: a user innovative health related solution, created by a patient or a caregiver, consisting of a new or modified device, an aid, an adaptive behavior or a low-cost alternative to an existing product.
- **Non-Solution**: a post that is not considered a solution.
- **Unpublished Solution**: a post with potential to be a solution, but considered otherwise because it did not pass by a screening process yet (which includes a medical evaluation) or, after the screening process, it was identified as a duplicate from a solution, there was not enough information to classify as so, it was not considered an innovation, not developed by a patient, caregiver or collaborator, it was of commercial intent, offensive, inappropriate or physically intrusive (dangerous), either because it involved a diet, drugs, chemicals, other biologics or an invasive device.
- **Forum Topic**: an idea for a solution, a health related advice or other information related to patient-driven solutions, but not actual solutions.

Given the definitions above, the focus of this thesis is to build a system that automatically searches for ”Solution” related webpages. It has to be given a special attention to the ”Non-solution” samples as well. Although one can see that these two classes are different, they can be easily mixed up by a non expert on the area. Some of the following sections will describe how this was taken into consideration during the system development.

3.2 Collecting data

In order to train and validate a custom classifier that identifies user innovative health related solutions amongst the large variety of topics that the Web addresses, we need not only samples from these solutions but also a set of samples that represents well the Web's diversity.

3.2.1 Solutions and Non-solutions

Two kinds of samples of patient-driven solution candidates were gathered, the treated samples and the original samples.

The treated samples were the posts of the Patient-Innovation platform. These were created by the analysts from the Patient-Innovation project. The data was filtered and exported to a raw format, to be further cleaned and used by other developed programs and scripts. The title and content of each post was retrieved. The total amount of available post samples was 914, 763 of them being considered ”Solutions” and 151 ”Non-Solutions”.

The original samples were the original content from the posts of the Patient-Innovation platform. Most posts in the platform have the URL of the webpage where the content was taken from. It is located in its content, following the expression ”Adapted from:”. These URLs were extracted and manually followed to extract the original URL, because most of them were encrypted. After, using Nutch, the URLs were crawled. Using them as seeds, and by executing a crawl of depth 1, all the content pointed by the URLs was fetched and parsed. For parsing, a filter called *boilerpipe*[1] was used. This filter is integrated into Nutch and provides algorithms to detect and remove non-relevant content (like header, footers, advertisement and references to other parts of the website) around the main textual content of a web

[1] https://code.google.com/archive/p/boilerpipe/

page, which is called boilerplate. It is far from being 100% accurate, but it is the best approach to avoid collecting uninformative and repetitive vocabulary, which would be considered as noise in the dataset, spoiling the text classifier training and validation.

These two datasets have their advantages and disadvantages as representatives of ”Post” samples. The first one has the advantage of having data that does not have noisy (boilerplate) content that webpages usually have, even when parsed with the boilerpipe filter. However, as this data was treated by analysts, it lacks the original syntactic and semantic content, and each post writing style may be biased by the analyst who wrote it. In contrast, the original content preserves the syntactic and semantic content of the targeted information. However it comes with boilerplate. In spite of that, it puts it in the same circumstances as the DMOZ data, which will be described afterwards.

Because not every post had a correspondent URL, the information in the URL's webpage was not in english, the webpage was not found or it had been moved, it was not possible to recover the original content for every post. Therefore, for these samples, the treated posts were used instead of discarding them. In the end, it was only possible to successfully fetch the original content of 443 of the 914 posts.

3.2.2 Other topics

In order to obtain a good set of examples that expressed the variety of content on the Web, an archive of websites called DMOZ[2] was used. This archive contains approximately 4 billion references to external websites that are human labeled in the directory. It is the largest human edited directory of the Web. It contains references to websites about all kinds of topics like Sports, Arts, Health, Business, Science and more, so it can be considered a good sample of the Internet. At the time of writing, this archive is no longer maintained, as of March 2017, which was not long before the time of the development of this Master thesis, so the samples in it could still be considered as good representatives of the current state of the Web. As the references were labeled by a group of chosen editors, the labels can be considered trustworthy for the guidance of our data collection.

DMOZ periodically offers dumps of its database, in the RDF format. The latest version was obtained and a subset of some of the available categories was sampled, in order to obtain a variety of labeled URLs. The content pointed by these URLs was obtained following the same procedure as the one used to obtain the original content from the posts. This is the reason why the samples from the original posts dataset are said to be in the same circumstances as the data retrieved from DMOZ.

Samples of Arts, News, Society, Recreation, Business, Sports, Science, Computer and Health categories were acquired. These were the categories thought to be more informative, and that would give a better picture of the Internet in general. The majority of the samples about Health were examined, to make sure no sample could be a representative of a ”Solution” or a ”Non-Solution”. No such samples were found. In the end, 808 Health samples and 3073 samples of topics other than ”Post” and Health representatives were gathered. Because this latter set represents a much larger domain than the ”Post” set, it was decided that it would have around three times as many samples. Although the first one represents a world much bigger than the second, this decision was made to avoid a highly unbalanced dataset. Also it was decided to separate the topic of Health from the others, as it is of major importance to identify health-related samples, even if they are not ”Post” samples.

In the end, two major datasets were created: the *Treated Dataset*, comprising the treated posts of the Patient-Innovation platform and the samples from DMOZ, and the *Original Dataset*, which contained the original content from the posts, the treated posts whose original content was impossible to collect and the samples from DMOZ.

[2]http://dmoz-odp.org/

3.3 Data cleaning and pre-processing

In order to clean the treated post samples, they were visualized to identify their basic structure and patterns. A repetition pattern of the expressions ”More info”, ”Adapted from” and ”Watch the video” was noticed at the end of the samples (examples may be seen in table 3.1), followed by hyperlinks and other non-informative expressions. These patterns were identified and they were removed along with the text following them. All the URLs were removed as well, as they present noisy syntactical information.

Examples
”... regardless of cancer status, Sirlin said. Adapted from: http://usat.ly/1MT6lVT More info: http://www.cwellness.com/ What about you, do you have any solutions? Please share them with the Patient Innovation community! https://www.youtube.com/watch?v=A-sM_0uaHcE”
”... move independently, it can react to uneven terrain. More info: http://www.oandp.org/jpo/library/2010_02_113.asp Adapted from: http://bit.ly/2hatoRN What about you, do you have any solutions? Please share them with the Patient Innovation community!”
”... campaign launched at the end of May. Watch the video: http://www.9news.com/story/news/local/ 2015/06/30/wheelchair-invention-inspired-by-personal-experience/29542425/ More info: www.quadshox.com Adapted from: http://www.metierlaw.com/colorado-state-university-graduate-invents-new-wheelchair”

Table 3.1: Examples of data patterns that needed cleaning.

In the ”Non-solution” post examples, the ”Unpublished solutions” presented a problem and some samples had to be filtered out. A count of every type of ”Unpublished Solution” is present on table 3.2. Referring to this table, posts with unpublished reason numbers 1, 4 and 6 were removed, because they were duplicates from positive examples, there was missing information or they were still under medical evaluation. These reasons cause these samples to be dubious about to which class, ”Solution” or ”Non-solution”, they should belong. Therefore, their removal should facilitate the contrast between these two classes.

Id	Unpublished reason	Count
1	Duplicate from a solution	30
2	Not a user innovation	23
3	Not developed by a patient, caregiver or collaborator	22
4	Missing information	17
5	Dangerous	6
6	Under medical evaluation	5
7	Not a solution	3

Table 3.2: Reasons why solutions were not published and their count.

The set about other topics was checked for duplicates, which could happen when two different URLs point to the same content, or when websites have a standard way of informing that an error as occurred, without being a fetching error (otherwise Nutch would notice it), for example, ”This site requires JavaScript and Cookies to be enabled. Please change your browser settings or upgrade your browser.” appeared in more than one parsed sample.

The posts and the samples of other topics were gathered. With the aid of the *OpenNLP* [3] library from Apache, all the samples have gone through the following steps (in the numbered order):

1. Samples from languages other than English were removed.
2. URLs removal.
3. Normalization consisting of lower casing, number removal and punctuation removal.
4. Tokenization.

[3]https://opennlp.apache.org/

5. Stopwords removal.
6. Removal of words consisting of less than 3 characters.
7. Removal of samples with less than 4 tokens.
8. Lemmatization/Stemming.

We want to remove samples from languages other than English because we are focused on building a text classifier which identifies patient-driven solution written in English, and we noticed that DMOZ and some original samples were written in other languages like Chinese, Spanish and French, which would be considered as noise in the training data. It is important to notice that the OpenNLP language detector identifies a sample as English if the majority of its terms are in English, even though some terms, e.g. medical terms, might not be recognized by their model. So, we can assume that these terms are never lost, and they can be important for domain specific classification.

Four different dataset configurations consisting of the same samples were kept: a stemmed one, a lemmatized one, one with full syntax of the words and one with full syntax and stopwords. These were used to test which dataset configuration performed better on the tested classifiers.

The number of samples of each class that were lost during the cleaning process can be seen on table 3.3

	Before cleaning	**After cleaning**
Solution	763	759
Non-Solution	151	98
Health	808	680
Other	3073	2607

Table 3.3: Dataset before and after the cleaning process.

It can be seen on table 3.3 that the dataset is rather unbalanced, specially when considering a subset containing the classes of ”Solution” and ”Non-solution”, which is used to train the second layer of our classifier. Ideally, the ”Non-solution” samples should be around two times as many as the ”Solution” samples. It is common practice to represent the negative class with at least double the samples there are on the positive one. It can be considered that some negative examples are already captured on the first layer of the classifier. Nonetheless, this high degree of unbalancing showed problems during evaluation, explained on Sections 5.1 and 5.2.

Chapter 4

System Development

This chapter presents the system architecture, the proposed approaches for each component of the system and how each component was integrated on the used open-source crawler.

4.1 System architecture

This section presents the system architecture. It starts with the presentation of a high-level picture of the overall system architecture, explaining the inter-process communications between components. Then, each main component is described, explaining their functionalities and processes.

