

# ECM3401: Individual Literature Review and Project Design and Implementation of a Naive Bayes Based Focused Crawler

April 28, 2019

## Abstract

In the modern age, the World Wide Web (WWW) has grown to become the largest source of unstructured data on the planet. Empowering individuals to be able to find the information one is searching for has become a fundamental problem faced by those working in the industry and conducting research. Web crawlers provide a potential solution to this problem. However, due to the growing and ever-changing nature of the WWW, it has now become unwise to try to crawl the entire WWW. Focused Crawlers provide an alternative to crawling the entire WWW by collecting web pages that are relevant to a specified topic. These types of crawlers need a way of determining the relevance of a web page. Machine learning has a long-documented success in producing classifiers in other applications. This project aims to create a focused web crawler that harnesses the power of machine learning in order to achieve high accuracy in determining whether a web page is relevant to a given topic. This report presents the design, implementation, testing and evaluation of the focused web crawler. Research showed that the Nave Bayes classifier on average produced the best results in focused web crawling so was selected for use in this project. The resulting classifier produced an 80% accuracy rate at recognising whether a web page was related to a given topic. The project was successfully developed into a web application using the Django web framework which helped to visualise the inner workings of the web crawler.

*I certify that all material in this dissertation which is not my own work has been identified.*

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Aims and Objectives . . . . .	2
1.2	Report Overview . . . . .	2
<b>2</b>	<b>Summary of literature review and specification</b>	<b>3</b>
2.1	Summary of the Literature Review . . . . .	3
2.2	Project Specification . . . . .	4
<b>3</b>	<b>Design</b>	<b>5</b>
3.1	Web Crawler Design . . . . .	5
3.1.1	Machine Learning Classifier Design . . . . .	6
3.2	Web Application Design . . . . .	8
3.2.1	Model . . . . .	8
3.2.2	View . . . . .	9
3.2.3	Controller . . . . .	9
3.3	Process . . . . .	10
<b>4</b>	<b>Development</b>	<b>10</b>
4.1	Web Crawler Development . . . . .	10
4.2	Machine Learning Classifier Development . . . . .	12
4.2.1	Data set Development . . . . .	12
4.2.2	Naive Bayes Classifier Development . . . . .	13
4.3	Web Application Development . . . . .	15
4.3.1	Django Admin and Models . . . . .	16
4.3.2	Commands . . . . .	17
<b>5</b>	<b>Testing</b>	<b>18</b>
5.1	Classifier Performance . . . . .	18
5.2	Web Crawler Performance . . . . .	19
5.3	Other Testing . . . . .	20
<b>6</b>	<b>Description of the Final Product</b>	<b>20</b>
<b>7</b>	<b>Evaluation of the Final Product</b>	<b>21</b>
7.1	Aims and Requirements . . . . .	21
7.2	Results . . . . .	22
7.2.1	Classifier Performance . . . . .	22
7.2.2	Web Crawler Performance . . . . .	23
7.2.3	Other Testing . . . . .	25
7.3	Comparison with Other Web Crawlers . . . . .	26
<b>8</b>	<b>Critical Assessment of the Project as a Whole</b>	<b>27</b>
8.1	Future Work . . . . .	28
<b>9</b>	<b>Conclusion</b>	<b>28</b>

<b>Appendices</b>	<b>33</b>
<b>A Kanban board during Web Crawler Project Development</b>	<b>33</b>
<b>B Django Admin Login Page</b>	<b>34</b>
<b>C Model for the <i>WebsiteTrainingMapping</i> table</b>	<b>35</b>
<b>D ModelAdmin for the <i>WebsiteTrainingMapping</i> table</b>	<b>36</b>
<b>E Web Application Main Page</b>	<b>37</b>
<b>F Websites Page</b>	<b>38</b>
<b>G Edit Website Entry Page</b>	<b>39</b>
<b>H Cross Browser Compatibility Checks</b>	<b>40</b>
<b>I User Acceptance Form</b>	<b>41</b>
<b>J Food Recipe Training Set</b>	<b>42</b>
<b>K Food Recipe Test Set 1: Random URLs</b>	<b>45</b>
<b>L Food Recipe Test Set 2: Food Recipe URLs</b>	<b>47</b>

# 1 Introduction

The World Wide Web (WWW) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents[1]. In other words, the Web is an interconnected computer network that allows users of one computer to obtain information stored on another computer over the internet. Navigating through the Web is made possible through the use of a client program called a web browser, e.g. Google Chrome or Internet Explorer. Web browsers work by sending requests to remote servers for information. The server sends back a response to the web browser which interprets the result. The returned documents are written in HTML and is responsible for laying out the text and graphics on the users computer screen.

In modern life, the use of the internet is growing exponentially. The WWW provides a vast source of information of almost all type. Nowadays people use search engines every day as large volumes of data can be explored quickly enabling users' to extract valuable information from the web. The web now contains more than one hundred and thirty trillion publicly indexed web documents spread all over the world on thousands of servers[2]. Being able to run a search that encompasses all Web Servers and pages is not realistic. Every day a large number of pages are added to the internet, or their location changed leading to changes in the state of the internet.

A web crawler is a program or automated script which surveys the World Wide Web in a systematic, automated manner. This process is called web crawling. Search engines are unable to keep up with the changing nature of the internet. Due to this fact, search engines are reliant on web crawlers for the collection of required pages[3]. Many online websites, in particular, search engines, use web crawling as a means of providing up to date information. Web crawlers are primarily used to produce a copy of all the visited pages for later processing by a search engine[4]. Search engines will then index the downloaded pages to provide fast searches. Further uses of web crawlers include website maintenance tasks, such as checking if links still work or validating HTML.

The most famous example of a web crawler would be Googles Googlebot. Googlebot is used in a fashion similar to that described above. However, due to the web crawler being an integral part of Googles business, there is very little information regarding exactly how Googlebot operates. Due to Googles monopoly over this area, there is not much research into web crawling, and open source alternatives are scarce. One open source alternative is Heritrix. However, Heritrix is built for web archiving, and the last stable release was in 2014. This paper aims to detail a project which combines machine learning and web crawling to create a focused web crawler. Machine learning is a topic at the forefront of Computer Science, and its application has proved successful in a multitude of fields.

The project aims to bring together machine learning, an area at the forefront of Computer Science, and web crawling. With the ultimate goal of creating a machine learning based focused web crawler. The project has three main components. The first part is the web crawler project which is in charge of all web crawler functionality such as downloading web pages. The second part is the machine learning classifier which is responsible for identifying pages relevant to the topic. The final part is the web application which pro-

vides an interface for controlling the web crawler and helping to visualise the results. The combination of these three components forms the entire project.

## 1.1 Aims and Objectives

The following are the aims of the project. The aims are in order of importance, with the first aim being the main aim of the project.

1. To combine machine learning and web crawling to create a highly accurate focused crawler.
2. Develop a web application that can house the web crawler to help users better understand how the crawler functions.
3. Bring light to an area of Computer Science that is dominated by large multinational conglomerates.

These aims are realised through the following objectives.

- Pages are downloaded and moulded into a state that can be utilised by the classifier.
- The machine learning classifier builds on Bayesian Theory to perform text classification.
- Data sets can either be created or extracted from the WWW.
- A web application framework is implemented, allowing any individuals to access the project.

## 1.2 Report Overview

The report focuses on the design, implementation and testing of the focused web crawler.

1. First, a summary of the literature around the topic of web crawling and machine learning is presented.
2. Next, a specification detailing the functional and non-functional requirements are listed.
3. Following on from this, the design illustrates how the specification is achieved. The design outlines all desired functionality and the different components needed to build the project.
4. The development section justifies any modifications from the original specification and explains technology choices. Along with presenting the methodology used to deliver the project.
5. A section on testing outlines the tests carried out ranging from classifier performance to web crawler performance.

6. Following this is a section describing the final product.
7. Then the next section evaluates the final product against the specification, presenting all results from tests to justify any claims made.
8. A section detailing a critical analysis of the project as a whole, outlining any future work that may be undertaken.
9. Finally, a conclusion which provides a final reflective overview of the overall project.

## 2 Summary of literature review and specification

This section contains a summary of the existing research that has been carried out into using machine learning within web crawling with a project specification.

### 2.1 Summary of the Literature Review

Although this project is centred around focused crawlers, there are other types of crawlers. The first type of crawler was a universal crawler. These crawlers came about when the internet was much smaller than today, in the late 90s. These crawlers took advantage of traditional graph searching algorithms such as breadth-first and depth-first search[5]. The goal of these crawlers was to crawl the entire internet[6]. With the growth of the internet, this became unfeasible, hence the rise of the focused crawler. Focused crawlers crawl the internet to collect topic-specific web pages[7] selectively. Machine learning can be used to classify these web pages into relevant and non-relevant web pages.

A wealth of problems utilise machine learning techniques; however, it is still relatively untested in the field of web crawling. Research showed that machine learning techniques are split into supervised and unsupervised learning algorithms[8]. Supervised learning algorithms are trained using labelled training data allowing the algorithm to check whether or not their classification was correct[9]. Unsupervised learning algorithms explore the data to try to draw an inference from the training data[10]. This training data is not labelled. Unsupervised learning algorithms include K-means clustering and neural networks. These algorithms are used when labelled training data is not available or if individuals are unsure about trends within the data and would like the algorithm to determine if there is one[10]. However, there are several training sets available for web crawling that can be used to train the machine learning model. These training sets are comprised of web pages that have been downloaded and come in a variety of languages and on a variety of topics.

