Final Year Project Report

An Online Document Retrieval and Classification System based on Information Retrieval and Machine Learning Techniques

Thomas J. Fitzpatrick

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Dr. Brett A. Becker

UCD School of Computer Science

University College Dublin

April 6, 2018

Project Specification

General:

This project combines state-of-the-art Information Retrieval and Machine Learning Techniques to develop a Web crawler and Document Classification System capable of finding and classifying documents of a specified type. Given a list of seed URLs, the crawler procedurally navigates through all the seeds links and, following them, looks for pages that are likely to be documents of a specified type (e.g. syllabi, patents, instruction manuals, etc.).

This consists of the following:

1. A Web crawler capable of systematically navigating through a website, finding all links and searching them efficiently.
2. A process for parsing these pages and preparing them for input into a Machine Learning classifier.
3. Use of the Bag of Words model to prepare documents for training and testing the classification algorithm.
4. Validation of the information retrieval and Machine Learning system to identify documents that show a high likelihood of matching the specified type.

The project is built with the Python programming language. As a proof of concept, the system is first trained to retrieve and classify programming syllabi. The crawler, when given a list of university home pages from around the world, is intended to return a list of pages detailing the syllabi of programming modules from those universities.

The Core objectives listed below are aspects of the project that define its success. Implementation of all Core features should result in a successful crawler. The Advanced aspects are further developments that may be added once the core features have been implemented in order to increase functionality or accuracy.

Core:

- Development of a Web crawler that can efficiently navigate through websites.
- Development of an effective method for parsing or pre-processing html pages.
- Implementation of the bag of words Machine Learning model for preparing parsed pages for the classifier.
- Classification of the pages retrieved by the crawler.

Advanced:

- Determining if other methods such as those below can improve the classification performance:
  - Word clustering
  - Support Vector Machine classification
  - Word2Vec
- Generalizing the system to retrieve and classify another type of document (e.g. other syllabi, patents, instruction manuals, etc.).
- Extending the classification to work with other file types (e.g. PDF, docx, etc.).

Abstract

Sourcing a large collection of similar documents of a specific type on the Internet can be a difficult task. Search engines are effective at narrowly finding specific documents or pages relevant to a certain query but when searching for a large collection of related documents of a specific type, such as computer programming syllabi, instruction manuals or executive assistant CVs (résumés), they often prove less effective, requiring human input. Thus, a system that can automate the process of returning a large collection of related documents of a similar type could save time and effort in many endeavors. This report documents the process of developing such a system using syllabi for computer programming modules (classes) as a case study.

This system combines techniques from the Information Retrieval and Machine Learning domains to narrow the document space and return results that hopefully contain large numbers of related documents of a specific type. This narrowing is achieved by employing a Web crawler that navigates the web starting from a specified list of seed pages, to discover documents. Results are then narrowed further by testing these documents using a Machine Learning model trained on a set of documents of the desired type.

Acknowledgments

I would like to thank my supervisor Dr. Brett Becker for not only the extensive support and guidance he offered throughout this work, but also for introducing me to the process of scientific research and its intricacies as a whole.

I would also like to extend my sincere gratitude to Dr. Catherine Mooney and Dr. David Lillis, who offered much of their time and expertise throughout the year.

Finally, I am very grateful for the environment created in the school of Computer science by both the faculty and my fellow students. Their enthusiasm and passion was a source of inspiration throughout the four years that lead me to this point.

Table of Contents

Chapter 1: **Introduction**

This report documents a breakdown of various aspects of this project, titled: "An Online Document Retrieval and Classification System based on Information Retrieval and Machine Learning Techniques". It is comprised of this introduction, a breakdown of the preparatory background research performed, the general approach adopted, specifics of the design and implementation, and details of the testing and evaluation performed on the resulting system.

The basis of this project deals with the problem of finding specific documents on the Internet. There are many tools which tackle similar issues, for example search engines, web directories and standard Web crawlers, however, they are not perfect, and are ill suited to tasks such as finding collections of similar documents such as syllabi. For this reason, another approach is explored here, combining a Web crawler which searches for links with a Machine Learning trained classifier which identifies links to matching documents.

1.1 Project Motivation:

The proposal for this project was motivated by an internship which I completed with Dr Becker throughout the summer of 2017. This work was part of an ACM/SIGCSE funded special project in which the learning outcomes of introductory programming modules were examined to find out what topics were being covered. This research resulted in a paper submitted to the 23rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018) titled "What Exactly Are We Expecting Our Introductory Programming Students to Achieve?". A significant portion of this work involved sourcing these syllabi manually, searching through University websites for module documentation and identifying the syllabi for these modules. I found that this issue was common when performing analyses in CS Education Research, for example, the authors of [1] resorted to a similar process, sourcing 103 syllabi from 101 institutions in 12 countries over a number of weeks. Endeavors like these demand significant coordination efforts and hours of manual searching. A tool like the one described in this report which could potentially automate this process and be used to generate and even larger collection of syllabi could prove extremely useful within CS education research and other areas.

A byproduct of this internship was a set of such programming syllabi which formed an apt set of data for training the Machine Learning algorithm that would in turn identify other syllabi. This training set is considerably large in comparison to those available in similar studies searching for programming syllabi and so it was expected to allow the development of an algorithm with a greater degree of accuracy.

Several other studies have attempted to create similar tools capable of sourcing Computer Science syllabi for the purposes of education research but none yet have combined a Web crawler with a classifier in such a way. These other projects will be discussed further in section 2.4. One aspect which makes this system unique is that running it with a set of seed links, in this case the home pages of University websites from the QS World University Rankings 2017, emulates the approach which we found most effective when sourcing syllabi; that is, going one by one through a list of universities and individually finding their programming syllabi. This approach aims to reduce the number of duplicate documents produced by other systems which often simply classified a set of

web pages from a search engine query.

Creating a Web crawler capable of doing this would also form an excellent proof of concept for a more general case, if it can work for syllabi then perhaps it could be extended to find sets of instructions for particular appliances or CVs of people with particular profiles. It is expected that success in this application would translate conveniently into other similar areas such as these.

1.2 Project Goals:

The primary goal of this project was the development of a Web crawler which, given a list of seed URLs for University web pages, can procedurally navigate through the seeds links and, following them, identify syllabi of Programming modules. This process involves two major components, the Web crawler itself and the Machine Learning algorithm which will be trained on the collection of syllabi previously discussed. Once these were implemented, experimentation with various aspects of the crawler to try improve performance or functionality was performed, for example by modifying it to handle PDF documents, or improvements to the set of non-syllabi examples being used to train the classifier.

As a whole, this project incorporates two major areas of Computer Science; Information Retrieval in the form of the Web crawler and Machine Learning in the classification system. In addition to these, it serves as a capstone project for my Undergraduate Studies, bringing together many other aspects of subjects studied over the years. The Web crawler implements the Data Structures and Algorithms as well as the Object Oriented programming methodologies covered in second year. The method used for crawling will also require knowledge of the internet and its protocols, all of which was covered in modules such as Networks and Internet Systems which I studied in third year. The various Machine Learning techniques were also formalized and further developed in UCDs Machine Learning module. Working on this project was an opportunity that has leveraged knowledge from most, if not all of my undergraduate studies.