4.1.1 Main components, tasks and interactions

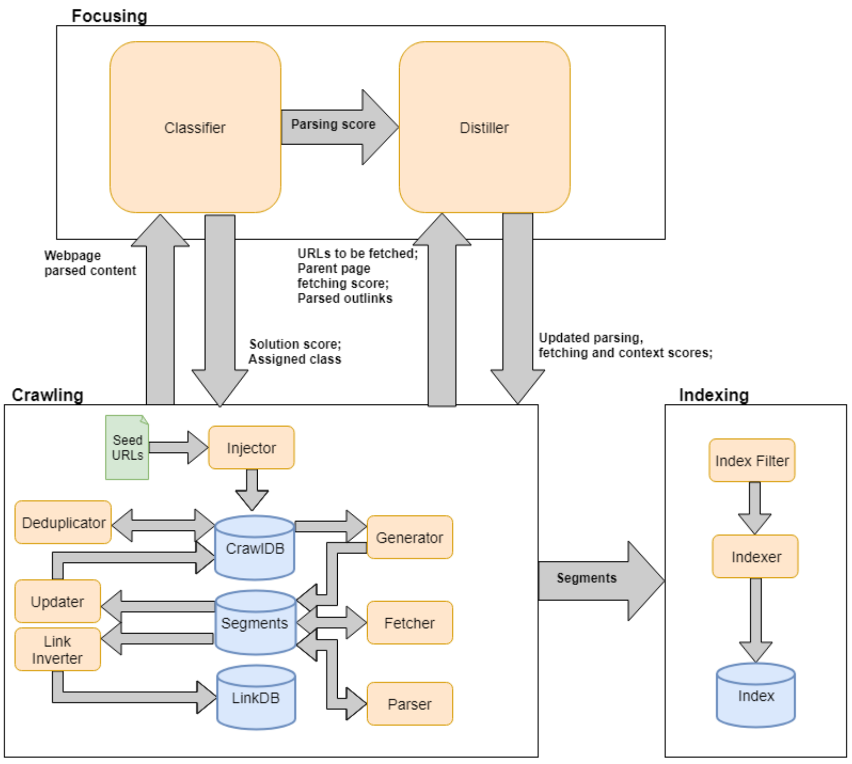

Figure 4.1: Architecture of the proposed system.

We refer to the proposed crawler as 2Gather4Health (2G4H) crawler. The goal of the proposed system is to crawl for specific information, patient-driven solutions, while indexing it to a web indexer. We can think of this goal as the fulfillment of three tasks: a *crawling* task, a *focusing* task and an *indexing* task. With that in mind, this system's architecture can be explained by describing each task components and interactions. A high level picture of this system's architecture can be seen in Figure 4.1, in which the mentioned tasks are depicted as well as its interactions, which will be explained on the following paragraphs.

The crawling task is the backbone of this system. It is the task that takes care of all the basic operations occurring during a crawl. That includes generating fetch lists, fetching URLs, parsing webpages, removing duplicate URLs, storing/updating parsed and link information, amongst other things. In this system, these operations are all executed by the core of Nutch, the open-source crawler opted for this system. Its core processes are explained on Section 2.2.6 and all process interactions are depicted in Figure 4.1.

The focusing task is in charge of target specification. In this case, this task focuses on identifying and following patient-driven solution webpages. Its components, the *classifier* and the *distiller*, are the components specifically made for this system. The classifier takes the parsed information from webpages, which is obtained by the crawling task, and classifies them as patient-driven solutions or other categories. The class assigned to each URL influences its scores, the *parsing score* and the *solution score*. The solution score and the assigned class are stored in the crawling segment, so they can be used for indexing. The distiller is an URL classifier. It takes the outlinks parsed from webpages and assigns them a score (the *fetching* score) which will be used to rank the fetching process of newly discovered URLs. This score depends on the parsing score calculated in the classifier, on previous fetching scores, that are kept on the CrawlDB and on the information parsed from the outlinks (anchor text and URL). Additionally, the distiller controls which URLs are re-fetched, for information updating purposes, and what score URLs use when setting the fetching order. The distiller outputs the updated parsing, fetching and context scores, which are explained on the following sections. Figure 4.1 pictures the described interactions between the focusing and crawling tasks.

The indexing task takes care of permanently storing and indexing the relevant parsed content, patient-driven solution candidates. The main purpose of this task is to make the relevant information accessible and easily searchable. In Nutch, the parsed information is stored in segments, one segment per iteration. At the end of each iteration, the indexer takes the segment and filters the information to store and index, exempting the crawler from having to store the segment. Figure 4.1 portrays the described indexing processes, as well as its interaction with the crawling task.

4.1.2 Classifier

As one of the main components of the system, the classifier's goal is to categorize crawled webpages and calculate the *parsing score* of each parsed URL. This score symbolizes the probability of an URL belonging to the assigned class. Additionally, it calculates the *solution score*, a score that represents the degree of resemblance to a patient-driven solution.

This system's classifier is a text classifier, which has two layers with different goals. In the first layer, the classifier has to separate 3 types of information. Information about a wide variety of topics like Sports, Science, Arts, Business, etc. are all aggregated in one class, called "Other". This class should represent all the information that there exists on the Web which is not health-related. Health-related information is divided into two classes in this first layer. A class "Post", represents all kind of information that one can encounter in the Patient-Innovation platform database (this information is described in Section 3.1). This information is all health-related, and contains the target information. The class "Health" should represent the other health-related information available on the Web. All the webpages classified as ”Post” advance to the second layer, while the other labeled samples are discarded and just used for scoring purposes. In its second layer, the classifier’s goal is to separate between patient-driven solutions and all the other similar but false solution information. The first class is called ”Solution” and the second one is called ”Non-solution”, respectively. These classes are representatives of the information presented in Section 3.1, presented with the same names. Figure 4.2 depicts the architecture described.

A detailed explanation of this architectural decision is given on Section 4.2.1.

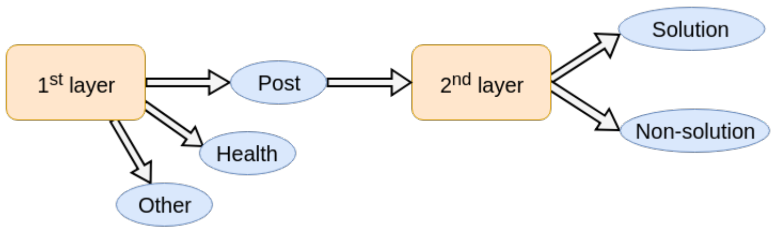

Figure 4.2: Architecture of the proposed classifier.

In its final configuration, the first layer is a Multinomial Naive Bayes text classifier, trained with the full dataset described in Chapter 3, using unigrams features with TF-IDF weights. While the second layer is a Fingerprint Classifier, trained with a subset consisting of just the ”Solution” and ”Non-Solution” labeled samples. This uses bigrams as features with TF weighting. This final configuration was a result of the performance evaluation described on Section 5.1.

In addition to classifying a webpage, the classifier also sets its parsing and solution scores. Both are taken from the probability distributions obtained in each layer. The parsing score is the probability estimation of the classified webpage belonging to the class that it was assigned to. The solution score is the probability estimation of the classified webpage belonging to the ”Solution” class, which is the similarity score, defined in the FFP-C approach, between the classified webpage’s fingerprint and the ”Solution” class fingerprint.

4.1.3 Distiller

The distiller sets the fetching order of the URLs. Its goal is to prioritize the fetching of relevant URLs, and when dealing with newly discovered URLs, without knowing the content of the webpages they point to. The fetching order is sorted by one of two scores. The score considered for each URL depends if it was already fetched, thus it is being re-fetched for update purposes, or if it is a newly discovered URL. In the first case its solution score is used, while in the second case it is used its *fetching score*. If a page is classified as ”Health” or ”Other”, its solution score is considered to be zero, so these pages are never re-fetched.

In order to define the degree of relevancy of a newly discovered URL, its fetching score has three components: the *parent fetching score*, the *parent parsing score* and the *context score*.

The parent fetching score and the parent parsing score are the fetching and (part of) the parsing scores of the parent page, respectively. The parent page of an URL is the page where the URL was found. If an URL has more than one parent, their fetching and parsing scores are averaged to produce the final parent fetching and parsing scores, respectively.

The parent parsing score is determined by the parsing score and assigned class that the classifier gave to the parent webpage. If the page was classified as ”Post”, the full parsing score of the parent page is used, if it was classified as ”Health”, only half of the score is used, if otherwise it was classified as ”Other”, the parent parsing score is considered zero.

The context score is the similarity score of the set of terms extracted from the URL anchor text in the parent page and from the URL itself, with the set of terms extracted from the Patient-Innovation post titles. The similarity function used is the cosine similarity. The set of terms comprised in the URL are obtained by splitting the URL path component (excluding the hostname and protocol) in all dots, hyphens, underscores and slashes. The terms obtained are pre-processed (normalization, stopwords removal, stemming) as are the titles they are compared to. A URL can have more than one anchor texts, from the same or more than one parent page. In that case, the scores are averaged to form a single context score.

The final fetching score is a linear combination of the three score components described. Their corresponding weights can be manually set in the Nutch's configuration file, the *nutch-site.xml*.

The reasons why it was followed this approach are explained in Section 4.2.2.

Figure 4.3 depicts the processes undertaken to calculate the fetching score, and it is illustrated with an example.

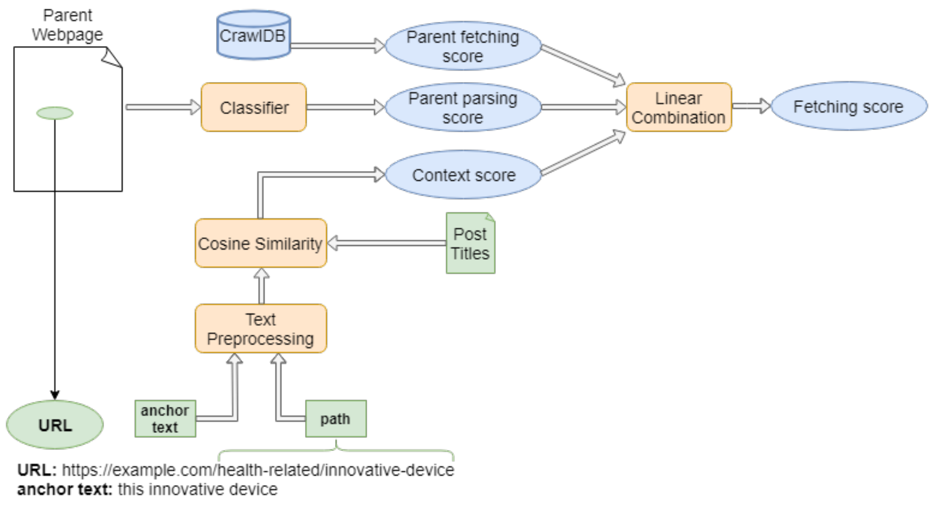

Figure 4.3: Distiller processes and example.

4.1.4 Web indexer

As it was stated before, the web indexer used in this system is Solr. It provides one of the fastest open-source web indexes, additionally coming with a web administration user interface which provides tools for managing, debugging and search through its index. Nutch already comes with integrated Solr interaction, so it was just needed some configuration.

At the end of each crawl iteration, after the Deduplication process, the segment created for that iteration holds all the information collected about the crawled URLs. This includes parsed text, outlinks, last fetched date, the class and the score assigned to them and other metadata. A this point, selected items from this information are stored and indexed in Solr. Each indexing process is followed by a cleaning process. This takes care of removing documents from the index whose URLs were marked as gone or duplicates in the crawl database. If a certain webpage is re-fetched, its information is re-indexed in Solr, overwriting its old index entry.