For this reason, supervised learning algorithms would, therefore, be more appropriate for this project. The first supervised learning algorithm looked at, was a probability-based classifier, namely Nave Bayes. This classifier has been applied to problems such as judging documents to determine if they are spam emails and has proved to be highly successful[11]. Classifying spam emails is a similar application to this project. The simplicity of this algorithm means that this algorithm is both fast and efficient. Decision Trees are another supervised learning algorithm[12] which splits the training data based on different attributes displayed by the training data. Features such as pruning [13] have allowed this algorithm to grow in speed and accuracy.

## 2.2 Project Specification

The project specification has changed over the year; this is mainly due to time and complexity constraints. The main requirement of using machine learning to carry out focused crawling on a specific topic has remained unchanged. However, the crawler is no longer based on the History Enhanced Focus Crawler[14]. A lack of research material around the techniques proposed in this paper made me cautious. As although this paper produced excellent results, the results had not been replicated by any other academics. After further research, a different selection of features was chosen for the web crawler.

As a result, the project specification will no longer include features such as the web segment predictor. Instead, the system uses a classifier that searches for keywords on a web page to determine the relevancy of the web page. The list of keywords is generated during the training of the model, where the model recognises which words are most commonly used in web pages about the given topic. The updated project specification is listed below, reflecting the minor changes made. The project specification is split into functional and non-functional requirements.

### 1. Functional Requirements

- (a) The web crawler must be able to download web pages from the internet, removing unnecessary text and code using the Python Requests[15] and Pyquery[16] packages.
- (b) The web crawler must not crawl the same page more than once.
- (c) The web crawler must start the crawl with the seed URLs.
- (d) The web crawler must be able to follow links on web pages to continue crawling.
- (e) Create a web crawler that can focus the crawl on a given topic using a machine learning classifier.
- (f) The machine learning model must use keywords that have been learnt to recognise the relevancy of a web page and assign it a score. This score will represent how confident the classifier is that the web page is relevant. The classifier will be implemented using the Natural Language Toolkit (NLTK)[17] Python Library.
- (g) There should be a mechanism for altering the crawl depth of a crawl.
- (h) A web application will be created to help visualise the progress of the crawl using visualisations. These visualisations will include tables to display which websites are being crawled. The web application will be created with the Django Web Application Framework.

### 2. Non-Functional Requirements

- (a) The web application will have an easy to use and simple design that does not complicate usage of the web crawler.
- (b) The web application should be able to run on any of the four main browsers; Google Chrome, Mozilla Firefox, Safari and Microsoft Edge.
- (c) The model must output high relevancy scores for web pages of the given topic and low relevancy scores for web pages not about the given topic.

### 3 Design

At a high level, the design of this project is split into three main parts. Firstly, is the design of the web crawler itself, secondly is the classifier and thirdly is the web application. The design for each of these components have evolved over the year and any modifications from the original design are justified in further sections.

#### 3.1 Web Crawler Design

The design of the web crawler can be broken down by its essential functions. Figure 1 shows a UML diagram outlining the web crawler design. The classes have been designed to be as simple as possible with specific functionality assigned to each of the classes. Employing these features adheres to software development best practices. This is because classes are loosely coupled and have simple and easy to understand structures.

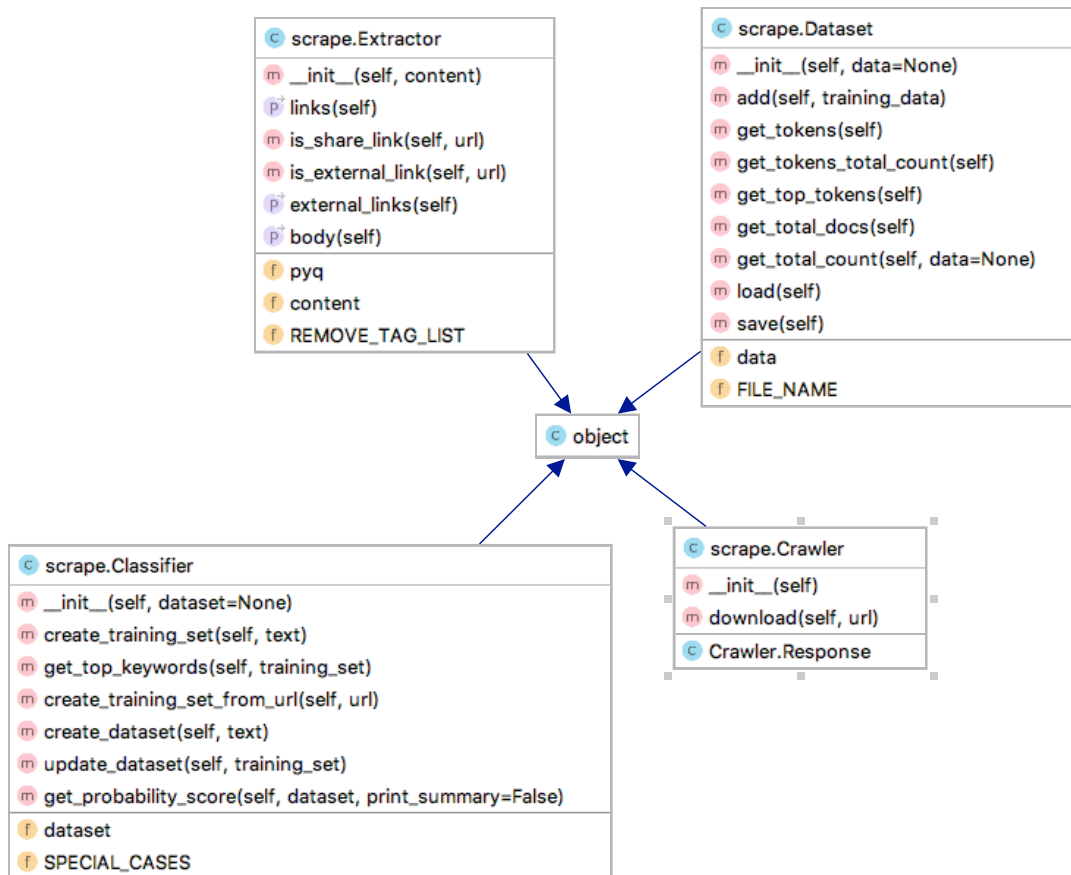


Figure 1: Web Crawler UML Design

The purpose of the Crawler class is to download web pages from the internet. Hence it is home to the download method. This method checks whether a URL has been previously



crawled, if not the web page is downloaded and cached. The caching is carried out to make any further manipulation of the web page faster.

The Response class is an inner class of the Crawler class and is there to help facilitate better error handling when crawling websites. The class has some helper methods that check if the response was successful. If the response wasn't successful, then the *get\_error\_message* method returns the error so it may be logged.

The purpose of the Extractor class is to transform web page data into a usable state. Placing the data into a usable state means disposing of any JavaScript, Cascading Style Sheets (CSS), images and other Hypertext Mark-up Language (HTML) that is no longer needed. Images are also disposed of to save space on storage when caching web pages. The irrelevant code is disposed of because the crawler is only interested in the text that helps to identify the topic of the web page. This class removes any links that might lead to social media pages and keep track of whether links are external or internal. External links start with '*HTTP*' and would take the user to another website. Internal links do not start with '*HTTP*' meaning they take the user to a different part of the same website. It is crucial to make this distinction as it is useful in preventing the crawler from getting stuck on one website.

To manage the data set, the Dataset class is created. This class contains functionality that allows the user to manipulate the data set, such as adding to the data set. Functionality to query the data set is also within this class. An example query is the total number of websites in the data set. The decision had been taken to build a data set as opposed to sourcing a data set online. Building a data set means a custom data structure can be designed to store the data set. Empowering the user to build their data set is the central feature in allowing the crawler to be trained to any topic.

### 3.1.1 Machine Learning Classifier Design

After taking into consideration complexity, timing and the literature around web crawling, the Naive Bayes classifier has been chosen. Naive Bayes is a probabilistic classifier based on Bayes Theorem[18]. The reason this approach is called naive is due to the algorithm assuming the value of a feature is independent of the value of any other feature[19]. For example, a carrot may be considered a vegetable if it is orange, long and about 8cm in length. Each of these features would be considered independently by Naive Bayes classifier. This means the classifier would ignore any correlation between the colour, shape and length of the vegetable. Each feature would contribute independently to the probability of this vegetable being a carrot[20]. Combining the two properties of firstly, being based on Bayesian Theorem and secondly, feature independence, forms the basis for a Naive Bayes classifier. However, Naive Bayes refers to a family of classifiers with these two properties; each algorithm builds on these two properties differently. This section outlines the specific implementation of Naive Bayes that is used for this project.