Chapter 2: **Background Research**

The Internet is the largest and most used source of information available to mankind. There are a myriad of tools available for searching through and filtering this information, search engines, web directories and Web crawlers all exist to this effect. However, when it comes to the task of reliably finding collections of similar documents, a great deal of human input and manual filtering is required. Using a generic search engine often yields many irrelevant links and is an error prone process. Search engine results often return a small selection of accurate results but fall short when seeking a large collection of results matching a particular query. Further, these systems all have their own bias, intentional or not. For example, the PageRank algorithm which is used by Google highly favours pages filling narrow criteria. Thus there is a need for tools that apply different methodologies and return, for example large collections of specific documents.

This project is based around the search for syllabi of introductory programming modules. As a resource, syllabi are of great utility, they form the basis of a course and can give great insight into how courses are being taught. Thus any technology which automates the gathering of syllabi for analysis could have considerable use in educational research. Further, such technology could be used to find important documents in many other domains.

The project goals will be achieved by marrying too distinct technologies: Information Retrieval and Machine Learning. Specifically, a Web crawler or spider will be developed which can navigate from a set of seed pages, through linking pages, searching for syllabi. These syllabi will be identified using a Machine Learning algorithm trained on similar documents.

Before undertaking such a task, sufficient background research in the chosen domain and the chosen technologies was necessitated. The following sections present the results of the background research performed in the areas of Information Retrieval, Machine Learning, Data Preparation and Computer Science Education.

2.1 Information Retrieval

The area of Information Retrieval is of major significance in this project. Specifically, sufficient research into Web crawlers and their structures and implementation was necessary to successfully integrate them into this document retrieval system.

Web crawlers were initially conceived at a time when the Internet was only a fraction of the size it is today. Eichmann discusses one of the earliest implementations of a Web crawler in his development of the RBSE Spider [2]. At the time, it was feasible to search and index the entire web, crawling over all links and building a complete model of its topology. The exponential growth of the Web since this paper was published in 1994 has rendered such a concept infeasible, and so many refinements have been made to the structure and functionality of these Crawlers to enable them to intelligently and efficiently search further and deeper into the depths of the web. Nonetheless, this paper serves as an excellent starting point for background research for this aspect of this project.

2.1.1 General Structure:

One of the most influential Web crawlers was Mercator [3], a well-documented, scalable, extensible implementation that is often used as the building block for any system which implements such a Crawler. This implementation clearly establishes the basic requirements of a standard Web crawler. A Web crawler starts with an input list of seed URLs and repeatedly performs the following operations.

1. Remove a URL from the URL list.
2. Download the corresponding document and extract any links contained in it.
3. For each of the extracted links, ensure that it is an absolute URL (derelativise it if necessary) and add it to the list of URLs to download, checking to ensure it hasn't been downloaded before.

This simple routine forms the basis of how Web crawlers work. For the purpose of this project a method of prioritizing links which would likely lead to URLs for syllabi was needed as well as a method for identifying the documents using the Machine Learning model. Links which are classified as syllabi are returned from the system once an end point has been determined, for example, when all seed links have been searched to a specified depth. In order to successfully implement these additional features to the base Web crawler structure it is necessary to further understand the individual components of one. Those relevant to this project will be discussed briefly below.

- *URL frontier:* This is simply a data structure which contains all the URLs that have been discovered and are yet to be downloaded and crawled, referred to by Heydon and Najork as a "URL frontier". In Mercator this is implemented as a FIFO queue, however, for the purpose of this project, the URL frontier will be implemented using a priority queue with priorities being given to URLs that are likely to lead to Computer Science sections of the domain being searched.
- *Link extraction:* In Mercator, link extraction is performed using a processing module associated with the MIME type of the document. Links are extracted, converted to an absolute URL and checked to see if this URL has been seen before. This URL-seen test checks if the URL is already in the URL frontier or has already been downloaded. If it is a new URL, it is added to the URL frontier. This process will remain almost identical in the system being described in this report apart from the fact that MIME types and link extraction in general will be handled by the Beautiful Soup Python library [4].
- *Checkpointing:* Because these Web crawlers are designed to run for hours or days at a time, continuously scraping the internet, it is necessary to prepare for the contingency of the crawler failing or being forced to shut down. Mercator handles this by implementing a checkpointing system in which it writs regular snapshots of its state to disk. Such a comprehensive system will not be necessary within this project as it will not need to store the state of the entire system. Instead, we seek only the positively identified syllabi attained from each seed URL. Therefore, simply saving these identified syllabi to disk as the application crawls will be sufficient. If the system fails it can simply be restarted from the most recent seed page, skipping the ones that have already been crawled.

2.1.2 URL Reordering:

The basis of this project is finding and identifying documents of a specific type. Although identification of these documents will be performed by a Machine Learning algorithm, the crawler must still be built with this functionality in mind. Thus, it must attempt to direct the search towards links that are more likely to match the document type being searched for. Therefore, it was necessary to research methods of improving the efficiency of the crawling using URL reordering within the URL Frontier. This topic is discussed in detail by Cho et al in [5].

In this study, Cho et al. discuss several metrics for assessing and reordering URLs that a crawler comes across during its runtime:

- *Similarity to a Driving Query Q:* This involves analyzing an entire Web document to check if it matches the query being searched for. This method would be overly complex as I will be procedurally testing URLs with a Machine Learning algorithm, also, the URL reordering procedure is simply a method of improving reach and efficiency in limited depth search in this case, rather than a full Web crawler.
- *Backlink Count:* A common metric for measuring URLs, this involves tracking the number of pages that link to a given URL. Again, this will not be suitable for this project as the crawler will not be reaching a sufficient portion of the web to accurately asses a URL on this basis.
- *PageRank:* One of the most common metrics of ranking used in Crawlers, PageRank adds to the Backlink approach by taking into account the importance of pages that link to a given URL. Again, the simplicity of the crawler and the niche and small number of links that it will reach would prevent this method from being practical.
- *Location Metric:* This is a simplistic approach which measures the importance of a page as a function of its location rather than its content. This involves, for example, checking for keywords in the URL that indicate relevance to a search query or topic. Location metrics can also take into account factors such as number of slashes within URLs which allows us to perhaps prioritize links with fewer slashes. This location metric, searching for keywords in the URL, is the approach primarily implemented for ordering URLs within the URL Frontier of the proposed system. The intention is that it acts as a simple heuristic for improving the selection of pages to crawl and thus the pages which are tested by the Machine Learning algorithm.

One observation which Cho et al. noted was the usefullness of visiting URLs that have anchor text similar to the driving query, have some query terms within the URL itself or have a short link distance to a page that is of high relevance. These are all potential modifications which could be implemented within this system to refine and improve its performance.

The most interesting conclusion which Cho et al. derived from their research into URL reordering was the effectiveness of a good ordering strategy on a Web crawler's ability to reach a significant portion of relevant pages early in a crawl. In particular, they commented on how this property can be highly useful when dealing with crawlers which only crawl a fraction of the web or have limited resources, both features of the crawler which will guide this system. Thus, implementation of an effective URL reordering scheme within the crawler will be of great importance if it is to be as effective as possible.

2.2 Machine Learning

Machine Learning will form the second area of research primarily relevant to this project. There are many Machine Learning techniques which can be used to train a model for text classification, all of which come with advantages and disadvantages that can make them more or less suitable for particular bodies of text or types of classification. For this reason, sufficient research was necessary before selecting a classification model for this project. After researching the options available using the work of Sebastiani et al[6] as a starting point, three techniques which commonly provide good results in text classification were selected to further consider, they were; K-Nearest Neighbour, Decision Tree and Support Vector Machine classification.