Choosing the fields to be part of the index is essential. On one hand, one would want to store as much information about a webpage as possible, on the other, it would be better to maintain a lightweight index, just with the essential information, so it can perform faster. Concerning this trade-off, besides some core fields that need to be present, it was decided to store the following fields: URL, title, content, anchor texts, last fetched date, solution score and assigned class. Information belonging to the first 4 fields goes through processes of tokenization, stopword removal, lower casing and stemming during the indexing and search querying processes. This helps maintaining a more lightweight index and improves result matching when searching through the index. The last fetch date can be useful to know the freshness of the document stored. The purpose of storing the solution score is to provide a sorting of the results by their similarity to patient-driven solutions. This provides a degree of importance to the results, that users (the analysts of Patient Innovation) can utilize to prioritize the analysis of webpages. Additionally, it is a way of sorting the results without needing a word/phrase query. Therefore, users can search through results by their importance, without having to focus on a specific topic, keyword or query. The assigned class is stored for users to know to which class the webpages were automatically assigned. It was decided to only index documents classified as ”Solution” and ”Non-solution”, therefore, those are the only options for this field. The reason why it was done so is because there is still an unclear automatic separation between these two classes. So, their samples could serve as training data to re-train the classifier, improving its performance. This is in fact shown in Section 5.2.1. Despite of that, users can chose to only search for ”Solution” samples on the index. Lastly, there is a combined field which aggregates all text fields (URL, title, content and anchor texts) to provide a default field for text search.

4.2 Proposed approaches

This section explains all the approaches tried when developing each component of the system. It describes them and presents the reasons why they were accepted or discarded in the final system configuration.

4.2.1 Classifier

The classifier is the most important component in this system, as it is responsible for the identification of patient-driven solutions. Therefore, different approaches were tried in order to achieve the best classification performance.

A standard single-layer classification approach was tried, but it was soon realized that a more complex method was needed. A double-layer classification approach proved to be more effective, but there were still challenges during its second-layer classification. Our focus laid on this layer, to provide a meaningful separation of the ”Solution” and ”Non-solution” categories.

The following sections describe all the different approaches tried during the classifier development.

Single-layer approach

The single-layer approach consisted of separating the samples into three classes: ”Solution”, ”Non-solution” and ”Other”. This last class represented all the information on the Web that did not concern Patient-Innovation, including other health concerning issues. While the ”Solution” and ”Other” classes showed to be separable, most of the ”Non-solution” class representatives were being classified as ”Solution”, for all the different classifiers and configurations tested. This happened because the ”Non-solution” class was the minority class, with a ratio of at least $1:7$ compared to the other classes, and samples of this class could be easily mistaken as belonging to the class ”Solution”.

Soon it was realized that a different approach was needed. By observation of the raw text of the ”Solution” and ”Non-solution” samples, and by using Fuzzy Fingerprints (FFP) as a result analyzer, it was noticed that their vocabulary was similar, and only a few amount of words differentiated those classes. Therefore, it emerged the idea of having a dedicated layer which was in charge of separating these two problematic classes. This way, customized methods could be applied to try to separate these classes, without having to concern about a third class separation.

Double-layer approaches

With this double-layered architecture, we reduce the amount of dimensions (classes) of each layer, as well as the amount of training data needed in each layer, because the second layer acts on a narrower domain, set by the first layer. Furthermore, merging ”Solution” and ”Non-Solution” samples in one class helps the dataset of the first layer to be more balanced. Additionally, even if the separation between these two classes is not successfully achieved, the system still narrows down the webpages crawled to ”Post” samples just by performing a successful separation in the first layer, which is intuitively easier. The downside of this approach is that relevant data can be lost in the first layer and samples that are not ”Solutions” neither ”Non-Solutions” can reach the second layer. The more the layers a classifier has, the more it can lose information and accumulate error.

Two different double-layered configurations were tried. In the first one, the first layer classified samples between ”Post” and ”Other” and the second layer between ”Solution” and ”Non-solution”. In the second approach, the first layer should classify samples as ”Post”, ”Health” and ”Other”, while the second layer remained the same.

In the first configuration, the first layer's goal was to separate all the information related to Patient Innovation from all other information that represented the rest of the Web. This way, it could be achieved a very straightforward separation by having the first layer classifying samples as positives or negatives, regarding Patient Innovation information. This decision was made because many classifiers perform better when dealing with binary classification problems, and by that time, there was no need for information regarding health-related topics other than the ones covered on Patient Innovation.

This configuration proved to have the best performance on the first layer classification, reaching weighted average values of accuracy, precision, recall and F-measure of above 98% during validation. However, later, due to advances made in the development of the distiller, explained in Section 4.2.2, there came the need for inclusion of a ”Health” representative class in the first layer. This resulted in a $5\% - 6\%$ drop in overall performance on the first layer. In spite of that, the performance drop was observed to be the result of missclassifications between classes ”Health” and ”Other”, leaving the performance values related to the class ”Post” unaltered. Therefore, it was justified to move to a three class first-layer.

The previous approaches led to the second and final configuration of this double-layer approach. In this approach, the classifier is supposed to differentiate samples between the classes ”Solution”, ”Non-Solution”, ”Health” and ”Other”. While the reason why we want to identify samples as ”Solutions” is obvious, the reason why we divide the non-relevant samples between three classes is not so obvious. The motivation behind it is that ”Non-Solutions” are very similar to ”Solutions”, so at a first stage, merging them in one class (”Post”) will help differentiating them from other topics. Also in the first layer, the crawler can benefit from having a ”Health” class because ”Post” samples are also health-related, so knowing that a webpage is not a ”Post” sample but it is health related can be useful to help the crawler stay on the health topic. Finally, classifying samples as ”Other” helps the crawler identify webpages that do not present relevant topical information.

Recapitulating, in the final configuration, the first layer of the classifier categorizes samples between the classes ”Post”, ”Health” and ”Other”, while the second layer categorizes samples between the classes ”Solution” and ”Non-Solution”. Only samples categorized as ”Post” move on to the second layer.

The performance of the final configuration of the classifier can be seen on Sections 5.1 and 5.2, which show and discuss its results during validation and testing, respectively.

Separation in the second-layer

It is important to remind that, in the cleaned dataset, ”Non-solutions” are ideas or advice for solutions or potential solutions that got rejected because they were of commercial intent, they were physically intrusive, not developed by the users or not innovative, so it is of no surprise that they share very similar vocabulary and writing style with ”Solution” samples. Furthermore, the lack of samples of the negative class hinders this class from standing out, i.e., having characteristic features that represent it well.

In order to try to surpass this problem, successfully separating the ”Solution” and ”Non-solution” classes, three approaches were tried, using the Fingerprint Classifier as the others proved to be ineffective, as showed in Section 5.1. Furthermore, having a custom implementation of this classifier provided better control of its behavior.

In the first attempt, with the help of Patient-Innovation analysts, we tried having a list of exemplar ”Non-Solution” terms, terms thought to be very frequent and representative of this class. This would be terms like ”engineer”, ”doctor”, ”professional”, ”diet”, ”food”, ”drug”, that should be representatives of the negative class. The idea was to boost this terms in the fingerprint of ”Non-Solution”, so that the similarity score between ”Non-solution” samples and its class could present higher values. Ideally, this would make ”Non-Solution” candidates being wrongly classified as ”Solution” change their class. However, this proved to be ineffective.

The approach to collect the original information regarding the posts in the Patient Innovation platform, conceiving the Original Dataset, was also a way to try to separate these two problematic classes, in addition to the other reasons mentioned in Section 3.2. By having the posts original content, it was expected that they would not be contaminated with the writing style of the analysts who wrote the posts for the Patient Innovation platform. Thus, we hopped that the vocabulary similarity between the ”Solution” and ”Non-solution” classes would fade away. Unfortunately, it was only possible to get the original content of around half the posts. Still, results presented in Section 5.1 show that this approach only led to a worst performance.

In the final approach, several transformations of the dataset using several combinations of n-grams were tried. Having a dataset of bigrams with TF weights proved to achieve the best performance. With the threshold set to zero we noticed that it achieved a performance similar to the methods already tried, and almost all samples were being classified as ”Solution”. However, after trial and error, a threshold was set that separated both classes, with very good performance. In this case, we noticed that most ”Non-Solution” samples were being classified as so because of the threshold, and not because their similarity score with the negative class was higher than with the positive class. This changes the essence of the classifier, as now it can be seen as a relevancy classifier, that classifies a sample as non-relevant if its score is below some threshold. Nonetheless, this approach was the only one that proved to be effective, as it can be seen in Section 5.1.

4.2.2 Distiller

The distiller is the component that makes this system a focused crawler. It is the component that controls the system’s efficiency, because it is the one that leads the crawler to relevant results. Therefore, it is of major importance that the distiller guides the crawler onto a path that has as few irrelevant pages as possible.

Two different distiller implementations were attempted. The first one, the *comprehensive approach*, was focused on crawling the Web in a manner that embraced as many patient-driven solution webpages as possible. However, this proved to be very inefficient, as it was noticed that too many irrelevant pages were being crawled. In the second approach, the *multi-score approach*, there was evidence of a balance between efficiency and effectiveness. In this one, less irrelevant pages were being crawled. Nonetheless, these pages were not being completely discarded, because they could be of some value to detect webpages regarding patient-driven solutions.

Comprehensive approach

As mentioned above, this approach consisted of finding as much patient-driven solutions as possible. In order to do it, three techniques were followed.

First, every page classified as ”Post” is kept and all outlinks from those pages are scored with the parsing score of the last page where the URL was found. This is a common basic approach that prioritizes the newly discovered URLs based only on their parent page's content. Then, two other measures are done in order to keep some outlinks of pages not classified as ”Post”, but that could lead to solutions. One of these consists of keeping all the links belonging to the websites of the seed URLs. The idea behind this is that all the seed URLs chosen are important to the target topic, and often their websites contain more than one solution webpage. The other measure is to keep all the links whose anchor text or URL pathname contained at least one of a list of pre-defined terms. This list is synthesized from all the tags presented on the Patient Innovation platform posts. These tags are keywords/topics that highlight the themes of the posts. These can range from diseases and recovery methods to actions and body parts. So, in order to avoid keeping the list too broad, it is filtered down to keep only the descriptive terms. The score given to all the URLs which came from these two measures is the probability estimation of their parent page belonging to the class ”Post”. These scores are obviously lower than the ones given to the URLs which came from ”Post” pages, as pages from these two measures are not classified as ”Post”. All URLs whose parent webpages are not classified as ”Post” and that do not fulfill the other two measures are discarded.