If we take the topic of cars, particular words have a particular probability of occurring on web pages about cars and web pages that are not about cars. For example, most websites about cars use the word 'engine'. However, it is rarely seen on a website that isn't about

cars. The classifier, however, doesn't know these probabilities straight away. Instead, the classifier must be trained to build up values for these probabilities[20]. To train the classifier, a user manually distinguishes between web pages of cars and web pages that do not pertain to cars. For each web page, the classifier analyses the words and adjusts the probability of each word appearing on a web page about cars or a web page, not about cars. For example, the classifier should have learned high probabilities for the words 'engine' or 'petrol' and low probabilities for other words such as 'carrot'.

Once the classifier has been trained, the word probabilities are used to find the probability that a web page with a particular set of words is about cars or not. Each word on the web page contributes independently to the probability the web page is about cars. Then, the probability the web page is about cars is calculated over all of the words on the web page, and if the probability score exceeds a certain threshold, i.e. 90%, then the web page is categorised as relevant. The formula for calculating whether or not a word is relevant to cars or not is shown in Equation 1[21]. However, when considering several words, a different formula is used which is shown in Equation 2[22].

$$\Pr(C | W) = \frac{\Pr(W | C) \cdot \Pr(C)}{\Pr(W | C) + \Pr(W | N) \cdot \Pr(N)} \quad (1)$$

where:

- $\Pr(C | W)$  is the probability that a web page is about cars given it has the word  $W$  on it;
- $\Pr(C)$  is the overall probability that any given web page is about cars;
- $\Pr(W | C)$  is the probability that a word  $W$  appears in web pages about cars;
- $\Pr(N)$  is the probability that any given web page is not about cars;
- $\Pr(W | N)$  is the probability that a word  $W$  appears on a web page, not about cars.

Most algorithms that follow this formula assume equal probability between a web page is about cars and a web page not being about cars; this is the approach that is followed. As a result,  $\Pr(C)$  and  $\Pr(N)$  cancel each other out.

$$p = \frac{p_1 p_2 \dots p_n}{p_1 p_2 \dots p_n + (1 - p_1)(1 - p_2)(1 - p_n)} \quad (2)$$

where :

- $p$  = probability that a web page is about cars;
- $p_1$  is the probability  $p(S | W_1)$  that it is web page about cars given that the web page contains the first word;
- $p_2$  is the probability  $p(S | W_2)$  that it is web page about cars given that the web page contains the second word;

- $p_n$  is the probability  $p(S | W_n)$  that it is web page about cars given that the web page contains the  $n^{th}$  word;

A combination of both Equation 1 and 2 are used to form the basis on the Naive Bayes Classifier. Equation 2 returns a final score, which if the score surpasses a threshold limit would mean the web page is categorised as relevant to cars.

## 3.2 Web Application Design

The web application will follow a Model View Controller[23] (MVC) design. The model is responsible for handling the data of the application. The model represents the data being transferred between the view and the controller and any logic for communicating with the database. The view contains all logic about the user interface (UI) of the web application. The controller is an intermediary between the model and the view. The controller manipulates the data received from the model and then interacts with the view to render output. The MVC architecture was adopted as the structure helps to facilitate much faster development but also cleaner and easier to maintain code.

### 3.2.1 Model

The model consists of seven classes, translating to seven tables in the database. Figure 2 shows the Entity Relationship (ER) Diagram for these tables.

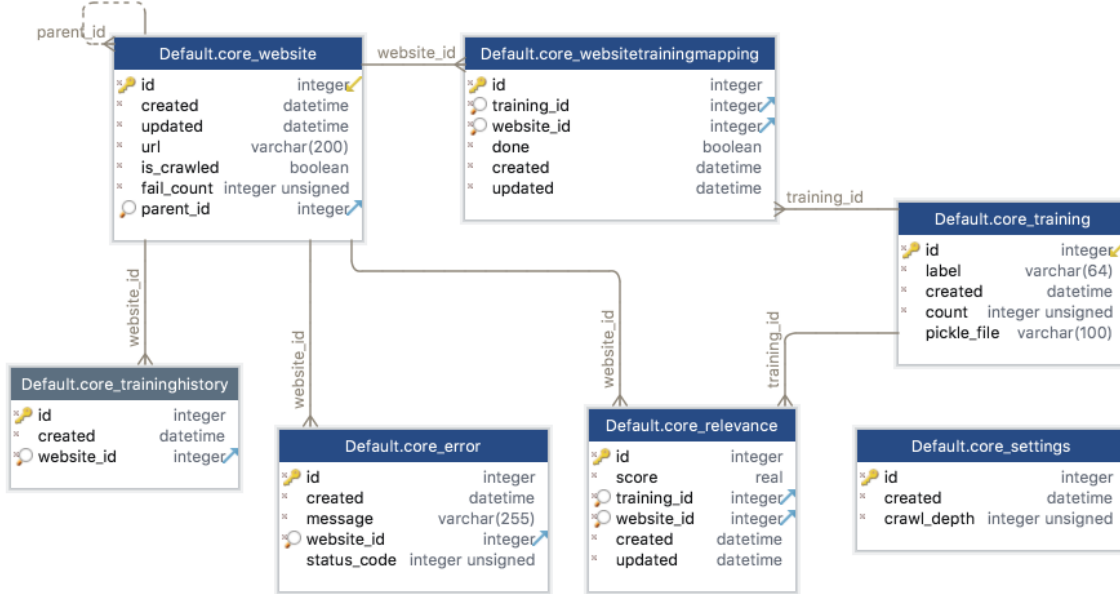


Figure 2: ER Diagram depicting Database Structure for Model.

As the crawler can be trained to any topic, the table *core\_trainings* is used to store the data set for each topic the user wishes to use to train the model. These data sets are given a name, a count for how many websites are in the training set and the name

of the *pickle*[24] file used to store the data set. The *core\_traininghistory* table is used to store a mapping between which websites have been used to train each of the models. The mapping prevents users from training the model with the same URL twice. The *core\_WebsiteTrainingMapping* table is used to store the mapping between websites and the training sets to which they belong. The mappings are created with websites that are stored in the *core\_website* table. Each website states whether it has been crawled or not to prevent crawling the same site again. Also, the fail count registers if the website was unable to be crawled. If it is unable to be crawled, an error is logged in the *core\_errors* table. Once a website has been crawled a relevancy score is generated and stored in the *core\_relevance* table. The *core\_settings* table stores the depth to which the crawl continues to and a relevancy score. While the web crawler is crawling, if the crawler finds web pages with a relevancy score that meets or exceeds the relevancy score set in the *Settings* menu, the web page would be added to the training set. This feature is used to help grow the training set.

### 3.2.2 View

Upon opening the web application, the user is greeted by the main menu. The main menu will have the following options; *Websites*, *WebsiteTrainingMappings*, *Relevancies*, *Errors*, *Settings*, *Training* and *Training History*. Clicking on any of these options takes the user to another page, and except for *Settings*, all these pages contain a table. For example, the *Websites* web page holds a table of all websites that have been found through crawling and whether or not these websites have been crawled or not essentially displaying the properties that are being stored in the *core\_websites* table. There is also functionality for the user to add websites they have found, or delete websites from the table. This set up is applied to the other pages, except for the settings page. The setting page consists of two text boxes. One for the user to enter in the integer value for the crawl depth they wish to set. The other is to set relevancy score for adding newly crawled pages to the training set.

### 3.2.3 Controller

The controller is where the functionality of the web crawler program is extended. There are two primary methods. The first method looks for URLs that have been added to the *core\_websitetrainingmapping* table and then trains/re-trains the model using the websites. This acts as the primary way to train the model. The websites can be added to the model through the views described in section 3.2.2. This method combines several of the methods within the web crawler program such as *download* and *extract* alongside new methods to save to the database. Scraping is the process of extracting data from a web page. The second method is used to scrape any websites in the *core\_websites* table that have not been scraped yet. This method takes functionality from the web crawler project to download the web page, transform the data into a usable state and calculate the score. New methods are added for reading from the *core\_websites* table then updating the *core\_websites* table to reflect the fact that the website has now been crawled. The coupling of the web application and the web crawler program forms a fully functioning web crawler. The web crawler can be trained on any topic and can then crawl the web looking for pages of that topic autonomously. It

is also within this method that web pages exceeding the relevancy score set in the 'settings' menu are added to the data set. The model may then be retrained with the new websites.

### 3.3 Process

Before development, it was vital to define a process that was able to be followed to develop the project in an organised manner. Agile development is a branch of software development. Agile development is iterative, and requirements evolve through the course of the project. Agile development was adopted as it allows for change and splits the project into time boxes. These time boxes define what functionality should be delivered when. It was important that changes could be incorporated as requirements change throughout the project. Requirements mostly change due to time and complexity issues. A Kanban board was used to facilitate agile development and to help organise the project.

The kanban board is split into four 'lanes', these 'lanes' all consist of tasks that need to be completed to finish the project. These lanes are 'To-do', 'Do Today', 'In Progress' and 'Done'. The tasks are then placed into one of these four 'lanes' depending on the status of the task. Appendix A shows the Kanban board being used during the development of the first part of the project, the web crawler section. Further to this, weekly meetings with my supervisor were organised where the current development and plans were discussed. The Kanban board was then updated to reflect these discussions.