KNN is widely used as a text classifier because of its simplicity and efficiency. KNN is a lazy learner, noted in [6] for the way it "defers the decision on how to generalize beyond the training data until each new query instance is encountered". KNN still suffers in some respects - it assumes that training data is evenly distributed among categories, although this shouldn't be an issue for the syllabus training set as it is simply a binary classifier and providing an equal split of training data will be trivial. KNN can also suffer from inductive biases and model misfits. There has been much research into refining the KNN classifier either by using weightings to compensate for unbalanced training sets (Neighbour-Weighting KNN) [7] or by "DragPushing"[8], a method of refining the classification model by taking advantage of training errors as the model is being trained. Because of the breadth of research that has been performed into KNN as a classifier and the amount of resources, it forms a good candidate for a starting classification system for this project. This will allow the crawler and data set to be tested easily, as well as providing a baseline benchmark for the system when it is time to move forward with the Random Forest classification system for comparison.

Following this, the Random Forest classifier was considered as a classification method. Random Forests are an ensemble method of classification which consists of many Decision Tree classifiers that introduce diversity to the classification, for this reason, Decision Tree classifiers must also be understood. Their specification in [6] was used as a starting point. Decisions Trees classify text documents by building a tree in which internal nodes are labeled by terms, branches are labeled by tests on the weight that the term has in the test document and leaves are labeled by categories. In this way, a collection of terms representing a body of text can be passed through the tree in order to categorize the document. Trees are recursively generated by repeatedly partitioning the data set into classes of documents on a particular term until all training examples are classified. At some point a decision to stop using less useful terms must also be made, often on the basis of the information gain or entropy of the term. One potential downfall of Decision Tree classification is the risk of creating too large a tree and overfitting the training set. This occurs when arbitrary terms and static is used to partition sets of data into categories. This can be prevented by pruning the tree of overly specific branches as they occur within the training process. The use of Random Forest classifiers is discussed by Breiman in [9]. In this we see how the diversity introduced by creating an ensemble of many trees can create a classifier which is more robust when it comes to noise.

Finally, using research performed by Joachims in [10] as a guide, Support Vector Machines were considered as a possible classification method. SVMs have seen much success in the area of text classification. In this approach, a model is built that assigns new examples to one category or the other, in this way examples are represented as points in space, mapped out so that the examples of the separate categories are divided by as clear a gap as possible. In this sense, an SVM model is a non-probabilistic binary linear classifier, which suits this dataset particularly well. They also handle high dimensional input space well and they can make use of many features without having to remove irrelevant ones. SVMs can also handle sparse vectors effectively, still producing a reasonably accurate model [10], consisting of an entry for every term remaining in any document in the training set after preprocessing. This sparse, highly dimensional data is exactly

the type produced by the Bag of Words and tf-idf methods, and so is well suited to this type of classification. They often also contain many odd words or even have blank entries for words that haven't appeared in a particular document, but SVM classification handles these types of issues well.

2.3 Data Preparation

The data being handled must be understood and appropriately represented for all components of the system to operate correctly. This crawler will constantly be encountering URLs for both HTML and PDF document types and must be able to represent them appropriately for both training of and testing by the classifier.

2.3.1 Accessing URL content:

A URL by itself could not be used for identifying whether a document is a syllabus or not. For this reason, every URL encountered had to be downloaded and parsed based on its file type. The downloading of documents is accomplished using python's Urllib module [11]. This module provides the urlopen() method which functions similarly to the built-in python open() function, but accepts URLs rather than filenames.

Once the content of a webpage was downloaded, a method for parsing it was required. HTML documents were considered first as the majority of documents being dealt with were simple HTML and XML web pages. Of the options available for this, the Beautiful Soup [4] package was selected as it provided the most flexibility in terms of managing HTML documents and accessing various aspects of them. Using this package, links to other pages could easily be extracted by searching for their anchor tags. This was ideal for the crawler, as it relies on being able to follow links to other pages from those encountered. The Beautiful soup package also provides tools for accessing the raw text of HTML pages and stripping out HTML without affecting the remaining content. This would be essential for feature generation described in section 2.3.2.

Similarly to HTML documents, a method of handling PDF file types was needed. PDF format documents are more complex than simple HTML as various different encodings of this format have been used and so there is no all encompassing PDF extractor. Although there are many tools available which attempt to cover all bases, the most successful and widely used is the PyPDF2 library [12]. Once a URL is identified as a PDF, this library allowed it to be decoded into its text. The only major difference between this and Beautiful Soup's urlopen() is that the output of PyPDF must be checked for correctness, it occasionally tries to open a PDF file of unusual encoding and instead produces a nonsense output. Once the output authenticity is verified it is functionally the same as the output of Beautiful Soup's urlopen() and so is also ready for feature generation.

2.3.2 Feature Generation:

Once a document has been reduced down to its textual content, a method of representation which could be passed to the classifier was needed. Because every document is different, an automatic method of feature generation which can create a representation based off the text of these documents alone was required. This is a common approach to text classification which has

seen good results across many domains. One of the most common starting points when looking at automatic feature generation for text documents is the Bag of Words model [13]. This simple model represents a body of text as a vector whereby every feature expresses the multiplicity of a particular word. For this reason, there is a feature for each unique word in all documents in the input set. This creates a very sparse, highly dimensional dataset and so this must be taken in to account when selecting a classifier.

Term-frequency inverse document frequency (tf-idf) was also investigated as a method of feature generation. This is a modification of the base Bag of Words model which reflects the importance of words within a document. In this approach a set of weights is created which estimates the importance of a word based on its frequency within the document. This has the effect of reducing the bias towards larger documents which the Bag of Words model suffers from [14].

2.4 CS Education

Research in Computer Science Education has received increasing attention in recent decades. Efforts such as by Hertz [15] and Pears [16] exemplify the need to improve CS education by analyzing current approaches. Of particular interest in the CS Education community is "CS1", the generic term for a first-year introductory programming course. Specific topics such as difficulty with programming [17] and high dropout rates [18] amongst others make CS1 a focus of significant attention. If standardized sources of data such as syllabi for these modules was made available it would form the basis for a simpler approach to analyzing these subjects, getting insight into what students are actually learning and eventually improving student learning in this important and challenging field. Additionally, such sources could also help inform many other areas, within computer science, as well as in other disciplines.

As there is currently no unified source of syllabi for Computer Science modules, CS Education researchers must manually gather and source syllabi if they are interested in performing any sort of study of them as a whole. There is also no standard for storage or presentation of these syllabi. They can be HTML or PDF documents, stored institutional or non-institutional websites, faculty member personal pages, school pages, structured "module directories", programme pages etc. Navigating to a syllabus from a search engine query is often less straightforward than desired. For instance, the authors of [1] resorted to a manual search of programming module syllabi resulting in 103 syllabi from 101 institutions in 12 countries, in order to source the learning outcome statements required to carry out their work. This took several researchers a number of weeks and demanded significant coordination efforts. A tool like the one described in this report would likely have produced more results for less effort.