After some trial crawls, soon it was realized that there were too many irrelevant pages on the crawled sets. The problems with this approach were evident. On one hand, giving the same score to all the URLs found in a ”Post” page is too simplistic. Most of the times, there are a lot of URLs pointing to irrelevant information like advertisements, online shops, social networks or other pages not directly related to the crawled page. So, there is the need to give a distinctive score to URLs within the same page. Furthermore, all the URLs that came from ”Post” pages were being crawled first than all the other kept URLs. This was not a good thing, as it was noticed that most solution webpages are not directly connected to other solution webpages. On the other hand, keeping all URLs from seeds' websites and whose anchor and pathname texts contained at least one term from the synthesized tags' list was including much more irrelevant URLs than relevant ones.

It was clear that this distiller implementation was too broad. A more complex model was needed. A model that would give priority to URLs on the target topic, but that could be more selective, although without discarding irrelevant pages that could led to relevant ones. After studying state-of-the-art approaches, it emerged the idea of the multi-score approach.

Multi-score approach

This approach is the one used on the final version of this thesis' system and its implementation is explained on Section 4.1.3. To summarize, the score given to newly discovered URLs is a linear combination of three score components: the parent fetching score, the parent parsing score and the context score. The fist two scores involve the URL's parent pages' score and content, respectively, and the last component involves the URL's pathname and anchor texts on the parent pages.

There were several reasons why it was decided to have these three components on a URL's fetching score. The parent fetching score makes the score of each URL spread to its children, meaning that if an URL is considered relevant (thus it should have a high score), there is a probability that some of its children are relevant too. Also, although it was noticed that, most of the times, webpages presenting ”Solutions” are not directly connected to other ”Solution” webpages, they might be indirectly connected, through one or a few more pages. Therefore, we want a ”Solution” page to influence the score of not only its children pages, but also of its higher degree descendants. The parent parsing score makes relevant pages have a higher influence on the final score of the unfetched URLs. The more a page is similar to the target content, the more it can influence it, otherwise, it has no influence at all (if it is considered as ”Other”). This favors the premise that pages about the same topic are connected. For the context score, instead of using the tags of Patient Innovation, it was noticed that a lot of the URLs of the original content of the posts contained the post title or a similar sentence in the URL path. For example, URLs *https://www.worcsacute.nhs.uk/news-and-media/729-patient-creates-personalised-covers-to-hide-lines-used-for-chemotherapy-treatment* and *https://eu.upstateparent.com/story/life/2018/04/06/kidney-patient-creates-silly-kidneys-boost-patients-morale/33571991/* contain the exact post title within their URL. Thus, we also believe that a similar sentence could also appear in its anchor text. Therefore, the similarity score presented could represent the level of similarity between an URL and the target information. This similarity score technique seemed also more trustworthy than the binary approach of word presence, used in the comprehensive approach. One thing to have in mind is that, in this approach, no URL is totally discarded. This means that the system will keep even the most irrelevant URLs. However, most probably these are never fetched because their scores are significantly lower than relevant, or at least health concerning, URLs. Additionally, a threshold can be set for the fetching score above which URLs will not be fetched. But experiments to properly set this threshold were not attempted, hence it was set to zero.

While this combination of scores is an original approach, it was inspired by other common approaches seen on researches. For example in [75] they use the parent page, along with other pages following the family analogy, to predict the relevancy score of a target page to a topic. The parent parsing score and the context score are both methods that use the surrounding text of an URL to predict the URL relevancy. In the first one the surrounding text is the full text of the page while in the second one it is just the anchor text. That technique is a very popular content based approach used for URL topic prediction, and it can be seen in, for example, [48], [45], [73] and [32]. The only difference is that , in this system, different methods are used for when the surrounding text is the full page and when it is just the anchor text.

4.3 Integration of the components into Nutch

In order to integrate our custom classifier and distiller into the Nutch crawler, a plugin that extends the *ScoringFilter* interface[1] was developed. This interface lets us manipulate the score of the URLs on each crawling phase (Injection, Generation, before and after Parsing, DBUpdate, Indexing), while having access to parsed and linkage (inlinks, outlinks) information. Nutch plugin functions act on a single URL at a time.

The classifier implements a function that runs after the parsing phase (*passScoreAfterParsing*). This function has access to an URL's parsed text. In its implementation, we included an instance of our custom classifier implementation for WEKA, to evaluate the parsed information, assigning a class and calculating the parsing and solution scores of the URL under analysis.

[1] http://nutch.apache.org/apidocs/apidocs-1.12/org/apache/nutch/scoring/ScoringFilter.html

The distiller implements the functions *generatorSortValue*, *distributeScoreToOutLinks* and *updateDbScore*. Our implementation of *generatorSortValue* chooses which score to use to set the fetching order, the solution score or the fetching score, depending if the URL under analysis was already fetched or not. The second function is implemented to calculate a partial fetching score of each outlink parsed from the URL's webpage, calculating its three components (parent parsing score, parent fetching score, context score). At this time, each URL has a partial fetching score for each outlink found during this iteration with the same associated URL. In order to properly calculate the final fetching score of an URL, our implementation of *updateDbScore* keeps the sum of all the partial fetching scores associated with the URL under analysis, from the current and previous iterations, and the number of different inlinks that the URL has. With that information we can calculate the final fetching score, which is the average of all the partial fetching scores associated with the URL under analysis.

The plugin can be run when added to the Nutch configuration file, called *nutch-site.xml*. Due to Nutch's batch processing style, the plugin is dynamically loaded everytime Nutch needs to run a function it implements. Although this form of processing is what offers the high level of modularity and extensibility to Nutch, it has some downsides. The greatest disadvantage of this style is the fact that a plugin has to be loaded and set up every time it starts its execution. This may involve loading heavy files into memory and doing extensive computations, which could be done only once if Nutch implemented a single process style. This fact hinders our crawler efficiency. Every time the programs moves to the post-parsing phase, when the classification occurs, the plugin is activated and the classifier has to be trained, which could be done only once if the plugin was always executing during the entire crawl.

The Solr indexer is already integrated into Nutch, so the only need was to make a plugin, which extends the *IndexFilter* interface[2], in order to manipulate the fields and pages that are meant to be added to the index. This plugin only implements one function, *filter*. Our implementation of it filters out all the documents classified as ”Health” or ”Other”, so they do not appear in the index. Also there, we add the fields of ”assigned_class” and ”solution_score” to the index, with the values associated with the document being indexed.

[2]https://nutch.apache.org/apidocs/apidocs-1.12/org/apache/nutch/indexer/IndexingFilter.html

Chapter 5

Experiments and Evaluation

This chapter presents the experiments done to evaluate the performance of the crawler's classifier and of the crawler itself. Along with each experiment there is a discussion of the obtained results.

The text classifier was tested on the dataset described on Section 3 and on sets of data crawled by the proposed crawler. The results of those tests are presented on this chapter, comparing different configurations on metrics of accuracy, precision, recall and f-measure. Lastly, the proposed crawler was tested on metrics of harvest rate and target recall, and compared with a standard broad crawling approach and different configurations of the proposed crawler.

In the end of this chapter, there is a section discussing important points of the evaluation, connecting the dataset used with the obtained results and proposing related strategies to improve the crawler's performance.

5.1 Classifier validation

To evaluate the performance of different classifiers and configurations over both the Original Dataset and the Treated Dataset, the *WEKA* [76] framework was used. It consists of an open-source software containing a framework for machine learning and data mining tasks. These include tools for data preparation, classification, regression, clustering, association rules mining, and visualization.

To extract the features from the datasets, WEKA's *StringToWordVector* filter was used. It can extract TF-IDF values form textual data, keeping a limited number of features per class (default to 1000, meaning only the 1000 features with the highest TF-IDF values are kept, for each class) and it also has the ability to perform stopword removal and stemming, although these were not used. After filtering, each of the Original and Treated datasets were transformed into a dataset featuring unigrams with TF-IDF values and a dataset featuring bigrams with TF values. An additional dataset was created for each major dataset. This dataset contained Word2Vec features, after performing this type of word embedding to the samples of the Original and the Treated dataset. The feature values were obtained by the procedure explained in 2.1, explaining the merging approach to transform a document into Word2Vec features. The pre-trained model called *GoogleNewsSLIM*[1] was used as our Word2Vec model, since the model it came from, the popular *GoogleNews*[2], was too heavy for the computer used in the development of this thesis. However, the probability of losing information is very low, since this slimmed down model kept all the terms that cross-referenced with English dictionaries and Urban Dictionary, a crowd-sourced online dictionary for slang words and phrases, meaning the large majority of English terms are represented in

[1]Available at: https://github.com/eyaler/word2vec-slim
[2]Available at: https://code.google.com/archive/p/word2vec/

this slim model. Additionally, this model loaded much faster, making the crawling system more efficient when using it. To load the model, an open source Scala implementation was used[3].

Several types of text classifiers and configurations were tried to train and validate the two-layered classifier. These include ZeroR, Multinomial Naive Bayes, Logistic Regression, SVMs and the novel approach of FFP-C. While WEKA offers an implementation of the first classifiers, an implementation of the last was developed together with the corresponding wrapper for WEKA, so all classifiers could be compared using the same evaluation framework. The layers were trained independently.

Classifier	**TPR (Post)**	**FPR (Post)**	**Precision**	**Recall**	**F-measure**
ZeroR	-	-	62.9 %	62.9 %	62.9 %
Multinomial Naive Bayes	93.7 %	1.9 %	91.0 %	91.0 %	91.0 %
Logistic Regression	91.9 %	0.7 %	90.6 %	90.8 %	90.4 %
Logistic Regression w/ Word2Vec	**93.8 %**	**1.6 %**	**92.4 %**	**92.6 %**	**92.4 %**
SVM	91.4 %	0.8 %	90.3 %	90.4 %	90.0 %
SVM w/ Word2Vec	**94.4 %**	**1.9 %**	**92.3 %**	**92.4 %**	**92.3 %**
FFP-C	95.0 %	3.4 %	89.0 %	88.5 %	88.6 %

Table 5.1: First layer validation using the Original Dataset.

Classifier	**TPR (Post)**	**FPR (Post)**	**Precision**	**Recall**	**F-measure**
ZeroR	-	-	62.9 %	62.9 %	62.9 %
Multinomial Naive Bayes	95.0 %	1.1 %	92.0 %	91.9 %	92.0 %
Logistic Regression	94.2 %	0.5 %	91.3 %	91.4 %	90.9 %
Logistic Regression w/ Word2Vec	**96.6 %**	**1.2 %**	**93.4 %**	**93.5 %**	**93.4 %**
SVM	93.8 %	0.8 %	91.1 %	91.2 %	90.8 %
SVM w/ Word2Vec	**96.7 %**	**1.1 %**	**93.0 %**	**93.2 %**	**93.1 %**
FFP-C	95.3 %	1.7 %	89.9 %	89.2 %	89.4 %

Table 5.2: First layer validation using the Treated Dataset.

The ZeroR classifier was our baseline classifier. This approach classifies all the instances with the most frequent class. As the datasets are unbalanced for both layers, this puts a considerably high benchmark to improve on, specially on the second layer.