## 4 Development

Development describes the process of implementing the design. This section highlights how the design was made a reality alongside discussing the different technologies that have been used for each stage of the project. Similar to the design, development was carried out in three main stages; development of the web crawler, development of the machine learning classifier and development of the web application. The technologies have changed massively since the original development plan, and any deviations are explained in their respective sections. The development of this project has been carried out in Python. Python was chosen for its simplicity, extensive standard library and its active support community. Additionally, there are a vast amount of tools available in Python that could be used within this project.

### 4.1 Web Crawler Development

When researching web crawlers, many had used the web scraping framework Scrapy [25]. Scrapy's primary features are that it provides a quick way of crawling web pages to extract structured data from web pages. Scrapy achieves this through handling HTTP requests to URLs for the user. The user provides a starting URL and what information they wish to extract. As a result, it was a natural choice as it provided the exact functionality needed. Scrapy also has a vast array of other features and it was well documented. However, using a framework means the program's flow of control is placed with the framework. Using the framework gives less power to the developer and reduces their flexibility. Flexibility is

paramount in a project such as this due to the uncertainty in the feasibility of the project. Therefore, research was conducted into what libraries were available that could be used to perform the same task without relinquishing control or flexibility.

First, a virtual environment has been created for the project. A virtual environment is set up as it creates an area specifically for the project that won't be affected by other programs. In particular, packages that are installed are confined to the virtual environment. Installing packages locally is better than installing Python packages globally. Installing packages globally may break system tools or other programs. The virtual environment also makes it possible to track the packages used in the project. Installing packages to the virtual environment may also be useful when running the program on other computers. A list of required packages can be generated for the virtual environment using Pip. For this project, the required packages can be found in the source files. A combination of the Python package manager 'pip'[26] and the 'virtualenv'[27] package was used to create this virtual environment. 'Pip' is used to manage Python packages, providing the functionality to install, update and delete packages. 'Pip' is used throughout the project for these purposes. 'Pip' is then used to install 'virtualenv' which creates an isolated virtual Python installation allowing the user to install packages into that virtual installation. Having gone to the projects directory the command `'python3 -m virtualenv env'` can be used to create a virtual environment. The -m flag represents the module name (virtualenv), and env is used as the name of the environment. With the virtual environment created, it was now possible to begin development on the web crawler program.

Development started with the Crawler class; ensuring there was a robust method for downloading web pages from the internet. To download web pages, a way of communicating with those web pages was needed. Downloading web pages is where the first and the most crucial library was needed, the Python Requests library. The Requests library is the standard library used for making HTTP requests within Python. The library abstracts the complexities of making requests by providing a simple to use API. The API allows one to focus on making requests rather than how they are made. By making use of this package, the download method could fetch a website and save it.

To fetch a website the following command was used: `'response = requests.get(URL, headers=HEADERS)'`. The 'get' method is called on the Request object. The request is an HTTP GET request which is a method used when requesting data from a specified resource. The method takes two arguments, one is the URL of the web page, and the other is headers. Headers are any additional information that may need to be passed with the request.

The user agent is the software that is acting on behalf of a user; in this case, it is the web browser. For this project, headers include the user agent only. The result is saved in the response variable. To save the web pages once they have been fetched, a unique name would need to be given to the text file storing the response. The SHA256 hashing library was used to hash the URLs. Hashing is a one-way mathematical function that generates a unique value from a string. SHA256 was chosen as the hashed values that are returned are 256 bits long. Hashing the URLs means the crawler can save millions of web pages all with unique names. However, error handling has been put in place to make sure the correct response is received. The response is verified by checking the status code of the result that has been

returned; a status code of '200' means the request has executed successfully. Additionally, to prevent saving the same website twice there is also a check in place to check whether the hashed URL has previously been saved.

Within the Crawler class, there is an inner class called Response. Response acts a wrapper class to any response received from a web page, such as a response to a GET or PUT request. After researching the meaning of different HTTP codes, two methods were developed that checked if the request was successful or not. By checking if the returned status code was 200, it would confirm the request was successful, and the web page had been downloaded. There is also a method that checks to see if any error messages were sent back with the request if it was not successful. The method is used for logging any responses. This class lives within the Crawler class as it in this class the only HTTP requests are sent, through use of the *download* method.

If a successful response is received, the results need to be processed into a usable format. Hence the Extractor class was next to be developed. This class contains one method also, the extract method. The goal of this method is to remove any JavaScript, CSS and HTML tags that contain random text. An example of the HTML that is removed is the 'nav' tag which contains the navigation bar at the top of most websites. The navigation bar usually would not contain any useful words. PyQuery is a 'JQuery like' library for Python. Loading the result into a PyQuery object meant that you could run JQuery queries on the object. One such query is removing the text via their HTML tags. For example, JavaScript is usually in the '<script>' tag so PyQuery could remove any text within those tags. The PyQuery package was used to filter down the result. The result that is returned is any text that is left within the HTML body of the web page. This is because the body is likely to provide us with the keywords needed to either help train the model or help classify the web page.

## 4.2 Machine Learning Classifier Development

Before developing the Naive Bayes classifier, it needed to be decided which library would be used to create the classifier. In the original specification, it was planned that the open source library TensorFlow[28] would be used to handle any machine learning aspects of the project. The original specification used TensorFlow because of the active support community and the number of different classifiers TensorFlow could be used for. However, after analysing the advanced capabilities and physical size of the library, decided that the benefit of using Tensor-Flow did not outweigh the costs. The project would be more significant in size and slower due to the weight of this library. Instead, the NLTK Python library was selected for use. This package focuses solely on providing tools that help manipulate human language; these features include tokenising and tagging text. The development for the Classifier is split into two parts; data set generation and the Naive Bayes classifier.

### 4.2.1 Data set Development

The first step in creating this classifier was to generate a data set to train the classifier. The Dataset class was created for this very purpose. A data structure is needed to store the data set. Figure 3 shows the custom data structure created to store the data set. The

```

'''
{
  'total_docs': 11,
  'total_count': 1234
  'tokens': {
    'token': {
      'count': 1,
      'docs': 2
    }
  }
}
'''

```

Figure 3: Dataset Data Structure

data set is organised in a way that makes performing calculations for the classifier simpler. The data structure keeps track of the total number of web pages that have been saved to the data set in *total\_docs* and *totalcount* stores the total number of keywords in the data set. The *tokens* field stores which word the token is, with the *count* storing the number of times the word has appeared through all of the web pages. The *docs* field stores the number of web pages the word has appeared in. The data structure was organised in this fashion as these are statistics that are needed when calculating the relevancy score of a web page; this is demonstrated in section 4.2.2.

The other methods in this class provide functionality that adds to the data set as well as helper methods that return specific values from the data structure. The data set is stored as a 'pickle'[24] file. The 'pickle' library is used to serialise a python object structure and converts the object into a character stream. This character stream contains all the information necessary to reconstruct the object in another python script. Using the 'pickle' file makes reading, writing and saving of the data set much simpler. As a result, the class possesses functionality that loads and saves 'pickle' files.

A data set needs to be created using the interface provided by the web application. First, a training set needs to be created via the 'Training' menu. Then websites need to be inputted manually via the 'Website Training Mapping' menu which links websites to training sets although this may be tedious at first. A feature that has been implemented that checks if a website that has been crawled meets or exceeds a user-specified relevancy score. This score is set via the 'Settings' menu. The addition of this powerful feature means that the data set can be grown without the need for copious amounts of data to start with. This also allows the crawler to crawl for any user-specified topic.

#### 4.2.2 Naive Bayes Classifier Development

To utilise the data structure that has been created, the crawled data must be placed into this format so that it may be added to the data set. Placing data in this format is one of the main functions of the Classifier Class. The classifier needs a list of keywords and how often they are used to judge the relevancy of web page; creating this list is the primary goal of the Classifier class.



The *create\_training\_set* method utilises the NLTK package to help format data into the correct structure. A web page is taken as input. NLTK is then used to tokenise the words on the web page and change the words to lowercase. Tokenising words is the process of splitting text into individual words. For example, tokenising the sentence *'Theboyjumped.'*, would result in ['The', 'boy', 'jumped', '.'] being returned. As the classifier is only interested in keywords that help to identify the topic of the web page, any text which isn't useful for this purpose needs to be removed. Dictionaries are created for different words or grammar that needs to be removed. Firstly, a dictionary containing punctuation is generated using the 'string' library. The web page text is then iterated over in an attempt to find matches in the dictionary; any matches are then removed. Next, any integers are removed using a dictionary of integers which are generated using the 're'[29] package. The 're' package is used to generate regular expressions.

Following this, any 'stop words' are removed. 'Stop words' is a list of words provided by NLTK and contains words such as 'is' and 'the'. URLs are also removed similarly to integers, through using regular expressions. Words which are less than three characters are removed to catch any words that were not removed by the 'stop words' dictionary. Words greater than thirty characters are also removed to catch words which have not been appropriately tokenised. Both the *SPECIAL\_CASES* and *BLACK\_LIST* dictionaries are used to remove any words or punctuation that was not removed in the previous steps. Lastly, the NLTK *lemmatizer* functionality was used to turn any plural words into single words. The process of generating a clean list of keywords was iterative. Deciding what needed to be removed was worked out through executing the program and analysing the list of words that were returned.