Although there is no currently agreed upon source or repository of CS syllabi, significant attempts at standardizing one have been made. To achieve general adoption, a baseline population of syllabi is necessary, otherwise Professors are unlikely to contribute their syllabi instead of simply publishing them on the Web erratically or not at all. Tungare et al. attempted to generate this population of syllabi by working with the results of a search to Google's search engine [19]. They manually classified 1000 documents and then used this data to train a Machine Learning algorithm to classify the remaining results of the search. The syllabi collected as a result of this work was stored as a part of the Computing and Information Technology Interactive Digital Educational Library (CITIDEL)[19], a collection of the National Science Digital Library (NSDL)[20]. This repository was sizable and managed to accurately gather many syllabi, however, there was also many duplicates and partial syllabi stored within CITIDEL and its use as a standardized reference of syllabi never caught on. It is currently no longer maintained. It was intended that the approach of incorporating a Machine Learning classifier into a Web crawler which is seeded with University

home pages would result in a more concise collection of syllabi with less duplicates and irrelevant results. The approach of using a Web crawler in such a way was intended to emulate the manual search procedure in which a programming syllabus would be found for a given institution; by going to that institution's webpage and navigating to the CS department and eventually to the module listings. This should in turn reduce the number of duplicates discovered by the system in comparison with the CITIDEL repository, which often returned duplicate syllabi or different editions of the same syllabus.

In general, syllabus classification is a task in which the community has shown great interest. Much consideration has gone into how syllabi should best be handled for this task. One approach which has seen success is the use of Automatic term recognition for extracting the unique characteristics of syllabi and Support Vector Machines (SVM) for developing the classifier [21]. As a side effect, this approach generates overviews for curricula in terms of the results of the term extraction. One significant issue that Ota et al. comments on which this approach faces is in regard to unbalanced training sets. This study was attempting to classify a set of syllabi in to several different types, resulting in some categories of training data only having one or two examples. This system should not suffer from this issue as the training set is larger with over 180 programming syllabi to train the algorithm on and simply a binary classification of either programming syllabus or not programming syllabus.

Another interesting approach to syllabus classification can be seen in the research performed by Yu et al.[22] in which a SVM classifier was used to identify syllabi. Interestingly, they also categorized their own features, such as whether the "syl" keyword appeared in the url or whether course code patterns (consisting of uppercase letters and digits) were present in the text. As an additional metric for classification, this is an interesting one which could be incorporated into the Machine Learning aspect of this project.

Similarly Matsunaga and Yamada [23] attempted to generate a syllabus database of Japanese universities using a crawler. They expanded the 649 results of a web search for syllabi to a depth of 5, giving them a total of 80446 pages. Then they performed a test on a sample of these and generated a list of keywords. These keywords were used to generate a Decision Tree which could be used to identify syllabi within the Web crawler. This approach is very similar to the proposed system and thus their forms of evaluation in terms of "Harvest Rate" (percentage of syllabus pages over all web pages collected during crawling) were certainly useful once the Machine Learning Algorithm had been trained. One potential improvement of the proposed system over this one is that the Decision Trees will developed will be based off a Bag of Words model based on the syllabi, this will allow more information to be used when classifying syllabi than would be possible by simply picking keywords.

As seen from considering the papers discussed thus far, there is still a need for a reliable source of programming Syllabi. If any standardized repository of such syllabi is to be adopted by the community, it must first be seeded with a population of syllabi. Additionally, the approach of pairing a Web crawler with a Decision Tree classification system has yet to be fully explored. I believe this methodology takes some of the more promising aspects of the papers discussed so far and should for that reason generate some useful results.

Chapter 3: **Approach**

The inception of this project was the idea that a Web crawler combined with a classifier could emulate the manual search for documents on the web. The initial approach to accomplishing this will be discussed in this chapter.

This system as a whole is designed to be flexible in its implementation. Similar to the original Web crawler: Mercator [3] which was discussed earlier, this is intended to be a base investigation into the effectiveness of a new type of system, in this case one which combines a Web crawler and a classifier. For this reason there are an infeasible number of variable and adjustable components which could not be fully explored in this study due to time constraints. Every decision adds more variability to the system and so its potential future extensibility will be documented as components are described.

3.1 Project Outline:

A study of related literature made evident that a structure encompassing the requirements and design of the proposed system would be necessary. Figure 3.1 presents the approach that the system described in this report is based on. This was based loosely on the system described by Matsunaga and Yamada in [23].

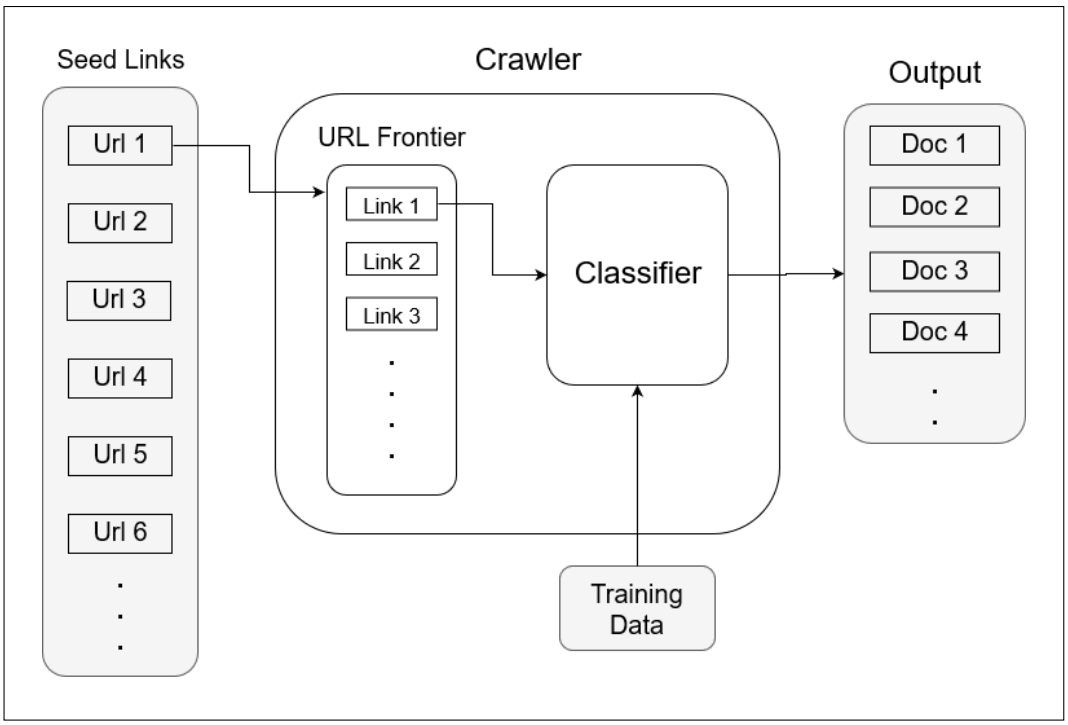

Figure 3.1: Initial system plan

This approach can be broken down into several components which will be described below.

Input:

- *Seed Links:* These will form the starting point for the system. Simply a list of URLs which each crawl will procedurally be called on. For the purposes of finding programming syllabi, the source of these seed links was the QS World University Rankings 2017 [24].
- *Training Data:* This consists of a number of examples used to train the Classifier within the crawler. These must be tailored to the type of document being searched for.

Output:

- *Found documents:* These are the pages which the crawler navigate to and the classifier identified as matching the desired type.