We began experimenting with the Treated Dataset. Initially, for every classifier, all configurations (lemmatized, stemmed, full syntax, full syntax plus stopwords) were tested, using unigram features with TF-IDF values with just the content (excluding the title) of the post samples. For all, the full syntax configuration performed better than the rest. Adding the titles of the posts to the dataset proved to lead to a better performance as well. The same was done with the Original Dataset, reaching to the same

[3]Available at: https://github.com/Refefer/word2vec-scala

conclusions. The final datasets configuration used full syntax of samples and post title and content. This configuration was used to obtain the derived datasets, using unigrams, bigrams and Word2Vec features.

For the first layer, it was performed a 10-folded cross-validation, while for the second layer it was 5-folded, as it was used a much smaller dataset.

The results for the first layer can be seen in tables 5.1 and 5.2, corresponding to using the Original Dataset and using the Treated Dataset, respectively. The precision, recall and F-measure are weighted averages of the three classes.

Classifier	**TPR (Solution)**	**FPR (Solution)**	**Precision (Solution)**	**Recall (Solution)**	**F-measure (Solution)**
ZeroR	100 %	100 %	88.6 %	100 %	93.9 %
Multinomial Naive Bayes	93.8 %	65.3 %	91.8 %	93.8 %	92.8 %
Logistic Regression	98.2 %	88.8 %	89.5 %	98.2 %	93.7 %
Logistic Regression w/ Word2Vec	95.7 %	83.7 %	89.9 %	95.7 %	92.7 %
SVM	96.2 %	84.7 %	89..9 %	96.2 %	92.9 %
SVM w/ Word2Vec	96.8 %	88.8 %	89.4 %	96.8 %	93.0 %
FFP-C	93.7 %	78.6 %	90.2 %	93.7 %	91.9 %
FFP-C w/ bigrams	**94.7 %**	**16.3 %**	**97.8 %**	**94.7 %**	**96.3 %**

Table 5.3: Second layer validation using the Original Dataset.

Classifier	**TPR (Solution)**	**FPR (Solution)**	**Precision (Solution)**	**Recall (Solution)**	**F-measure (Solution)**
ZeroR	100 %	100 %	88.6 %	100 %	93.9 %
Multinomial Naive Bayes	96.3 %	59.2 %	92.6 %	96.3 %	94.4 %
Logistic Regression	98.8 %	86.7 %	89.8 %	98.8 %	94.1 %
Logistic Regression w/ Word2Vec	98.3 %	88.8 %	89.6 %	98.3 %	93.7 %
SVM	98.4 %	86.7 %	89.8 %	98.4 %	93.9 %
SVM w/ Word2Vec	94.5 %	77.6 %	90.4 %	94.5 %	92.4 %
FFP-C	92.2 %	46.9 %	93.8 %	92.2 %	93.0 %
FFP-C w/ bigrams	**98.8 %**	**1.0 %**	**99.9 %**	**98.8 %**	**99.3 %**

Table 5.4: Second layer validation using the Treated Dataset.

In the first layer all classifiers prove to be effective, improving greatly over the baseline. The FFP-C is the only one with metric values below 90%, although still very high. Both classifiers using Word2Vec features show the best results. The Multinomial Naive Bayes is the one which performs better between the approaches that use unigrams with TF-IDF values. This is important to have in mind, as the tasks of preparing the training set and the samples to be classified are much faster when using the TF-IDF approach rather than the Word2Vec one. So, when crawling the web using this text classifier, one can chose to use the Multinomial Naive Bayes to achieve a more efficient and still effective crawl, as its validation results do not differ much from the Word2Vec approaches.

The results for the second layer can be seen in tables 5.3 and 5.4, corresponding to using the Original

Dataset and using the Treated Dataset, respectively. Every metric is with respect to the ”Solution” class.

As we can see in both tables 5.3 and 5.4, separating between ”Solutions” and ”Non-Solutions” in the second layer using the same approaches as in the first layer is not satisfactory, as all approaches present a high false positive rate. They also present a very high true positive rate. This means that the positive class (”Solution”) is absorbing the negative class (”Non-solution”). The large majority of samples are being classified as ”Solution”.

As mentioned in Section 4.2.1, the FFP-C using bigrams with TF weights and a set threshold was the only approach that proved to be effective on the separation of the positive and negative classes. This can be seen on tables 5.3 and 5.4, as this is the only approach that shows a low false positive rate, while presenting the highest values of precision, recall and F-measure.

For both first and second layers, using the Treated Dataset showed better results over using the Original Dataset.

5.1.1 Testing with layer dependency

In order to test the classifier's performance as a whole, a single test set has to be classified by both layers of the classifier, sequentially.

		Classified			
		Solution	Non-Solution	Health	Other
Actual	Solution	38	0	2	0
	Non-Solution	2	12	6	0
	Health	0	0	28	12
	Other	2	1	6	91

(a) Confusion matrix.

TPR / Recall (Solution)	FPR (Solution)	Precision (Solution)	F-Measure (Solution)	Recall	Precision	F-Measure
95.0 %	2.5 %	90.5 %	92.7 %	84.5 %	84.8 %	84.7 %

(b) Evaluation metrics.

Table 5.5: Results of the classifier on a test set, during validation.

As WEKA does not let us build and evaluate a two-layered classifier, its evaluation framework could not be used to test our custom classifier as a whole. Therefore, the validation dataset was divided into a training and a test set, and our text classifier implementation for Nutch was used to classify the latter, after training it with the training set. The classification results were then cross-checked with the true labels of the test set to produce the results on table 5.5. The first four measures regard the ”Solution” class, while the last three are weighted averages of all classes.

The text classifier was configured using the Multinomial Naive Bayes with unigram TF-IDF features for the first layer and the FFP-C with bigram TF values and set threshold approach for the second layer. On the first layer we decided to use the Multinomial Naive Bayes approach because, as mentioned, it is more efficient, in time and memory, than the Word2Vec approaches and its validation results were just slightly worse than these. So this will be the configuration used when crawling for solutions. It was decided to train the first layer with the Original Dataset and the second layer with the Treated Dataset. Although the Treated Dataset proved to have better validation results for both layers, in the first layer they do not differ much from the Original Dataset results and ideally we want to train the text classifier with data as close as possible to the crawled data (webpage content). However, for the second layer, training the classifier with the Treated Dataset proved to outperform significantly the one trained with the

Original Dataset during validation. Additionally, the test set was obtained from the Original Dataset, to better represent real crawled data.

Looking at the confusion matrix, important points to stand out are: some ”Post” samples are being mistaken as ”Health”; some (but few) not ”Post” samples, in this case ”Other”, are being classified as ”Solution”, when they pass to the second layer. Having in mind the goal of this text classifier in the overall system, which is to identify patient-driven solutions, the first point affects the solution recall, while the second affects its precision. However, the results show that these consequences just slightly affect the overall performance.

Table 5.5 serves as a term of comparison when evaluating the text classifier on crawled data (Section 5.2).

5.2 Evaluation on crawled data

A performance evaluation on crawled data was done to test the text classifier performance on a real scenario.

Starting from seed URLs provided by analysts of the Patient-Innovation organization, a broad crawl of depth 5 was done. In total, around 2000 webpages were crawled and consequently classified. From those, two disjoint subsets of around 200 webpages each were sent to a Patient Innovation analyst for manual labeling. She was told to label a page as Positive or Solution Hub (”Solution”), Negative (”Non-solution”), ”Health” or ”Other”. A Solution Hub is a webpage that contains or links to several solutions, often providing a short descriptive text of each solution. For the purpose of classification this kind of webpage was considered to be from class ”Solution”. The subsets obtained were randomly sampled and had the same class distribution of the set they came from.

The text classifier was configured with the same specifications of Section 5.1.1.

Tables 5.6 and 5.7 show the results of the text classifier on crawled data, test set 1 an test set 2 respectively, after cross-checking the manual labels with the labels automatically given by the classifier.

		Classified			
		Solution	Non-Solution	Health	Other
Actual	Solution	22	1	0	2
	Non-Solution	29	2	1	7
	Health	28	6	46	25
	Other	3	3	2	15

(a) Confusion matrix.

TPR / Recall (Solution)	FPR (Solution)	Precision (Solution)	F-Measure (Solution)	Recall	Precision	F-Measure
88.0 %	35.9 %	26.8 %	41.1 %	44.3 %	61.9 %	51.6 %

(b) Evaluation metrics.

Table 5.6: Results of the classifier on crawled test set 1.

As it can be seen, the text classifier performance drastically decreased from the one showed during validation. Observing the results, we can see that the main reasons for the classifier’s bad performance are: many ”Health” samples being missclassified in the first layer, and its important to notice that ”Health” samples being classified as ”Post” (the majority as ”Solution”) is specially bad for this classifier purpose; and many not ”Solution” samples being classified as so, in the second layer.

By manually inspecting the samples of the crawled test sets, we saw that the majority of ”Health” samples classified as ”Post” were either blog posts telling stories of people living with some kind of

		Classified			
		Solution	Non-Solution	Health	Other
Actual	Solution	24	2	1	7
	Non-Solution	27	9	5	10
	Health	9	1	33	26
	Other	1	0	7	23

(a) Confusion matrix.

TPR / Recall (Solution)	FPR (Solution)	Precision (Solution)	F-Measure (Solution)	Recall	Precision	F-Measure
70.6 %	24.5 %	39.3 %	50.5 %	48.1 %	60.5 %	53.6 %

(b) Evaluation metrics.

Table 5.7: Results of the classifier on crawled test set 2.

health related condition, the majority talking about how they overcome some obstacle imposed by their condition, or webpages describing health organizations for people with special needs. It is presumed that these were classified as ”Post” due to the similarity in vocabulary that these have to real ”Solution” samples and because the vast majority of ”Health” samples in the training set do not cover these cases. They are mostly webpages about general health topics. These include websites of health organizations addressing common health problems, like alcoholism and other addictions, physical therapy, nutrition, AIDS and other famous diseases; information describing diseases an treatments; news about health conditions, operations and well-being; clinic and pharmacy websites; health educational websites.

The second problem seems to have appeared due to the lack of training samples representing the negative class on the second layer. Therefore, most likely most samples that reach the second layer will fall on the ”Solution” category, as this class may have a larger variety of significant and characteristic terms (in this case bigrams) representing it, while the ”Non-solution” class may be represented by more sample specific terms, that do not capture the entire diversity of ”Non-solutions”. These terms might have a low frequency on the ”Non-solution” training data, but still appear highly ranked in its fingerprint due to the lack of samples. Additionally, even though there is a threshold set for the ”Solution” classification, the configured value was chosen according to the validation. Having different and more negative representatives may alter the threshold needed to separate both classes, and bring significance to the fingerprint of ”Non-solution”, changing the used *FFP-C* essence back to a similarity classifier.