Through this, it was worked out what words or grammar in the list needed to be removed. The output is a list of keywords with a value for how many times the word appeared on the web page. The result is called a 'training set'. A training set is different from the data set as the 'training set' is the output from one web page whereas the data set stores all collected keywords data so far. Figure 4 shows two example frequency tables; the first table shows the raw data and the second table shows the processed data. It can be seen there is much irrelevant and usable data in the first table compared to the second. Hence, creating a need for this processing. New methods in the Crawler class include creating a new data set and then adding the newly generated training set to this newly created data set. There is also functionality to update the existing data set. These methods both utilise the *add* method in the Dataset class.



Token	Frequency
...	8
facebook	12
Chicken	28
<a href="#">www.twitter.com</a>	1
<a href="#">privacycontactaboutus</a>	1

Token	Frequency
food	34
recipe	31
chicken	28
vegan	23
salad	22

Figure 4: Data Processing for Classifier

The method *get\_probability\_score* contains the logic which calculates the relevancy score of a web page. The tokens from the current web page and the tokens from the data set are assigned to variables using helper methods from the Dataset class. The code then iterates through the list of the current web pages’ tokens and checks if any of these tokens are in the list of top keywords from the data set. Although the data set may contain hundreds of keywords, only the most common twenty words are used. The most common words are used as including less common words meant that the scores given to relevant sites were much lower than expected. This is because even if the current web page possesses the most common words if it does not possess the lower frequency words, this can drastically reduce the score given to the web page. Twenty was finalised on after experimentation with this value until the scores being produced accurately reflected the relevancy of the web page, for example, if the given topic was food recipes, web pages about recipes should be receiving a score of 80% or higher.

If a match is found between the top keywords list and the tokens from the current website, then Equation 1 is used. Equation 1 to calculate the probability that the web page is about recipes given that the website contains the matched word. For this example, let’s take *food* to be the matched word. The probability that the word *food* appears in web pages about recipes is worked out by dividing the frequency of the word *food* in the data set by the total number of tokens in the data set. The probability that the word *food* appears in web pages that are not about recipes is simply one minus the value for the probability that the word *food* appears in web pages about recipes. For this project, it is assumed that the probability of a web page being about recipes and a web page not being about recipes is equal. Therefore both these values are fixed at 0.5 for each calculation. Assuming that the word *food* appears 92 times and the total number of words in the data set is 687, Equation 3 shows the result.

$$\frac{0.134 \cdot 0.5}{(0.134 \cdot 0.5) + (0.866 \cdot 0.5)} = 0.123 \quad (3)$$

By dividing 687 by 92, the probability that the word appears in web pages about recipes is worked out at 0.134. To work out the probability that the word appears in web pages, not about recipes, it is 1 - 0.134, 0.866. Plugging these calculated values into the formula gives you a probability of 0.123. These steps need to be repeated for each of the top twenty words in the keyword list. These probabilities are combined by summing the individual words probabilities. If this score reaches above 0.8, then the website is deemed relevant. As a note to the user, the word appearing 92 times out of a small data set is relatively unrealistic, and the probability values returned are less than 0.123, this has been used for purely demonstrative purposes.

### 4.3 Web Application Development

A web application framework was opted to develop the web application. Having taken into consideration that the code base was currently in Python and considering the design, Django[30] was selected for use. Django is a Python-based, open-source web framework. Django follows a Model-View-Template (MVT) architecture. MVT is the same architecture

as MVC, but with a different naming convention. Again, the model handles any interactions with the database. The view interacts with the model to carry data and executes business logic. The template handles the UI. However, there is no separate controller. The framework itself handles any actions performed by the controller.

When creating a new Django project, the Django framework creates a Django Admin. The Django admin is a built-in feature which generates a user interface to perform create, read, update and delete (CRUD) operations on the models. The Django Admin is a powerful tool that fits exactly with what was required from the web application. Therefore, a web application was created that utilised this tool. Opting for this method, meant that the web application wouldn't utilise the standard MVT architecture. Instead, only the model part of MVT would be used. The architecture is further explained in section 4.3.1.

### 4.3.1 Django Admin and Models

The Django Admin reads metadata from the model to provide a fast and straightforward interface where the user can manage the content of their application. This feature is automatically enabled and used to perform administrative tasks. Django ships with its lightweight development server which is useful for development as it means an individual does not have to set up a production server before they can develop a web application. Running the server automatically starts hosting the web application. The following address would need to be typed into a web browser: `http://127.0.0.1:8000/admin` to access the Django Admin. The address is the Internet Protocol (IP) address of the server followed by a request for the admin page. Appendix B shows the Django Admin login page. As the Django Admin is meant for administrative use, one must first create a superuser account to gain access. A superuser account was created via the command line. Having gained access to the Django admin, next, the models would need to be set up so that the Admin could display them.

The models are stored in a file called *models.py*. There have been seven different models created, consistent with the design presented in section 3. These each correspond to a table in the database. Appendix C shows an example model of the *core\_websitetrainingmapping* table. Each field within a model represents a column in the database. Therefore, *core\_websitetrainingmapping* has five fields being 'created', 'updated', 'website', 'training', 'done'. The 'created' and 'updated' fields are both DateTime variables meaning they store dates. The parameters given to these variables mean they are non-editable and will therefore not be displayed in the Django Admin. The 'website' and 'training' fields are both foreign keys from the *core\_website* and *core\_training* tables. The first parameter references the table from which it is a foreign key and the second parameter specifies that if the entry is deleted, all entries that have reference to this entry. The 'done' field is a Boolean that is set to false. This process is repeated for the various tables. Where necessary, the `__str__` method has been redefined to return a specific value for those pieces of data that are stored as objects.

Once the models had been created, they needed to be registered with the Django admin. This is achieved through this line of code: `admin.site.register(models.WebsiteTrainingMapping, WebsiteTrainingMappingAdmin)`. This code is placed in the *admin.py* file. This file is used by the Django Admin to inform

the Django Admin of which information to load. Also within this file are classes that specify how the Django Admin represents the model. These are called ModelAdmins. Appendix D shows the ModelAdmin for *WebsiteTrainingMapping*. The 'list\_display' option specifies the model should be represented as a table with the following headings. The same setup has been used for all the models when creating their ModelAdmins.

Once the models have been set up, the database tables needed to be created. Creating the database tables is achieved with Django's 'migrate' command. The 'migrate' command translates the model into a database schema for the database of your choice. The 'migrate' command is also used for propagating changes made to the model and updating the database schema to reflect this. PostgreSQL[31] was selected as the database. PostgreSQL was chosen as it can handle heavy workloads and there is an active support community. Handling heavy workloads is crucial as crawling can happen at faster rates depending on the hardware used and also provides room for further scalability. The next step was to implement the web crawler logic from the web crawler program, having set up the database tables.

### 4.3.2 Commands

Commands are used to control the Django Admin and tell the Admin to perform some action. For example, the command '*djangoadmin flush*' tells the Django Admin to remove all data from the database. These commands are all built into the framework. However, a method was also needed in getting the Django Admin to perform actions specified by a user. Django allows the user to do so through its custom command feature. By creating a new Python file in the *commands* sub-directory, Django automatically registers the name of the file as what is used in the command line to perform the action. Two commands have been created to reflect the two methods that were outlined for the controller in Section 3 Design.

The first command is in a file called *train.py*. The purpose of this command is to train the model. First, a user must add a training set via the *Trainings* page. To add websites to the training set, the user must then go to the *Website Training Mapping* page. The model can now be trained with the websites that have been added to the data set. The model can be trained using the command *django – admintrain.py* in the command line.

The command tells the Django Admin to execute the *train.py* file. The train command then iterates through each website in the *core\_WebsiteTrainingMapping* table. Functionality from the crawler program is used to download the website. If the download is not successful, an error is logged, and the crawler moves on to the next web page. If the download is successful, the downloaded web page is then organised into the custom data structure outlined in Section 3 Design. A training set file is created, if one does not already exist. The data is then added to the training set along with its relevancy score, and the file is then saved. Once the page has been crawled, all the internal and external URLs are added to the *core\_websites* table. These URLs form the basis for what websites are crawled next. The more websites that are added, the more accurate the list of keywords generated by the training are, producing better results. The crawler depth is also checked here. The crawler stops training and scraping after a user-defined crawler depth is reached. The crawler depth

is tracked by storing the parent of each website and the depth that the website was found at.

The second command is in a file called *scrape.py*. The purpose of this command is to scrape any websites that are in the *core\_websites* table and haven't been crawled yet. Scraping is achieved through first iterating over all the websites in the *websites* table which has not been crawled. The websites are downloaded. Again, if not successful, the error is logged, and the crawler moves on. The classifier is then loaded up by selecting the file and loading it into an instance of the Classifier class. The score of the web page is then calculated using the classifier that has been loaded. The relevancy is then saved and the *core\_website* table updated. If the relevancy score exceeds the score set by the user in the settings menu, the web page is then added to the *core\_WebsiteTrainingMapping* table. The addition of web pages to the data set is repeated for all websites that exceed or meet this score. By providing a few URLs to start with, this feature makes it easy to grow the data set at an exponential rate without much data to start with. The combination of these two commands forms the basis for allowing the crawler to run autonomously by repeating this cycle of training the model, finding links based on the training data and then crawling them.