Crawler:

- *URL Frontier:* This structure essentially functions as a priority queue. Based off of Mercator [3], the URL Frontier is used for storing URLs which are yet to be visited. These are ordered based on a priority assigned upon their discovery. This data structure will be discussed further within chapter 4.
- *Classifier:* The component responsible for recognizing documents that match the type being searched for. Trained on the input Training data.

Once an outline of the system was in place, it was possible to abstract away these components and deal with them individually, combining them and eventually building up to the whole system as they were dealt with.

3.2 Data Preparation:

Training data, web pages encountered and the seed link inputs essentially consisted of URLs. These were useless on their own and so a system for handling and manipulating them needed to be developed before anything else could function. Depending on the situation URLs would be handled in one of two different ways: Either to access their content and find pages that they linked to, or else for representation as a vector to train or be tested by the classifier.

3.2.1 Content access:

Every page encountered by the crawler is downloaded using python's Urllib library [11]. Urllib is a standard tool used in python for this purpose. Although several alternatives were considered, Urllib offered the most flexibility in terms of detecting HTTP errors and status codes and handling them.

After being downloaded, the pages are then then parsed depending on the document type. For HTML pages, the Beautiful Soup library [4] was used. This is a simple module that pulls data

out of HTML and XML files once they have been retrieved. This allows access to all of the URLs linked to by web pages and so serves for this aspect of the URL handling. These extracted links can be added to the list of URLs yet to be crawled. Beautiful soup also allows for easy removal of HTML and extraction of text content. This is essential for the feature generation stage described below.

For PDF documents, the Py2PDF2 [12] module was used. Extraction of text from a PDF is more complex than the HTML counterpart. PDF files often employ different encodings so any implementation involving reading PDFs required sufficient error checking and handling to ensure that a poorly read PDF didn't end up generating nonsense features which could affect the training of the classifier.

3.2.2 Feature Generation:

Both pages used to train the classifier and pages encountered by the system had to be represented in a way that the classifier could handle. A simple vector representation was chosen as it easily allowed for tweaking for further experimentation. The process for preparing a document was as follows:

1. Initially, downloaded pages were stripped of their HTML. This was done because a classifier which evaluated only word content would allow for easily classifying both HTML and PDF documents on the same basis. Handling both document types individually was considered initially. This could allow the classifier to make use of the HTML content of web pages but in the end was deemed nonessential to the system's requirements and was left for future investigation.

2. Next, punctuation and numbers were removed. This was again to simplify the complexity of the resultant vectors. The vectorzation method used is based off of word frequency so the presence of punctuation or number frequency was expected to increase the dimensionality of the feature space excessively.

3. The remaining body of text was then tokenized into a vector of all the individual words. This was required for the feature generation method employed.

4. Finally, stop words were removed. The stopwords removed were sourced from python's Natural Language Toolkit (NLTK)[25] library and consists of 153 words. NLTK's standard English corpus of words was chosen as a simple starting point. Experimentation with different selections of stopwords is left as future work for this project.

After this process was complete, the remaining vectors consisted of only words (excluding stopwords) that were found in the documents. These then needed to be represented in such a way that could be used to train the classifier. As a starting point, the Bag of Words model (discussed in section 2.3.2) was used for this. This model was applied using the CountVectoriser method from the sklearn python package for Machine Learning. Taking this a step further, term frequency-inverse document frequency (tf-idf) experimented with (also discussed in section 2.3.2). Similarly to Bag of Words, tf-idf was applied using the sklearn package, particularly the TfidfVectoriser method. Both these methods were limited to creating a maximum number of 5000 features representing the 5000 most common remaining words present in the input documents after they were prepared. This was performed to create less sparse resultant set of feature vectors.

Once this pipeline for preparing documents was decided, the rest of the system's architecture could be implemented. Chapter 4 will detail the functionality and structure of the system developed and Section 4.5 in particular will detail some of the issues encountered and decisions made while developing them.

Chapter 4: **Detailed Design and Implementation**

In this chapter, the specifics of various components of the system will be detailed. First, the classification methods explored and their implementation within the system will be described. Then a path from the seed link through to the system's output will be traced, noting all of the components involved along the way. Finally the various parameters and settings available within the system which can be used to tailor an experiment will be outlined. As various components of this system were implemented, decisions had to be made as to how this was done and so there are many areas which could be explored further. Support for future extensibility and customizability will be discussed thoroughly in Chapter 6 but will also be briefly mentioned in this section as decisions about the implementation give rise to these variables.

4.1 Crawler

The crawler aspect of this system is what makes it unique as an approach to sourcing sets of particular documents. It allows the system to emulate the manual search performed by a person if they were trying to source documents in this manner. It provides the crawler with a more targeted and directed set of test documents while excluding many of the extraneous results of a search engine query. The implementation of the crawler defines its ability to do this and so various components of its structure will be expanded in this section.

4.1.1 URL Frontier

As described by Heydon and Najork in [3], a URL Frontier was developed for storing the URLs that have yet to be crawled. It functions as a priority queue with priorities assigned to every link within. Pages are continually removed from the URL frontier, any URLs they link to are assigned a priority according to the URL reordering mechanism and added into the URL Frontier. Then the page itself is classified and its URL added to the list of already crawled pages. The next URL is then taken from the Frontier and the process repeats.

Because Python has no built in definition of a priority queue, a simple MyPriorityQueue class was developed to serve as the URL Frontier. It stores all URLs yet to be crawled in order of their priority, allowing the addition of new URLs as well as the returning of the URL with the highest priority. It also allows returning of the N highest priority URLs stored. This functionality was used for generating the initial set of negative examples for training the classifier. Running the system without any classifier and taking a sample of the links which it came across, then specifically selecting ones that weren't of the type being searched for provided a set of negative examples that were likely to be ranked highly by the system and thus evaluated by the classifier.

4.1.2 URL Reordering

Within the URL Frontier, a method of assigning priorities to guide the URL reordering was required. This assignment would direct the system and dictate what pages are explored and thus classified. As discussed within the chapter 2, there are several methods available for this purpose. Of those proposed by Cho et al.[5], the simple location metric was chosen for this system. In this, priorities are assigned to a URL based simply on the URL itself rather than the content of the page it references. This is done by searching the URL for keywords specific to the document type being searched for. For example, in the implementation searching for syllabi for programming modules, higher priorities were given to URLs with "syl", "syllabus" or "programming", while priority was reduced for URLs containing keywords such as "biology", "chem", "twitter" and "facebook". The use of keywords in this manner made the system flexible as, for example when it was noticed that the crawler was diverging from the domain of University pages into social media, URLs mentioning the names of these social media could be penalized more significantly.

4.2 Classifier

The classifier is a core component of this system and is trained on an example set of data that is used to further identify similar documents encountered during a crawl. Clearly, further investigation into the performance of particular classifiers with this sort of application could reveal more optimal combinations. For the purpose of this study, two were selected for comparison. Of the options considered during Background research the Random Forest and Support Vector Machine classifiers were selected to experiment with.

The implementation of both of these classifiers was performed using the sklearn[26] toolkit which is an open source python library that provides simple and efficient tools for data mining and data analysis. Both a RandomForest and SVM classes were created to handle testing and training of the classifiers. Either one can be specified when the system being run. Actual training within a run occurs immediately before a crawl begins, taking in the training set of documents. The documents are converted into either the Bag of Words or tf-idf representations depending on what is desired and the classifier is trained on them. There is also functionality to simply test the specified classifier using 5-fold cross validation, outputting all results and measures of performance within this test to a results file storing: Precision, recall, f1-score, true positive, false positive, false negative and true negative rates. The training of the classifier within this class returns a trained model capable of providing a prediction as well as a vectorizer object which is used by the crawler to represent documents it encounters as vectors for their prediction.