The problems seem to be mostly in the training data, not on the model. Therefore, a solution could be to populate the training data with the manually labeled samples. The second problem could also be mitigated by setting a higher similarity score threshold on the second layer. However, the later solution could also reduce significantly the recall for the ”Solution” class, while the first proposed solution would not affect much the recall and it has the potential to significantly improve the precision.

One thing to notice is that, although the classifier does not seem to identify precisely the ”Solution” samples, it appears to identify the majority of health-related samples as so, while having a good precision. Health-related samples are the samples from the classes of ”Solution”, ”Non-solution” and ”Health”. Another thing to notice is that we have much more health-related samples on these test sets, than on the one used when testing with layer dependency, during validation (Section 5.1.1. This happens because the seed URLs used are health-related, and from them the crawler might not encounter a lot of ”Other” samples. This might affect the metric comparison of these results with the ones from validation (table 5.5) on the first layer, as the test sets used in this evaluation do not have the same membership percentage per class as the one used during validation. Nonetheless, the second layer classification on crawled data shows much poor results than the one during validation.

5.2.1 Using crawled data to improve the classifier

To prove the hypothesis given in the section above, test set 2 was added to the training data, using the resulting set to train the system's text classifier. The classifier model was kept unchanged. Test set 1 was used to verify if this addition improved the classifier's performance. The results can be seen on table 5.8. We opted by using test set 1 to verify the performance change because it was with this set that the classifier showed a worst performance.

		Classified			
		Solution	Non-Solution	Health	Other
Actual	Solution	22	2	0	1
	Non-Solution	18	14	1	6
	Health	25	13	59	8
	Other	3	6	4	10

(a) Confusion matrix.

TPR / Recall (Solution)	FPR (Solution)	Precision (Solution)	F-Measure (Solution)	Recall	Precision	F-Measure
88.0 %	27.5 %	32.4 %	47.3 %	54.7 %	67.5 %	60.4 %

(b) Evaluation metrics.

Table 5.8: Results of the classifier on crawled test set 1, after adding test set 2 to its training data.

Observing the results of this experiment, several things can be noticed. It can be seen an overall improvement on the classifier's performance. All metrics improved. Looking at the confusion matrix, we can see that more "Health" samples are being classified as so and a much better separation is achieved between the "Solution" and "Non-solution" classes, without affecting the "Solution" recall. This contributes to a better precision at classifying solutions. This confirms the theory that repopulating the training data with significant examples, even without changing the classification model, can improve the classifier's performance.

However, it can also be noticed that a lot of "Health" samples were still classified as "Solution". As there were not many of these examples in test set 2, the classifier could not improve significantly on this matter. Still, a slight improvement can be seen.

5.3 Crawler performance comparison

Besides from checking the text classifier performance, evaluating the effectiveness of the custom distiller is also very important to analyze the overall crawler performance. Checking the distiller performance is checking the focused crawling performance. In fact, it is the distiller that turns this crawler into a focused crawler, as it is the component that gives priority levels to unseen URLs.

To analyze the system's crawling performance, four crawls were made, following four different crawling approaches. A breadth-first approach, a best-first approach, a URL context approach and the proposed approach. A best-first approach gives priority to the URLs contained in webpages with the highest parsed score, basically it is an approach that just uses the parent parsing score. A URL context approach gives priority to the URLs with the highest context score, following the approach mentioned on 4.1.3. These two approaches were followed by setting a weight different than zero only to the parent parsing component in the best-first approach, and to the URL context component in the URL context approach. The proposed approach used equal weights for each component of the fetching score. All crawls started from the same set of 11 seed URLs, given by the Patient Innovation analysts. Additionally, all crawls were done through Nutch, ran for the same amount of iterations and a limitation of 25 pages per host per iteration was imposed to increase webpage diversity. All crawls were performed using a dual-core machine with capacity for 4 simultaneous threads and with $4GB$ of ram. In the end, all approaches crawled a total of around 4000 webpages.

The evaluation classifier used to classify the webpages visited was the one implemented for the proposed system, with the same configuration used on Section 5.1.1. As it was shown in Section 5.2, this classifier, with the current training data, does not present a high performance classifying ”Solution” samples on crawled data, and its classification can not be used as ground-truth. However, it can be used to compare different crawling approaches, because this way all will use the same classification truth. One just has to keep in mind that the measures presented, in reality, might have a lower true value.

To calculate the target recall, all URLs from pages classified as ”Solution” were collected during the breadth-first crawl. This would be the set of target webpages. As the broad approach was used to collect the target set, this was not used for comparison on target recall.

Figures 5.1 and 5.2 show the harvest rate and target recall on the ”Solution” topic for each crawling strategy, respectively. All the measure points presented were calculated at the end of each iteration for each approach, except for the first iteration. This was taken out because it represented no added value, as it was the seed URLs fetching iteration.

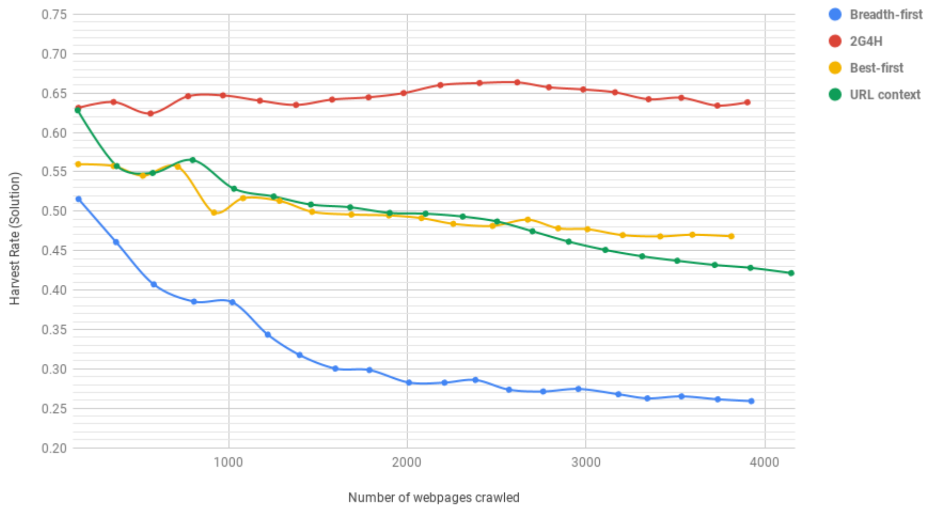

Figure 5.1: Comparison of harvest rate on the ”Solution” topic.

As it can be seen on Figure 5.1, all focused crawling approaches perform better than the broad approach in terms of harvest rate on the target topic. The broad approach, as the other ones, begins with a reasonably high harvest rate, but it soon starts to decrease, rapidly hitting low values and maintaining them. The best-first and the URL context approaches also decrease, although much slowly than the broad crawl. These seem to perform similarly on this metric. As for the 2G4H approach, it can be seen that the harvest rate maintains approximately a constant value. Additionally, it begins from the highest value of harvest rate, meaning that on the second iteration it already performed a better choosing of URLs than the other techniques.

Regarding the target recall, it can be seen that the proposed approach's value is always higher than the other two focused crawling approaches. Furthermore, the 2G4H approach's target recall keeps constantly increasing, as opposed to the other approaches. These appear to stagnate at a point. On this metric, it can be seen that the best-first approach outperforms the URL context approach.

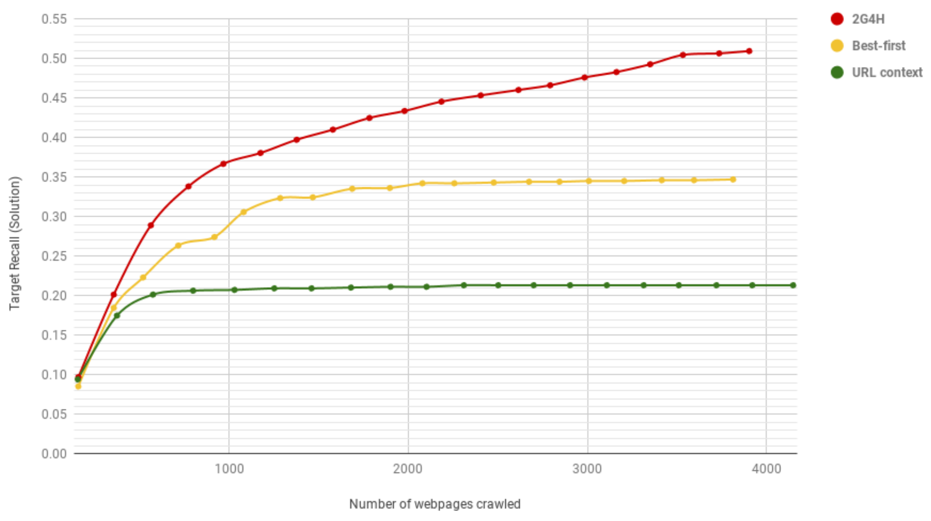

Figure 5.2: Comparison of target recall on the "Solution" topic.

Concerning time efficiency, the 2G4H crawl made with a single dual core machine with capacity for 4 threads and with $4GB$ lasted around 160 minutes. Considered it crawled about 4000 webpages, this system has the capacity of delivering 25 classified webpages per minute, with the mentioned specifications. Considering a harvest rate of 65%, which is the mean harvest rate for the 2G4H crawler showed on Figure 5.1, this system delivers around 16 webpages classified as "Solution" per minute. This is more than sufficient considering that the results still have to go through medical screening, for them to be published in the Patient Innovation platform (or similar). Furthermore, considering the precision shown on the improved performance of the classifier on crawled data (32.4% on table 5.7), we can estimate that the crawler outputs around 5 true "Solution" webpages per minute, which is far more efficient that the manual search currently done.

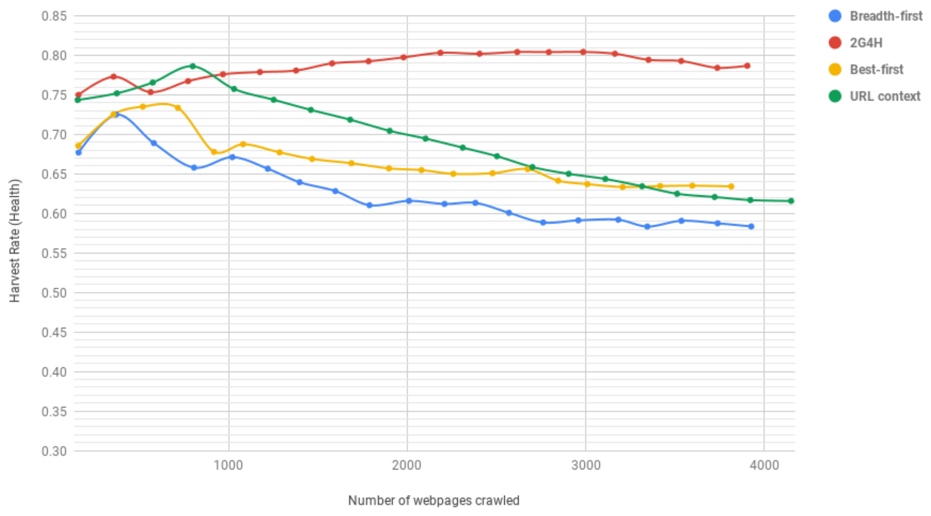

Figure 5.3: Comparison of harvest rate on the health-related topics.