## 5 Testing

The testing for this project is split into two main parts:

- **Classifier Performance:** test the performance of the classifier trained using keywords generated from a test dataset.
- **Web Crawler Performance:** reviewing the results from a large crawl.

### 5.1 Classifier Performance

When testing a classifier, traditionally the dataset used to split into training and testing data. Training data is used to train the model. In this case, to generate a list of keywords and their frequencies to recognise websites that use the same keywords. Test data is then used to measure the accuracy of the training by providing the model with unseen websites. The model gives each website a score from 0 to 1 where 0 is not relevant, and 1 is relevant.

The model is trained using 100 websites on the topic of food recipes. The model is then given 100 unseen websites (test data) and asked to classify the web pages. 50 websites in the test data are about food recipes, and the remaining 50 are on any number of other topics. To measure how accurate the classifier is, any websites that receive a score higher than 0.5 are categorised as relevant. A score of 0.5 indicates the web page is more relevant to food recipes than not and vice versa.

Using this data, we can calculate three metrics used to measure the performance of a text-based classifier. Precision is the first metric and is the fraction of relevant web pages found among all web pages that were found[32]. Precision indicates how well the classifier is performing at rejecting irrelevant web pages. Recall is the second metric and measures

the fraction of relevant web pages found over the total amount of relevant pages[33]. Recall indicates how well the classifier is at recognising relevant web pages. If we assume that  $R$  is the set of relevant web pages in test dataset;  $C$  is the set of relevant web pages assigned by the classifier. Therefore, precision and recall can be defined as outlined in Equation 4 and 5[34].

$$Precision = \frac{|R \cap C|}{|C|} \times 100 \quad (4)$$

$$Recall = \frac{|R \cap C|}{|R|} \times 100 \quad (5)$$

Although precision and recall are industry standard methods for assessing classifier performance, they are not without fault. One flaw with precision and recall is that trying to increase the value of one of the measures decreases the value of the other. Hence the F-measure was developed to counteract this problem. The F-measure is defined in Equation 6.

$$F\text{-measure} = \frac{(\beta^2 + 1)Precision \times Recall}{(\beta + 1)Precision + Recall} \quad (6)$$

$\beta$  is the value for the relative importance for precision and recall[35]. If  $\beta = 1$ , then precision and recall have equal importance. If  $\beta > 1$  then recall is given higher importance. If  $0 < \beta < 1$  then precision is given higher importance. A  $\beta$  value of 0.5 is used here to reflect that recall is more important in this project than precision.

## 5.2 Web Crawler Performance

A more massive crawl to a crawl depth of 3 was carried out to gain measurable statistics about the web crawler performance. The crawl size is more significant than any previously carried out during development but was still limited by the hardware and timing that was available. The number of links grew exponentially at each level of the crawl making it unfeasible to continue crawling without great expense or several days of crawling. The crawl starts with one seed URL that form the basis of the crawl. The statistics include but are not limited to the total number of requests made, the number of successful requests and the number of unsuccessful requests. The unsuccessful requests are filtered down further by the error that was returned. These statistics indicate how well the web crawler crawls pages by looking at the number of errors received. It also shows problem areas that may need to be improved upon in future work.

Crawl depth was chosen over a specified time which is typically used. Crawl depth was used because a high percentage of web crawler performance depends on the hardware. Therefore by using a metric not affected by the hardware, this gives a more accurate indication of the performance. However, the harvest ratio[36], which is defined in Equation 7

where  $t$  is the number of pages crawled, can be contrasted with other web crawlers. Harvest ratio is very similar to precision.

$$\text{Harvest Ratio} = \frac{\text{Number of Relevant Pages}}{\text{Pages Crawled}(t)} \quad (7)$$

The existing Web Crawler project functionality alone is used to test the classifier. The giant crawl and harvest ratio are generated using the web application. The results for both types of testing are given in Section 7 Evaluation of the Final Product.

### 5.3 Other Testing

Two other tests are undergone to satisfy all of the requirements. Firstly, a test to ensure the web application is fully functioning on the four main web browsers; Google Chrome, Mozilla Firefox, Safari and Microsoft Edge. The web application is opened in each of the web browsers and set list of tasks are completed on each browser. Appendix H shows a full list of operations that were completed on each browser. These operations are kept the same for each web browser to ensure fairness. If all these operations are completed successfully on each browser, then the test is successful.

Secondly, user acceptance testing is carried out to gain feedback on the web applications user interface. User acceptance testing is carried out with a small cohort of six Computer Science students as the application is intended for use by members of the Computer Science discipline. Appendix I shows the questions the users were asked.

## 6 Description of the Final Product

This section seeks to bring together the three components of this project, to help the reader understand the project as a whole.

The final product is a focused web crawler based on the Naive Bayes classifier, presented as a web application. The web application takes advantage of the Django Web Application Framework and in particular the Django Admin feature. Users first need to create an account to access the application. An account is obtained with the 'python manage.py createsuperuser' command. A user is then prompted for a username, email and password. Appendix E shows the page users are greeted with upon logging in. Each of the headings listed under 'CORE' represents a table in the database.

Clicking on any of the headings takes the user to a page where they can perform CRUD operations on the database tables through the user interface. The data is presented in the form of a table, displaying useful information about the website such as whether or not it had been crawled and the link from which that link was found. Appendix F shows the 'Websites' page, which contains the list of websites that have been crawled or have been scraped from websites. By clicking 'Add Website', the user can add a website to the table. If the user wishes to update a record, then they can do so by clicking on said record. Appendix F shows the page the user is taken to. The user may then update the record and save it by pressing the 'Save' button. There are three different types of saving to make the

user experience faster. 'Save and add another' instantly takes the user back to that page so they may add another record instead of taking them back to the main menu.

There is a specific process that needs to be followed to run a crawl and to ensure the crawl runs correctly.

1. Create a new training set through the 'Training' menu.
2. Add websites to that training set through the 'Website Training Mapping' menu.
3. In the command line, execute the 'train' command './manage.py train' to train the model with the websites just provided.
4. An optional step would be to set values for the crawl depth and relevancy score threshold to add newly crawled websites to the training set. Setting the crawl depth and relevancy score can be completed in the 'Settings' menu.
5. Execute the 'scrape' command './manage.py scrape' in the command line to scrape all the websites that were found from the websites used to train the model. (The websites used to train the model act as the seed URLs).
6. Repeat the commands of 'train' and 'scrape' commands until the crawl depth is reached or the user is satisfied with the results.

The procedure described is a broad overview. However, there is much flexibility with setting up a crawl. A user may be able to load in an existing training set in the form of a 'pickle' file if they wish to do so. Besides, the websites added to the training set can come from the user or the list of crawled websites. The final headings of 'Relevances' and 'Training History' are both used to ensure the user has access to all the information they may need through the web application. Relevance stores the websites and their associated relevancy score and presents this in a table. Training History keeps track of which websites have been used in which training set and the score they were given.

## 7 Evaluation of the Final Product

This section seeks to evaluate the product as a whole. The evaluation is carried out in two ways. Firstly, by addressing if the project has fulfilled all functional and non-functional requirements. Secondly, by reviewing the test results from the tests outlined in Section 5 Testing.

### 7.1 Aims and Requirements

The primary aim of the project was to try to combine machine learning with web crawling to create a focused crawler that can carry out crawls on any topic. Secondary aims include helping to shed light on an area of research dominated by multinational companies and to provide a web application to display the web crawler. These aims are broken down into the functional and non-functional requirements. All the functional and non-functional requirements were met and extended throughout this project.



The functional requirements were achieved as follows. The web crawler can download web pages and remove any unnecessary text and code. Downloading web pages is achieved through the 'download' and 'extract' methods. The 'crawled' field in the *core\_websites* table shows whether or not a website has been crawled. Before crawling a website, the 'crawled' field is checked to make sure the website has not been crawled before. The crawler bases the crawl on the URLs provided by the user.

The 'get\_all\_links' method extracts all the links on a web page to allow the crawler to continue crawling from the seed URLs. The crawling has been focused through the use of a machine learning classifier. The classifier bases its relevancy score of keywords that are found on websites of the selected topic. The crawl depth can be altered through the settings menu providing termination criteria for each crawl. The crawler functionality is all housed within a web application which uses tables to help visualise the work being done by the web crawler. The web application was created using the Django Framework.

The none functional requirements were achieved as follows. The application was kept simple and adhered fully to Django best practices. Adhering to Django best practices should mean access to the web application through different browsers should lead to no differences in performance or functionality. The application was also given to a small selection of Computer Science students to see how the students would react to the user interface. User acceptance testing allowed me to gain feedback on the ease of use of the application. Therefore changes could be made based on this feedback to help achieve an easy to use design. The results for the cross-browser compatibility test and user acceptance testing was mostly positive. Some users did suggest some possible improvements that could be made. These are discussed in section 7.2 Results alongside a full breakdown of the results from these two tests.

## 7.2 Results

This section will outline the results gathered from the testing plan created in Section 5 Testing.