4.2.1 Random Forest

Random Forest classifiers are robust and versatile classifiers and so were chosen as a starting point for classification. Although Decision Trees generally don't handle sparse data well, after some experimentation, the Random Forest ensemble method proved to be reasonably successful when the feature generation process was limited to a smaller size of 5000 features as was mentioned in the previous chapter. Random Forest classifiers are also eager learners which was also considered a benefit initially as there was concern that the classification process would bottleneck the system's throughput. It was the URL retrieval which later turned out to be the bottleneck in the system however, so this factor wasn't as relevant in the end.

4.2.2 Support Vector Machine

The Support Vector Machine classifier was chosen as a comparison with the Random Forest classifier because it is regarded as well suited to text classification [10]. As mentioned in Background research, SVM is well suited to sparse and highly dimensional data, which is exactly what is produced by the Bad of Words and tf-idf models. SVM also relies heavily on a sufficiently large set of training data, this is a key difference between Random Forest and SVM classifiers which played a large part in the results discussed in chapter 5.

4.3 Operation trace

In this section the operation of the system will be explained in a chronological order, assuming that the classifier has been trained as described above and the crawler is given an input of seed urls. For each one, the following procedure is executed.

1. *A seed link is selected.* From the input list of seed links, one is chosen. This page is parsed and the title of the page is extracted and added to the list of words used for URL Reordering later. All other links from this page are then extracted, their priority assigned based off the URL Reordering Mechanism described earlier, and added to the URL Frontier.
2. *Crawling process is performed.* Pages are continually taken from the top of the URL Frontier. They are checked to see that they haven't been encountered before and if not: they are parsed depending on whether they are PDF or HTML format, tested by the classifier and added to a list of found matching documents if they are classified as such. All links to other pages are extracted, their priorities calculated and then they are also added to the URL Frontier. This step is repeated until an end condition is met.
3. *Crawling of a seed link ends.* Once an end condition is met, the crawling ceases. All found pages are continually output to a csv and a json file, linked with their starting seed page. This is done to provide some insight to the origin of found pages, whether certain domains have pages that are more easily mistaken as the target document or if certain types of seed pages are more useful. These output files also act as a checkpointing system so that, crawling of a set of seed links can be restarted from a certain point if the system crashes or further crawling is needed. After this, provided there is another uncrawled seed page, the process goes back to step 1.

4.4 Structure of Experiments

Once all components of the system were integrated into a single system, various settings and parameters were made available to easily tailor an experiment. A run file was made for each experiment specifying the parameters and inputs for a given one. This allowed each experiment, its inputs, measures of performance and outputs to be maintained individually for analysis. The various inputs and options were as follows:

Input variables:

- *Positive_urls:* Location of the positive URL examples to be used for training.

- *Negative_urls:* Location of the negative URL examples to be used for training.
- *Dataset_path:* Destination of a CSV file containing all examples parsed and coupled with their class identifier.
- *Output:* Destination of the output CSV and Json files, containing all found documents and their origin seed pages.

Boolean modifiers:

- *Crawl_PDFs* To determine whether PDF documents should be parsed and evaluated or skipped.
- *Build_set* To determine whether this is an initial run or a continuation of a previous one (thus requiring loading of the previous state).
- *Test_classifier* To determine whether the classifier should be tested and its performance output to a file.
- *Svm_over_rf* To determine whether the Support Vector Machine or the Random Forest classifier is used in this particular experiment.
- *Tfidf_over_bow* To determine wheter tf-idf or Bag of words model is used to represent the documents for training and testing.

Variables:

- *Max_links:* Determines the maximum number of links leading out of a page to extract and add to the URL Frontier. Links are selected in order of the prioritiy assigned to them.
- *Max_pages:* The number of pages to be crawled from a given seed link.
- *Max_syllabi:* The maximum number of documents identified as a programming syllabi to gather from a seed before ending the crawl from that seed. This is an additional end case that could be useful for either ending crawls with classifiers that recognise far too many matches (ending them early) or crawls that recognise very few (in this case allowing them to crawl until it at least finds something).

Every experiment has its own run file specifying each of these accompanied with its output and performance statistics.

4.5 Challenges Faced

The implementation and evaluation of this type of system raised many challenges which will be detailed in this section.

Accessing and handling different filetypes was one of the major issues which this project had to overcome during implementation. Firstly, downloading of web pages using the urllib [11] module was not consistently successful. Slow servers and status codes being returned from HTTP requests meant that pages often couldn't be downloaded and parsed. Several alternatives were considered before urllib in an attempt to reduce the number of these errors but none offered as much flexibility in terms of handling them. This module allowed failed pages to be retried a specific number of times before moving on as well as setting of maximum values on time waiting for a response.

Once pages were downloaded, handling them was often not trivial, particularly in the case of PDF documents. As mentioned earlier, various encodings used when creating PDF files means that there is no universal PDF parser. When implementing this functionality, this required much consideration in how to detect PDF files which were parsed incorrectly so that they would not be used to train the classifier, which would essentially introduce nonsense examples to the training set.

When it came to the evaluation of this system, the large number of documents being downloaded and tested resulted in long running times, often exceeding five to eight hours. This made it very difficult to fully evaluate the effects of all of the possible settings and parameters. For this reason, focus was placed on evaluating the impact of the classifier on the overall output of the system. Investigating other areas further is left to future work which is detailed in Section 6.2.

An issue which is common to crawlers is their potential to impact network bandwidth and impact on web servers. This is discussed further by Bal in [27]. Although this must be considered if comprehensive and deep crawls are being performed, because this project is an investigation into a new type of system and all experiments involved sampling a limited number of pages from any domain, the impact of this was not significant. This is also an issue which is easily dealt with by staggering HTTP requests between several different domains rather than sequentially targeting the same one.

Chapter 5: **Evaluation**

As discussed earlier, there are many interchangable components of this system which could improve or reduce its performance. For the purpose of evaluation, three were chosen and experimented with to see their affect on the overall output of the system. They were: the choice of classifier ,the method of document representation for the classifier and the training set for the classifier. Experiments were performed to see how various combinations of these factors would affect the output of the system.

For each experiment, the system was run on a set of 240 seed links, crawling exactly 100 pages from each for total of approximately 24,000 classifications of pages (this number is approximate because pages occasionally return HTTP status codes on retrieval and therefore cannot be accessed). Each experiment took approximately 5-8 hours to run to completion, because of this, running time was a limiting factor in the extent to which the system could be tested. The focus of experiments was on the effect of the classifier and its ability to determine the output of the system as a whole. Given more time, further experiments could be performed to find out exactly what the other optimal settings and components of this system are. This is discussed further in chapter 6.

5.1 Evaluation Metrics

Evaluating the system's performance was an important aspect of this project. There are two main views of evaluation used in this section: Evaluation of the classifier and it's training, and evaluation of the system as a whole.

Evaluation of the classifier was a trivial task. The sklearn library provides simple tools for performing cross fold evaluation of a classifier on a training set. This feature was used on each classifier trained, performing 5-fold cross validation and calculating the average precision and recall across all folds of the classifier's performance on the training data.