While the evaluation classifier does not have a high accuracy classifying "Solution" samples, it has high accuracy classifying health-related samples, as stated in Section 5.2. So it is interesting to analyze its harvest rate on health-related topics ("Solution", "Non-solution" and "Health"), which is depicted in

Figure 5.3. As it can be seen, in all crawl approaches above 50% of the webpages crawled regard the health theme, even the broad crawl. This one is just due to the starting seed URLs, which are health-related. As for the focused crawling methods, it can be seen that the same tendency pictured in Figure 5.1 is maintained. What is important to highlight is that on Figure 5.3 the values presented should be accurate. Therefore, one can see that the proposed crawling approach reaches values of harvest rate on health-related topics slightly above 80%. These are considerably high values, and much higher than the other approaches.

To conclude that the proposed approach is in fact a focused crawling approach, a requirement is for it to be more efficient than a breadth-first (broad) crawling approach, while searching for patient-driven solutions. This system accomplished that requirement and it even showed that it can perform better than other common focused crawling approaches regarding metrics of harvest rate and target recall.

5.4 Discussion

Upon analyzing the results, it can be seen that there is much room for improvement. While the crawler performance seems satisfactory, the tests on crawled data show that the text classifier needs improvement.

The main problem may be on the dataset used to train the classifier. Using human labeled webpage repositories, like DMOZ, to collect data for webpage classification has proven to be a satisfactory method when the classes in play are broad topics like Health, Sports, Politics, Technology, etc. However, for very specific classes there must exist a lot of specific information describing the mentioned classes and separate the specific classification from broad classification. We tried to achieve the later by adding the two layers to the text classifier. But there was clearly a lack of negative examples on the second-layer, to properly separate the positive and negative classes.

In [74], Christensen et al. had two individuals classifying 3000 messages from the website they were studying to find innovative ideas. From those samples, both analyzers agreed that 137 were ideas and 2666 were non-idea texts. From a website that is known to have a lot of innovative ideas, it can be seen that non-idea information is much more present. This illustrates the ratio of relevant information that one can expect from places considered to be sources of user innovation. Illustrating with a practical example, even if a crawler is crawling the right websites, it should encounter a lot more of non-relevant information. This information should be well represented in the crawler's classifier, in order to be avoided.

Although the part of our dataset used to train the second layer was also obtained by human screening of thousands of text examples and it contained a lot of positive information regarding patient-driven solutions, it lacked a lot of important negative information. Its effects could be seen while testing the classifiers performance. The negative class could not stand out from the positive on most approaches followed during validation, and the same thing happened during testing, even with the validated approach.

Moreover, identifying user innovation is a controversial topic. It is harder to identify if a sample is considered a user innovation than to identify if it belongs to a broad topic. Many people could disagree on the first task. In [74] five individuals were chosen to manually label the test set used to test the classifier on this research. Having several individuals labeling the same samples improves the manual labeling consistency and correctness. In this Master thesis, the training set used was labeled by more than one person, because it was extracted form the Patient Innovation database and their analysts have to agree on the classification of posts. However, the test sets used could only be labeled by one person. This could have hindered the classifier's tests.

To avoid the problems stated above, prior to the classifier validation, more negative samples should have been identified and the manual classification process should always have two or more people agreeing on the classification.

One thing that could help on the lack of negative examples is broadening the ”Non-solution” class definition. In Section 5.2, it was noticed that a lot of ”Health” samples were being classified as ”Solution” due to the similarity in vocabulary. The majority of these samples described stories of people living with some health-condition, talking about how they overcome everyday life obstacles. Although these are not considered to be ”Post” samples in the current class definition, the ”Non-solution” class definition could be broadened to include these examples, as they share the same nature of this class samples. They are pieces of texts originated from the same source (patients) and they both describe actions leading to overcoming a health-related problem, besides not being considered ”Solutions”. Having these samples populating the ”Non-solution” class should increase the number of negative examples, and it should also further accentuate the separation between the ”Health” class and the ”Post” class.

[74] used a complex classification model to identify innovative ideas. They obtained satisfactory results, while focusing on just a single online community. On the other hand, this crawler is supposed to identify user innovation throughout the entire Web. This last fact and this system’s classifier’s test results highlight that automatic classification of user innovation is a much bigger problem than what it was initially thought.

Chapter 6

Conclusion and Future Work

In this Master thesis, it was built a system with the goal of automatically searching for innovative patient-driven solutions and index the results. The system is composed of a focused crawler, with two components responsible for classification and focused navigation, the classifier and the distiller, respectively, and a web indexer, which indexes the webpages considered relevant by the crawler.

Following state-of-the-art approaches, it was decided to pursue a machine learning based method to classify the visited webpages. As there were patient-driven solutions already available, this seemed to be the best technique to use. It was opted to have a text classifier with two layers.

The classifier validation results showed to be very good, with all metrics above 90%. However, the classifier performance on crawled data was not so satisfactory. Nonetheless, it was noticed that the classifier's results on crawled data were justified by the lack of more contextualized representative samples on the ”Health” and ”Non-solution” classes of the training data. This demonstrated that the initial dataset needed to be improved, as the problem seemed to be on the classifier's training data and not on the model. Further results showed that the classification can be improved by re-training the classifier with manually labeled crawled data.

The distiller is the component that prioritizes the URLs to be fetched. It was based on common state-of-the-art approaches. These approaches relied on the content, URL context or scores of the pages where the URL being analyzed was found. By combining these methods, it was expected to have a better result than using a single one, as they target different aspects and each one proved to work on past researches.

When testing the proposed crawling approach, it showed it is more efficient than a broad approach and than some of the focused approaches the distiller is based on. This proves the belief that the combination of methods used can perform focused crawling and perform better than using a single method for the focused task.

This Master thesis shows that it is possible to build a system that successfully searches for innovative patient-driven solution. The proposed system can be used on online platforms for patient-driven solution diffusion, like Patient-Innovation, to increase the search efficiency and diversity of solutions to be posted. Additionally, this opens doors to the automatic search of user innovation. User innovation researches can work upon this thesis' methodology and results to learn better ways of collecting and studying user innovation cases.

6.1 Future work

The Master thesis here presented showed that it is possible to search for and automatically identify patient-driven solutions through the Web. The classifier and distiller appeared to work well together, presenting a good crawling efficiency when compared to other common approaches. However, the problem of user innovation identification showed to be much harder than it seemed. The classifier has a lot of room to improve and the distiller can be optimized for even better performances.

As it was shown in Section 5.2.1, the classifier's performance can improve just by adding meaningful data to its training set. It was also pointed out in Section 5.4 that the classifier would achieve a better performance if there were more "Non-solution" samples in its training set. Additionally, as this system is supposed to run as solution provider for patient-driven solution diffusion platforms, there will always be analysts screening through the provided results. Therefore, the system would benefit from an online training classifier that would re-train itself with more relevant data, automatically updating its model through a process of several validation tests, when a batch of new manually labeled results are available. This system would be supervised by the platform analysts, which would label the new training samples. Currently, the training set can be manually improved by adding the new samples through the same batch of processes used to produce the results of Section 5.2.1. However, this does not alter the classifier's model.

Currently, the fetching score that the distiller assigns to newly discovered URLs is a linear combination of three other scores. Its linear coefficients can be manually set on the Nutch configuration file, and they were all set to equal values during the crawler performance evaluation. However, tests were not made to find the optimal combination of the coefficients. A sample web graph can be taken from the Web, to create a controlled environment within which an optimization method to solve linear problems can be used to determine the best configuration for the coefficients. The objective function of this optimization problem would involve the harvest rate and the target recall, in order to maximize both at each iteration or at the end of all iterations.

Also, the current approach to determine the fetching score of URLs could be combined with link-analysis algorithms, like PageRank or HITS, to study its influence on the crawler's performance. Additionally, one could understand what kind of webpages are directly connected to patient-driven solution webpages.

Bibliography

[1] S. Hyysalo, M. Johnson, and J. K. Juntunen, “The diffusion of consumer innovation in sustainable energy technologies,” *Journal of Cleaner Production*, vol. 162, pp. S70–S82, sep 2017.

[2] A. Ng, “Support Vector Machines,” in *CS229 Lecture notes*.

[3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[4] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, “Focused Crawling Using Context Graphs,” in *VLDB*, 2000, pp. 527–534.

[5] J.-Y. Nie, “IR - INDEXING.” [Online]. Available: http://www.iro.umontreal.ca/{~}nie/IFT6255/Indexing.pdf (Accessed 2018-08-20).

[6] F. Gault, “User innovation and the market,” *Science and Public Policy*, vol. 39, no. 1, pp. 118–128, 2012.

[7] C. Baldwin and E. von Hippel, “Modeling a Paradigm Shift: From Producer Innovation to User and Open Collaborative Innovation,” *Organization Science*, vol. 22, no. 6, pp. 1399–1417, 2011.

[8] E. Von Hippel, *Democratizing innovation*. MIT press, 2005.

[9] S. Hyysalo, J. K. Juntunen, and S. Freeman, “User innovation in sustainable home energy technologies,” *Energy Policy*, vol. 55, pp. 490–500, apr 2013.

[10] H. Nishikawa, M. Schreier, and S. Ogawa, “User-generated versus designer-generated products: A performance assessment at Muji,” *International Journal of Research in Marketing*, 2013.

[11] C. Hienerth, E. Von Hippel, and M. Berg Jensen, “User community vs. producer innovation development efficiency: A first empirical study,” *Research Policy*, 2014.

[12] S. Hyysalo, “User innovation and everyday practices: Micro-innovation in sports industry development,” *R and D Management*, 2009.

[13] M. E. Hinsch, C. Stockstrom, and C. Lüthje, “User Innovation in Techniques: A Case Study Analysis in the Field of Medical Devices,” *Creativity and Innovation Management*, vol. 23, no. 4, pp. 484–494, 2014.

[14] P. Oliveira, L. Zejnilovic, H. Canhão, and E. von Hippel, “Innovation by patients with rare diseases and chronic needs,” *Orphanet Journal of Rare Diseases*, vol. 10, no. 1, p. 41, 2015.

[15] S. Hyysalo, P. I. A. Helminem, S. Makinen, M. Johnson, J. K. Juntunen, and S. Freeman, “Intermediate search elements and method combination in lead-user searches,” *International Journal of Innovation Management*, vol. 19, no. 01, p. 1550007, 2015.