### 7.2.1 Classifier Performance

Classifier performance is tested through training a model using 100 URLs on the topic of food recipes. The model was then tested against a set of 100 URLs, half of which were about recipes and the other half which were randomly selected. Appendix J shows the training set and the relevancy scores generated. Appendix K and L show the test set and the relevancy scores generated.

The results from the classification of the arbitrary URLs are promising. The classifier correctly identified 43 out of the 50 random URLs as being irrelevant, giving them scores of under 0.5. This is an accuracy of 86%. Two of the websites were incorrectly identified as relevant with scores of 0.52 and 0.56. Although relevant for this test, these two may not have been considered relevant if the threshold was higher. This threshold should be higher for models with more training data. The remaining five websites resulted in errors, meaning the website was not able to be successfully downloaded. The errors could be for several reasons, and these reasons shall be explored further below.

The results from the classification of the recipe related URLs are also very positive. The results show that 45 out of 50 of the URLs were correctly identified as relevant. This is an accuracy of 90%. Three websites were incorrectly identified as irrelevant. However, their scores were still close to the 0.5 thresholds with the highest score of the three being 0.49 and the lowest 0.44. The majority of the websites score relevancy scores of 0.8 and above which shows the crawler performs well at recognising the recipe websites. These results also achieve the non-functional requirement of the model producing low scores for irrelevant pages and high scores for relevant pages.

It is possible to work out the precision, recall and F-measure for the classifier using the results that have been amassed. Precision is calculated using Equation 4 and the result is shown in Equation 8.

$$Precision = \frac{44}{100} \times 100 = 44 \quad (8)$$

The recall may be worked out using Equation 5 and is calculated in Equation 9.

$$Recall = \frac{44}{50} \times 100 = 88 \quad (9)$$

Using these two values the F-measure may be calculated using Equation 6 and is calculated in Equation 10.

$$F\text{-measure} = \frac{(0.5^2 + 1) \times 0.44 \times 0.88}{(0.5 + 1) \times 0.44 \times 0.88} = 0.83 \quad (10)$$

Precision looks at what proportion of positive identifications was correct. Although a score of 44 out of 100 is quite low, it is, in fact, a high score. It is a high score because the highest precision could only have been 0.5 due to the number of relevant web pages in the dataset. Therefore this is a commendable score and proves the classifier can differentiate relevant from irrelevant websites. Recall looks at what proportion of actual positives was identified correctly. Recall was calculated to be 88 out of 100, meaning this is an excellent recall score. A high recall score means the crawler is unlikely to classify a page that is relevant incorrectly. F-measure is an overall measure of accuracy that combines precision and recall. The crawler achieved a score of 0.83 out of 1. A high F-measure score was achieved because more importance was given to recall over precision. It was intended that the crawler be more focused on correctly identifying pages rather than only collecting relevant pages. This score can now be used to compare performance with other text-based classification systems.

### 7.2.2 Web Crawler Performance

To gauge the performance of the web crawler, a more massive crawl than usual was undertaken. This crawl used one starting URL and crawled to a crawl depth of 3. The crawl only used one URL and crawled to small depth as the number of URLs at each depth grew exponentially. For example, five seed URLs at a crawl depth of three resulted in 15 million

web pages needing to be crawled. As this project did not have access to the physical hardware to perform this, a smaller crawl was carried out. The results for each are shown in Table 1 and Table 2.

Basic Crawl Stats	Total	Percentage
HTTP Requests	3455439	100.00%
Successful Requests	3222458	93.26%
Unsuccessful Requests	232981	6.74%

Table 1: Basic Crawl Statistics

From Table 1 it can be seen that the crawl resulted in a total of 3.45 million requests. This crawl took place over 18 hours. Of these requests just over 93% of these requests were successful. This meant that around 230 thousand requests were unsuccessful. Achieving a success rate of over 90% means the crawler performs well at not overloading websites or being blocked by websites. However, even though the percentage was low, there still are hundreds of thousands of websites that failed. Table 2 shows the breakdown of these errors.

HTTP Errors	Total	Percentage
Total Requests	3455439	100.00%
200 (OK)	3222458	93.26%
403 (Forbidden)	108216	3.13%
404 (Not Found)	37909	1.10%
301 (Moved Permanently)	23468	0.68%
302 (Moved Temporarily)	14456	0.42%
401 (Unauthorised)	12778	0.37%
500 (Internal Server Error)	3964	0.11%
Other	32190	0.93%

Table 2: HTTP Error Breakdown

The most common HTTP error received was a 403 error meaning access to the resource was forbidden. A 403 error is usually caused because the web crawler does not have the correct permissions to access the resource so is being blocked. The next principal error codes that were received all mean either the link has been moved, died or is broken. A small number of errors were related to errors on the server with around 0.11% of requests resulting in a 501 Internal Server Error. These errors can't be helped other than trying to request the resource at a later date. A potential fix for the other error codes is outlined in Section 8.1 Improvements.

Depth	Found	Relevant	Ratio
1	112	101	0.901785714
2	12544	11023	0.878746811
3	3442783	2634504	0.76522511

Table 3: Results from Crawl

The last metric to be observed for the web crawler performance is the harvest ratio.

Table 3 shows the number of relevant pages found at each level of the crawl. The ratio between the total number of pages and the number of relevant pages found is then calculated. Relevant pages in this crawl were defined as having a relevancy score of 0.6 or above. Figure 5 shows the graphical representation of these results.

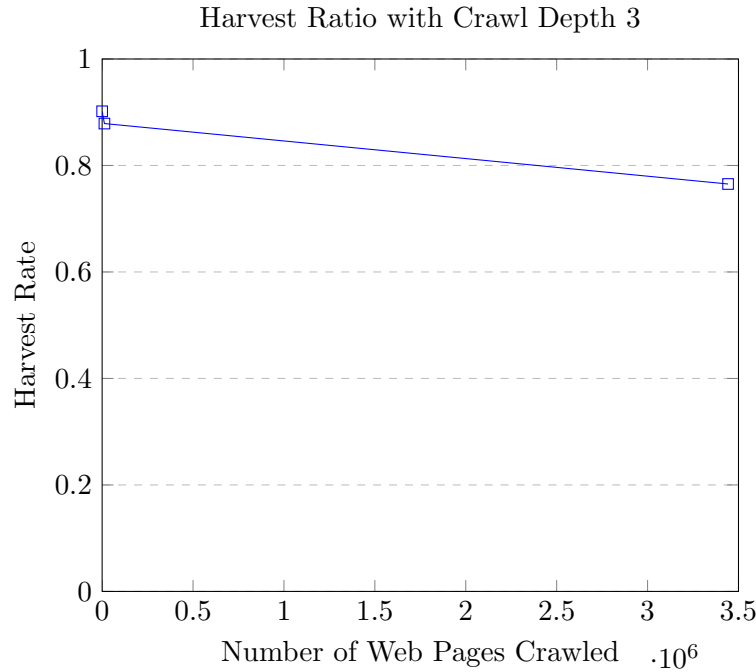


Figure 5: Graph showing the Harvest Rate over a Large Crawl

From Figure 5, it can be seen as that the harvest ratio starts strong with a score of over 0.9. A harvest ratio of 0.9 suggests over 90% of the websites being crawled were relevant to the topic. However, as the number of pages crawled grew, the harvest ratio started to drop. At a depth of 3, around 25% of the pages that have been crawled were not relevant. A drop in harvest ratio is expected as the crawler could quickly exhaust the web pages around a particular topic.

### 7.2.3 Other Testing

In terms of cross-browser compatibility, the list of operations outlined in Appendix H. All operations was carried out in the same order on the different browsers. No problems arose when performing the tasks and all operations were able to be carried out successfully meaning that the non-functional requirement of the application running on all four web browsers was achieved. The lack of problems was due to the simplicity of the application and the power of Django that had meant the application could function well across the different browsers.

The last form of testing carried out was user acceptance testing. The user acceptance testing was carried out with six Computer Science students. Some of the key takeaways from the testing are listed below.

1. 83% of the users said that the information was clearly laid out.
2. 67% of users said that they had all the information required.
3. 83% said analysis of the crawl results was simple and easy.
4. 0% of users said it was easy to set up a crawl.

The results all seem promising with nearly all the users agreeing that the interface was simple to use and understand. However, one glaring problem is that none of the users found it easy to set up a crawl. The users struggled because there were no instructions provided for doing so. As a result, a README file has been added to the project which explains the process for execution. The user acceptance testing was repeated once a README file had been added, and 83% of users found it simple to run a web crawl.

Some other notable feedback included users expressing a distaste for using the command line with the web application. Users found this part inconvenient as the simple web application became slightly more complicated using the command line. The users all expressed that once the crawl depth has been specified the crawler should run autonomously until it is reached. Currently, the crawler stops after each depth, and the crawler needs to be told to carry on.

### 7.3 Comparison with Other Web Crawlers

Using the metrics calculated above it has been possible to compare the performance of the web crawler with other web crawler systems made by academics. One such system was proposed by Wang et al. was a similar web crawler that was based on the Naive Bayes Classifier[37]. The classifier used is split into three components; page analysis, character extraction and relevance analysis. The results from the web crawler show a harvest rate of between 0.4 and 0.5 over a crawl of 10,000 web pages. This crawler seemed to be experiencing problems with URLs pointing to the same page which could explain the much lower harvest rate. As the proposed web crawler at around 10,000 web pages was achieving a harvest rate just under 0.9 according to Figure 2. This project did not suffer the same problem as the project differentiated between internal and external links.