Evaluation of the system as a whole was more complex. It was running on thousands of unlabeled examples, encountering approximately 24,000 pages in each experiment. For this reason, a method of estimating the system's precision and recall were needed. For calculation of precision, a random sample of 100 of the pages which the system found and identified as programming syllabi were gathered. They were then manually checked to see how many of them were actually programming syllabi, this number was taken as the precision of the experiment. For recall, an estimate of the total number of syllabi within the sample space was needed. To calculate this, a sample of 300 of of the 24,000 pages encountered was taken. Again these were manually inspected to find what percentage of them were actually programming syllabi. The result of this was 1%, this would suggest that 240 of the 24,000 pages are in fact programming syllabi and so this value could be used to calculate the estimated recall and f1-score of the system. It was interesting to note the value of URL reordering when manually expecting this sample, almost all pages encountered by the crawler were Computer Science or Syllabus like. The ability of the simple location metric employed to direct the crawl towards more relevant areas was remarkable.

5.2 Results

The following pipeline was created for generating training sets for the classifiers:

1. *Non-PDF simple:* The non-PDF syllabi paired with a random selection of non-PDF, non-syllabi examples generated by running the crawler without a classifier and selecting pages that weren't syllabi.
2. *Non-PDF improved:* The non-PDF syllabi paired with a set of false positives randomly selected from the output of 1. The intention of this was to see if the accuracy could be improved by training the classifier to better identify the pages that 1 classified poorly.
3. *PDF inclusive set:* All syllabi examples paired with a selection of non-syllabi examples generated by running the crawler without a classifier and selecting pages that weren't syllabi. The intention of this set was to see if increasing the size of the training set by incorporating PDF examples would improve the classification accuracy.

The effect of the classifier was examined first with the simple Bag of Words model being used for document representation. The effect of the classifier was examined first with the simpler Bag of Words model being used. The results of these experiments can be seen below in table 5.1.

Table 5.1: Random Forest with Bag of Words

Experiment		Classifier Precision	Classifier Recall	# "syllabi" returned	System precision	System Recall	System f1
1	Non-PDF simple	0.947	0.878	628	9%	23.6%	13%
2	Non-PDF improved	0.89	0.9	7962	<1%	<33.2%	<2%
3	PDF inclusive set	0.89	0.975	5449	<1%	<23.1%	<1.9%

As these results show, the Random Forest classifier's performance did not improve when introducing the more tailored negative set in the second experiment. Similarly, introducing the extra PDF examples in 3 also decreased the overall quality of the output. The high precision and recall of the classifier seem to indicate that overfitting was an issue for these examples, this is reflected in the very low precision of the overall system, producing an output with a very high true positive rate. The best overall results are seen with the simplest training set of just HTML examples.

Next these experiments were repeated with the Support Vector Machine classifier being used to identify programming syllabi and the same Bag of Words model. The results are shown in table 5.2.

Table 5.2: Support Vector Machine with Bag of Words

Experiment		Classifier Precision	Classifier Recall	# "syllabi" returned	System precision	System Recall	System f1
1	Non-PDF simple	0.87	0.866	3140	6%	78.5%	11.1%
2	Non-PDF improved	0.85	0.833	5434	2%	45.3%	3.9%
3	PDF inclusive set	0.88	0.975	2774	5%	57.8%	9.2%

As with the Random Forest classifier, the second experiment where negative examples were improved by taking the False positives of the first resulted in a lower overall accuracy from the system. This was perhaps because it made the positive and negative examples more similar and resulted in the system misclassifying pages as syllabi when they weren't. Unlike the Random Forest however, SVM benefitted more from the addition of the PDF data, achieving a precision of 5%. Still the best outcome was again the simplest case of HTML syllabi examples paired with a random selection of non-syllabi from university websites.

Moving on from this, the impact of representing documents using term frequency-inverse document frequency was investigated. All 6 experiments were repeated with the same training sets represented as tf-idf vectors rather than Bag of Words. Although tf-idf generally improves performance in text classifiers over Bag of Words, the affect of the limited size of the training sets appeared to be compounded by this representation. The results of the experiments were unanimously worse than in the Bag of words experiments, apart from one exception: the non-PDF simple experiment with Random Forest classifier. The results of this case are detailed below.

Table 5.3: The effect of tf-idf on RandomForest

Experiment		Classifier Precision	Classifier Recall	# "syllabi" returned	System precision	System Recall	System f1
1	Bag of Words	0.947	0.878	628	9%	23.6%	13%
2	Tf-idf	0.933	0.836	660	29%	79.8%	42.5%

As table 5.3 shows, term frequency-inverse document frequency had a considerable impact on the performance of the Random Forest Classifier. Identifying only 660 pages of the 24,000 classified as programming syllabi, with a precision of 29% and a recall of 79.8%, this combination far outperformed all other experiments. Looking at the classifier performance in 5-fold cross validation, we see slightly lower scores for precision and recall, this could possibly indicate less overfitting which may explain the lower number of true positives identified by the system as a whole when compared to the other experiments.

Chapter 6: **Future Work and Conclusions**

6.1 Requirements Met

The project described in this report satisfies all core requirements of the project specification:

- A Web crawler capable of systematically navigating through a website, finding all links and searching them efficiently.
  - See Section 4.1 on the crawler specification.
- A process for parsing these pages and preparing them for input into a Machine Learning classifier.
  - See Section 3.2.1 for details on how pages were parsed.
- Use of the Bag of Words model to prepare documents for training and testing the classification algorithm.
  - See Section 3.2.2 on how parsed documents were prepared for classification.
- Validation of the information retrieval and Machine Learning system to identify documents that show a high likelihood of matching the specified type.
  - See Chapter 5 for details of system validation.

Further, the following aspects of the advanced goals were met:

- Determining if other methods can improve the classification performance:
  - Random Forest and Support Vector Machine classification were both implemented and evaluated. See section 4.2.
- Extending the classification to work with other file types (e.g. PDF, docx, etc.).
  - The classification was extended to work with the PDF file type. See section 3.2.1. This was chosen because syllabi are frequently stored as such and it would be important to support this file type for finding them. Other formats were not used sufficiently to justify adding more support, but should they be needed it would not be overly complex to add functionality for this.

6.2 Future Work

As this system was a case study for a new type of document retrieval system which combines a Web crawler with a classifier in the manner described, it was not feasible to comprehensively test every possible component. The focus of evaluation in this case was on the impact of the classifier on overall performance, looking at two particular types of classifier and how document representation and choice of training set affects them. However, there are many other aspects of this system which could be altered to manipulate the way it runs and possibly improve its performance. With this in mind, the extensibility of the various components of this system will be noted here.

- Classifier: Only Random Forest and SVM classifiers were considered here. There are many other options which could be experimented with to get more promising results. Even the use of a neural network such as Word2Vec which could take into account the context of words within a document and cluster documents appropriately. Alternatively, a classifier which provides a probability of being the specified document could be used to guide the classifier, potentially following links from pages that are more likely related to the document being searched for.

- URL Reordering: Much more experimentation with the URL reordering technique being used is possible. Alternate sets of keywords and different weightings given for the presence of those keywords could potentially guide the crawler in completely different directions, exposing it to different pages more quickly. Completely different methods of reordering the URLs within the URL Frontier could also have promising results, for example the use of PageRank, as discussed in [5].