[16] L. Zejnilović, P. Oliveira, and H. Canhão, *Innovations by and for Patients, and Their Place in the Future Health Care System*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 341–357.

[17] E. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text Classification Using Machine Learning Techniques," *WSEAS transactions on computers*, vol. 4, pp. 966–974, 2005.

[18] E. Cambria and B. White, "Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]," *IEEE Computational Intelligence Magazine*, vol. 9, no. 2, pp. 48–57, 2014.

[19] R. Jindal, R. Malhotra, and A. Jain, "Techniques for text classification: Literature review and current trends," *webology*, vol. 12, no. 2, p. 1, 2015.

[20] Google, "word2vec," 2013. [Online]. Available: https://code.google.com/archive/p/word2vec/ (Accessed 2018-08-15).

[21] J. Lilleberg, Y. Zhu, and Y. Zhang, "Support vector machines and Word2vec for text classification with semantic features," in *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, 2015, pp. 136–140.

[22] N. Homem and J. P. Carvalho, "Authorship identification and author fuzzy &#x201C;fingerprints&#x201D;," in *2011 Annual Meeting of the North American Fuzzy Information Processing Society*, 2011, pp. 1–6.

[23] S.-B. Kim, H.-C. Rim, D. Yook, and H.-S. Lim, "Effective Methods for Improving Naive Bayes Text Classifiers," in *PRICAI 2002: Trends in Artificial Intelligence*, M. Ishizuka and A. Sattar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 414–423.

[24] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, no. 1. Citeseer, 1998, pp. 41–48.

[25] A. Cardoso-Cachopo and A. L. Oliveira, "An Empirical Comparison of Text Categorization Methods," in *String Processing and Information Retrieval*, M. A. Nascimento, E. S. de Moura, and A. L. Oliveira, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 183–196.

[26] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[27] J. P. Carvalho, H. Rosa, and F. Batista, "Detecting relevant tweets in very large tweet collections: The London Riots case study," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017, pp. 1–6.

[28] S. Curto, J. P. Carvalho, C. Salgado, S. M. Vieira, and J. M. C. Sousa, "Predicting ICU readmissions based on bedside medical text notes," in *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 2144–a–2151–h.

[29] F. Batista and J. P. Carvalho, "Text based classification of companies in CrunchBase," *IEEE International Conference on Fuzzy Systems*, vol. 2015-Novem, 2015.

[30] M. Najork, "Web Crawler Architecture," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 3462–3465.

[31] Maurice de Kunder, "World Wide Web Size." [Online]. Available: http://www.worldwidewebsize.com/ (Accessed 2018-06-18).

[32] V. Papavassiliou, P. Prokopidis, and G. Thurmair, “A modular open-source focused crawler for mining monolingual and bilingual corpora from the web,” in *Proceedings of the sixth workshop on building and using comparable corpora*, Sofia, Bulgaria, 2013, pp. 43–51.

[33] G. Mohr, M. Stack, I. Rnitovic, D. Avery, and M. Kimpton, “An Introduction to Heritrix,” in *4th International Web Archiving Workshop*, 2004, pp. 109–115.

[34] M. Kumar, R. Bhatia, and D. Rattan, “A survey of Web crawlers for information retrieval,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 6, pp. e1218—-n/a, 2017.

[35] T. V. Udapure, R. D. Kale, and R. C. Dharmik, “Study of web crawler and its different types,” *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 1–5, 2014.

[36] N. K. Kameswara Rao and G. P. Saradhi Varma, “Architecture of a WebCrawler,” in *National Conference on Research Trends in Technology in Information Technology*, 2010.

[37] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank Citation Ranking: Bringing Order to the Web.” Stanford InfoLab, Technical Report 1999-66, 1999.

[38] J. M. Kleinberg, “Authoritative Sources in a Hyperlinked Environment,” *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[39] K. Sigurosson, “Incremental crawling with Heritrix,” 2010.

[40] Internet Archive, “Internet Archive: Digital Library of Free & Borrowable Books, Movies, Music & Wayback Machine.” [Online]. Available: https://archive.org/ (Accessed 2018-06-20).

[41] A. Bialecki, “AdaptiveFetchSchedule (apache-nutch 1.12 API).” [Online]. Available: http://nutch.apache.org/apidocs/apidocs-1.12/org/apache/nutch/crawl/AdaptiveFetchSchedule.html (Accessed 2018-09-24).

[42] D. N. Singhal, A. Dixit, and A. Sharma, “Design of a Priority Based Frequency Regulated Incremental Crawler,” *International Journal of Computer Applications*, vol. 1, 2010.

[43] S. Xi, F. Sun, and J. Wang, “A cognitive crawler using structure pattern for incremental crawling and content extraction,” in *Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on*, 2010, pp. 238–244.

[44] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, “UbiCrawler: a scalable fully distributed Web crawler,” *Software: Practice and Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[45] R. Campos, O. Rojas, M. Marín, and M. Mendoza, “Distributed Ontology-Driven Focused Crawling,” in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 108–115.

[46] M. Badawi, A. Mohammed, A. Hussein, and M. Gheith, “Maintaining the search engine freshness using mobile agent,” *Egyptian Informatics Journal*, vol. 14, pp. –, 2013.

[47] H. Hao, C. Mu, X. Yin, S. Li, and Z. Wang, “An improved topic relevance algorithm for focused crawling,” in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, 2011, pp. 850–855.

[48] Y. Du, W. Liu, X. Lv, and G. Peng, “An improved focused crawler based on Semantic Similarity Vector Space Model,” *Applied Soft Computing*, vol. 36, pp. 392–407, nov 2015.

[49] Z. Zheng and D. Qian, “An improved focused crawler based on text keyword extraction,” in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, 2016, pp. 386–390.

[50] T. Peng and L. Liu, “Focused crawling enhanced by CBP–SLC,” *Knowledge-Based Systems*, vol. 51, pp. 15–26, oct 2013.

[51] M. Kumar, A. Bindal, R. Gautam, and R. Bhatia, “Keyword query based focused Web crawler,” *Procedia Computer Science*, vol. 125, pp. 584–590, 2018.

[52] Y. Du, Q. Pen, and Z. Gao, “A topic-specific crawling strategy based on semantics similarity,” *Data & Knowledge Engineering*, vol. 88, pp. 75–93, 2013.

[53] F. Yuan, C. Yin, and J. Liu, “Improvement of PageRank for Focused Crawler,” in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, vol. 2, 2007, pp. 797–802.

[54] G. Pant and P. Srinivasan, “Learning to Crawl: Comparing Classification Schemes,” *ACM Trans. Inf. Syst.*, vol. 23, no. 4, pp. 430–462, 2005.

[55] H.-T. Zheng, B.-Y. Kang, and H.-G. Kim, “An ontology-based approach to learnable focused crawling,” *Information Sciences*, vol. 178, no. 23, pp. 4512–4522, dec 2008.

[56] C. Su, Y. Gao, J. Yang, and B. Luo, “An efficient adaptive focused crawler based on ontology learning,” in *Fifth International Conference on Hybrid Intelligent Systems (HIS’05)*, 2005, p. 6 pp.

[57] M. Shokouhi, P. Chubak, and Z. Raeesy, “Enhancing focused crawling with genetic algorithms,” in *International Conference on Information Technology: Coding and Computing (ITCC’05) - Volume II*, vol. 2, 2005, pp. 503–508 Vol. 2.

[58] H. Liu, J. Janssen, and E. Milios, “Using HMM to learn user browsing patterns for focused Web crawling,” *Data & Knowledge Engineering*, vol. 59, no. 2, pp. 270–291, nov 2006.

[59] S. Chakrabarti, M. Berg, and B. Dom, “Focused crawling: A New Approach to Topic- Specific Web Resource Discovery,” *Computer Networks*, vol. 31, no. 11, pp. 1623–1640, 1999.

[60] G. Pant and P. Srinivasan, “Link contexts in classifier-guided topical crawlers,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 107–122, 2006.

[61] P. Srinivasan, F. Menczer, and G. Pant, “A General Evaluation Framework for Topical Crawlers,” *Information Retrieval*, vol. 8, no. 3, pp. 417–447, jan 2005.

[62] Wikipedia, “Denial-of-service attack - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Denial-of-service{_}attack (Accessed 2018-08-18).

[63] M. Koster, “The Web Robots Pages,” 1996. [Online]. Available: http://www.robotstxt.org/robotstxt.html (Accessed 2018-08-18).

[64] P. Prokopidis and V. Papavassiliou, “ILSP Focused Crawler.” [Online]. Available: http://nlp.ilsp.gr/redmine/projects/ilsp-fc (Accessed 2018-06-11).

[65] P. Jack, “Heritrix.” [Online]. Available: https://webarchive.jira.com/wiki/spaces/Heritrix/overview (Accessed 2018-06-11).

[66] R. Khare, D. Cutting, K. Sitaker, and A. Rifkin, “Nutch: A flexible and scalable open-source web search engine,” *Oregon State University*, vol. 1, p. 32, 2004.

[67] T. A. S. Foundation, “Apache Nutch.” [Online]. Available: http://nutch.apache.org/ (Accessed 2018-06-11).

[68] A. Tripathi, “Unit-14 Overview of Web Indexing, Metadata, Interoperability and Ontologies.” IGNOU, 2017.

[69] Z. Malki, “Comprehensive Study and Comparison of Information Retrieval Indexing Techniques,” *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 7, no. 1, pp. 132–140, 2016.

[70] The Apache Software Foundation, “Apache Solr.” [Online]. Available: http://lucene.apache.org/solr/ (Accessed 2018-08-20).

[71] The Apache Software Foundation, “Apache Solr - Resources.” [Online]. Available: http://lucene.apache.org/solr/resources.html (Accessed 2018-08-20).

[72] The Apache Software Foundation, “Apache Lucene 7.4.0 Documentation.” [Online]. Available: https://lucene.apache.org/core/7{_}4{_}0/index.html (Accessed 2018-08-20).

[73] A. Abbasi, T. Fu, D. Zeng, and D. Adjeroh, “Crawling Credible Online Medical Sentiments for Social Intelligence,” in *2013 International Conference on Social Computing*, 2013, pp. 254–263.

[74] K. Christensen, S. Nørskov, L. Frederiksen, and J. Scholderer, “In Search of New Product Ideas: Identifying Ideas in Online Communities by Machine Learning and Text Mining,” *Creativity and Innovation Management*, vol. 26, no. 1, pp. 17–30.

[75] X. Qi and B. D. Davison, “Knowing a Web Page by the Company It Keeps,” in *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, ser. CIKM ’06. New York, NY, USA: ACM, 2006, pp. 228–237.

[76] E. Frank, M. Hall, P. Reutemann, and L. Trigg, “Weka 3 - Data Mining with Open Source Machine Learning Software in Java.” [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/index.html (Accessed 2018-06-10).