- Depth of Crawl: In the experiments described in chapter 5, shallow crawls were performed, sampling only 100 documents crawled from each seed page. The effect of the depth of crawls has yet to be fully explored. Crawling deeper could result in encountering desired pages that were entirely missed, similarly, if the URL reordering was truly effective, and desired pages were mostly present in this shallow search, then crawling further would likely result in only generating more false positive results.

- Training dataset size: Sourcing more training data would almost certainly result in improving the performance of the classifiers. Particularly the SVM classifier, which is known to suffer from smaller training sets. If more data were available, there would likely be less overfitting and so a lower false positive rate from the overall system.

- Alternate case studies: Testing this system by searching for programming syllabi was just one use case. It could be potentially useful to test the system in other use cases, such as instruction manuals, CVs, resumes or syllabi for other areas of education. Doing this could reveal aspects of the system which were not highlighted in this example.

- Document representation: There are many other methods besides tf-idf and Bag of Words which can be used for feature generation of text documents for training of classifiers. Similarly, the preprocessing steps to applying theses models are also tailorable, for example, modifying the list of stop words removed from documents before vector representation or including numbers and punctuation. Experimenting with various representations of text documents could result in better classification and thus performance of the overfall system.

- System efficiency: The operations performed from each seed page are fully independent, for this reason it would be very straightforward to make the system run in parallel. Having several crawlers simultaneously running, crawling from different starting points and then combining their output could easily reduce the running time of a full crawl by a significant factor.

- Load on web servers: This system was tested using a shallow search, taking a sample of 100 pages linked from each seed page. Because of this, the load placed on servers was negligible. If the system were to be run to a much greater depth and could potentially be accessing many more pages from a specific web server, steps should be taken to reduce the load. For example, a technique suggested in [3] proposes simultaneously crawling multiple sites and alternating HTTP requests to them in order to reduce load on any specific one.

6.3 Conclusion

The ability to source a large collection of similar documents on the web is an important one. Although many tools exist for this purpose they fall short in certain areas, particularly when looking for a large set of documents of a very narrow type. The system discussed in this report endeavors to accomplish this by combining a Web crawler with a Machine Learning classifier in an attempt to emulate the manual search which a human would perform when sourcing documents in this way.

Although the results observed in chapter 5 were not as significant as initially hoped, it's important to note that, in the case of programming syllabi at least, the number of these documents on the World Wide Web is very small, surely representing a miniscule fraction of web pages. The system described in this report can, given a set of 180 examples and a set of starting points, generate a list of pages which are 29% programming syllabi, a percentage that is orders of magnitude larger than is present in the web. The extensibility and potential for future improvement of this system has also been detailed throughout this report. Further research into finding the most optimal settings and components could result in a more flexible and useful overall tool.

In the area of Computer Science Education Research, this system is already potentially useful. Compared to the approach adopted when searching for syllabi during the research which motivated this project, running this system to a greater depth and manually checking its output would be a far more efficient method of sourcing these syllabi. Using this system to source other document types such as instruction manuals for specific appliances for instance, would require minimal work, and should offer similar results.

Bibliography

[1] Andrew Luxton-Reilly, Brett A Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, page 388, New York, NY, USA, 2017. ACM.

[2] D. Eichmann. The RBSE spider Balancing effective search against Web load. *Computer Networks and ISDN Systems*, 27(2):308, 1994.

[3] Allan Heydon and Marc Najork. Mercator: A scalable, extensible Web crawler. *World Wide Web*, 2:219–229, 1999.

[4] Leonard Richardson. Beautiful Soup Documentation. *https://www.crummy.com/software/BeautifulSoup/bs4/doc/*, pages 1–72, 2016.

[5] Junghoo Cho. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1-7):161–172, 1998.

[6] Fabrizio Sebastiani and Consiglio N Delle Ricerche. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[7] Songbo Tan. Neighbor-weighted K-nearest neighbor for unbalanced text corpus. *Expert Systems with Applications*, 28(4):667–671, 2005.

[8] Songbo Tan. An effective refinement strategy for KNN text classifier. *Expert Systems with Applications*, 30(2):290–298, 2006.

[9] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 10 2001.

[10] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1398:137–142, 1998.

[11] Python Software Foundation. Urllib Documentation. *https://docs.python.org/2/library/urllib.html*, 2018.

[12] Phaseit. PyPDF2 - A python pdf data extractor. *https://pythonhosted.org/PyPDF2/About PyPDF2.html*, 2016.

[13] Huan Liu and Hiroshi Motoda. *Computational methods of feature selection*. CRC Press, 2007.

[14] Gerard Salton and Christopher Buckley. Term Weighting approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[15] Matthew Hertz. What do CS1 and CS2 mean? Iinvestigating differences in the early courses. *41st ACM Technical Symposium on Computer Science Education*, pages 199–203, 2010.

[16] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. In *ACM SIGCSE Bulletin*, volume 39, page 204, New York, New York, USA, 2007. ACM Press.

[17] Ellie Lovellette, John Matta, Dennis Bouvier, and Roger Frye. Just the Numbers: An Investigation of Contextualization of Problems for Novice Programmers. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 393–398, New York, NY, USA, 2017. ACM.

[18] Brett A Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. Effective compiler error message enhancement for novice programming students. *Computer Science Education*, 26(2-3):148–175, 2016.

[19] Manas Tungare, Xiaoyan Yu, William Cameron, GuoFang Teng, Manuel A. Pérez-Quones, Lillian Cassel, Weiguo Fan, and Edward A. Fox. Towards a syllabus repository for computer science courses. *SIGCSE 2007: 38th SIGCSE Technical Symposium on Computer Science Education*, pages 55–59, 2007.

[20] Carl Lagoze, William Arms, Stoney Gan, Diane Hillmann, Christopher Ingram, Dean Krafft, Richard Marisa, Jon Phipps, John Saylor, Carol Terrizzi, Walter Hoehn, David Millman, James Allan, Sergio Guzman-Lara, and Tom Kalt. Core Services in the Architecture of the National Science Digital Library (NSDL). In *Proceedings of the 2Nd ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '02, pages 201–209, New York, NY, USA, 2002. ACM.

[21] Susumu Ota and Hideki Mima. Machine learning-based syllabus classification toward automatic organization of issue-oriented interdisciplinary curricula. *Procedia - Social and Behavioral Sciences*, 27:241–247, 1 2011.

[22] Weiguo Fan Yubo Yuan Manuel Pérez-quiñones Edward A. Fox William Cameron Lillian Cassel Xiaoyan Yu, Manas Tungare. Automatic Syllabus Classification using Support Vector Machines. In *Handbook of Research on Text and Web Mining Technologies*, page 14. 2009.

[23] Yoshihiro Matsunaga, Shintaro Yamada, Eisuke Ito, and Sachio Hirokawa. A Web Syllabus Crawler and its Efficiency Evaluation. In *International Symposium on Information Science and Electrical Engineering 2003 (ISEE 2003)*, pages 565–568, 2003.

[24] QS. QS World University Rankings. *https://www.topuniversities.com/university-rankings/world-university-rankings/2016*, 2017.

[25] NLTK Project. Natural Language Toolkit. *https://www.nltk.org/*, 2017.

[26] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, BBertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Satinder Bal Gupta. The Issues and Challenges with the Web Crawlers. 1(1):1–10, 2015.