A second crawler proposed by Lu et al. combines a Naive Bayes classifier with a link priority evaluation (LPE) algorithm[38]. The LPE was designed by Lu et al. and partitions web pages, calculating the relevancy for each partition separately. Partitioning worked well as the web crawler achieved a harvest rate between 1 and 0.8 over 10,000 web pages. These results are equal to the harvest rate displayed in this project. The results suggest that partitioning the web page into different sections may work well in helping to classify web pages better and could be something to adopt in the future. However, the results generated from the second crawler were averages taken from a range of topics the web crawler has crawled whereas this project just considered one crawl.

## 8 Critical Assessment of the Project as a Whole

Each stage of the project, design, implementation and testing, was carried out successfully. This culminated into a final product that fulfils the aims of the project. However, considering each section of the project individually, some compromises were made that would be changed if the project were to be undertaken again.

The kanban board used to plan and organise the project, helped to facilitate the right level of project management. Using the kanban board allowed me to track the progress of the project, allowing me to estimate time frames for when parts of the project would be completed. A grasp on time frames made it easier to plan this project and create a project that was suitable for the time frame allocated to develop the project. One way this could be improved, however, would be to break down the tasks even further. An example task written on the kanban board would be 'Complete Extractor Class'. Breaking this down further into individual methods may have put one in better stead with being able to organise their time more effectively. This higher level of detail would allow one to gauge the work required for each class better.

The design and development were split into three main components, the web crawler system, the classifier and the web application. Splitting the components worked well as it meant that focus and research could be dedicated solely to the area currently being worked on. The increased focus made designing and developing each component much simpler. However, as a result of this splitting, some thought had to be given as to how to join the components together when they were completed. Joining the components together was planned for by giving each of the components distinguished functionality and specifying how the different functionality of the components would work together. The plan worked well as a plan until a different path was taken when creating the web application.

Using the Django admin feature meant that the design would need to be changed to incorporate the different architecture. This change meant that compromises would be made with the way the web crawler would be used by users. The UI provided by the Django Admin feature was only provided with the functionality to perform CRUD operations. The Django Admin feature is difficult to modify; therefore it was not possible to add to the UI provided. In this case, the solution opted for was to make custom commands that performed the training and scraping of websites. However, this meant the incorporation of the command line to control the web application. As realised through the user acceptance testing, this was not a well-received change to the original design.

The feedback made me realise that user acceptance testing should have been carried out throughout development as opposed to at the end of development. Smaller changes such as adding in the README file were possible at this stage. More significant changes of reverting to the standard Django MVT architecture are changes that would require much more time. Moreover, it was not apparent that the duration needed to execute the web crawler would be so large. The duration of the tests limited the number of tests that could be carried out. Lu et al. proposed the same tests in their paper as had been done in this project. However, the tests were carried out over a variety of topics, as the topic could affect the results of the web crawler. If the project were to be undertaken again, enough time would be left to run crawls over several different topics. As a result average precision,

recall, F-measures and harvest rates could be calculated. Averaging the results would give a complete picture of the performance of the web crawler.

Overall, I think the project was successful. Thanks to a ridged design and the kanban board, development was carried out well and within the allocated time frame. The project does fulfil the main aim of fusing web crawling with machine learning. The testing has shown the Naive Bayes classifier can produce some accurate results. Furthermore, the web crawler itself is capable of crawling millions of web pages in a respectable time for the hardware available. The testing did bring light to a few minor changes that were implemented but also further changes that may be adapted in the future.

## 8.1 Future Work

Some changes could be made to the project after reviewing the results of the test and analysing any feedback. The first change would be as a direct result of the user acceptance testing. The design would be modified to revert to the standard Django architecture of MVT. The MVT architecture would allow for crawling to be carried out without needing to start the crawl via the command line at each depth. This would thereby relinquish the command line of any functionality and have all functionality confined to the web application. With the web application following this new design, deployment onto a server would be the next step. Deployment onto a server with users would provide data about the crawler that can be used to improve the performance.

When running crawls, currently the crawler is crawling all links found on a web page. Crawling all links seems to be the primary reason why the number of links grows at an exponential rate at each crawl depth. Therefore, analysis of the URL may need to be added, before crawling the URL. Analysing the URL would be to determine whether or not the link could be related to the topic at hand. This could be implemented by modifying the parameters of the existing classifier to look for keywords in URLs. The coupling of these improvements with the existing system should put the system in good stead as an excellent example of machine learning and web crawling.

## 9 Conclusion

The overall aim of the project was to combine machine learning and web crawling to produce a focused web crawler. The focused web crawler would be presented as a web application for ease of use and helping to visualise the inner workings of the crawler. The secondary aim is to bring light to a field ruled by multinational search engine companies by providing a solid example of how machine learning can be integrated into web crawling.

These aims have been achieved through a careful design, implementation and testing phase. The design broke the project down into three separate components namely, the web crawler system, the Naive Bayes classifier and the web application. The web crawler system took advantage of the requests[15] package to download web pages and filter out any unnecessary text and code. The Naive Bayes classifier then used the NLTK[17] package to transform this web page data into data sets. These data sets were then used to train the classifier into recognising web pages that are related to the data set. The web application,

using the Django Framework[30], then housed the combination of the classifier and the web crawler system. With the goal of the web application to make the project more accessible and easier to understand.

The results achieved by the classifier, show that the classifier has an 80% simple accuracy at recognising relevant pages. The classifier was further tested by calculating its precision and recall. The precision score was 0.44 and the recall 0.88. These scores are fantastic as a high recall shows the crawler has very little change of incorrectly identifying a relevant page. The recall score is reflected in both the F-measure score of 0.83 and the harvest rate. A high harvest rate of between 0.9 and 0.75 up to 3.5 million web pages show the crawler is good at crawling a majority of relevant websites. These high scores seemed to rival and better some other web crawlers with similar structures. Although these crawlers may use more advanced techniques to calculate relevancy, none of these projects does anything to visualise the results for the end user.

Future works include modifying the design of the web application to remove any command line functionality. A modified design would ensure complete functionality is kept within the web application. Further classification of URLs before crawling them to determine if they may have relevant content could lead to vast increases in performance. Overall, the potential for future work in this project will lead to an exciting challenge for those that may wish to pursue this project further.



## References

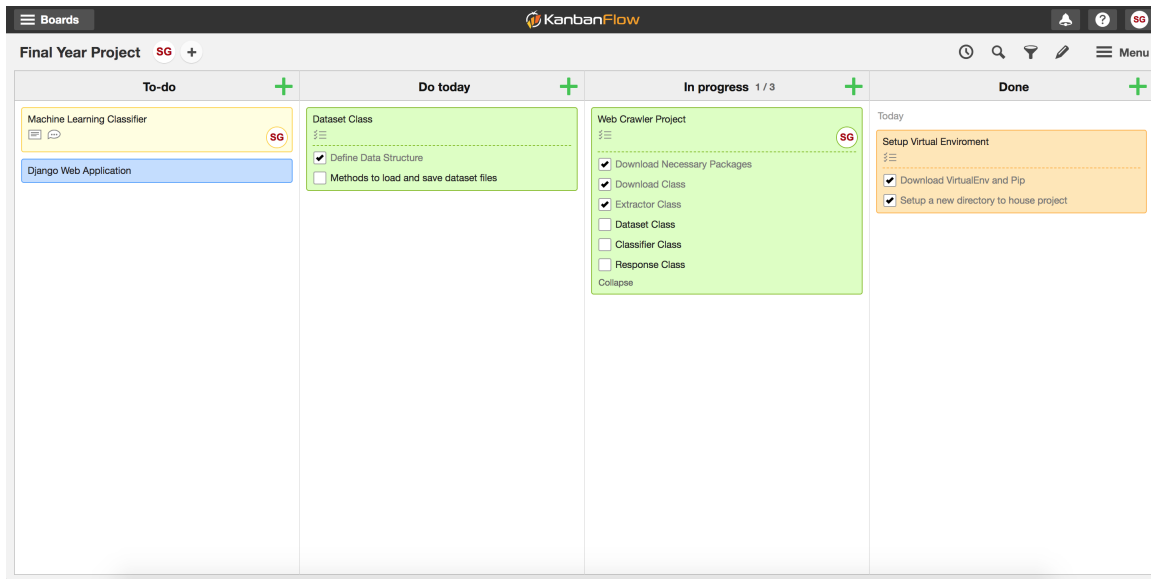
- [1] Timothy J Berners-Lee and Robert Cailliau. World-wide web. 1992.
- [2] Internet usage 2018. URL <https://www.internetworldstats.com/>. [Online; accessed 9-November-2018].
- [3] Md Abu Kausar, VS Dhaka, and Sanjeev Kumar Singh. Web crawler: a review. *International Journal of Computer Applications*, 63(2), 2013.
- [4] S Balan and P Ponmuthuramaling. A study on semantic web mining and web crawler. *International Journal Of Engineering And Computer Science*, 2(09), 2013.
- [5] Alan Bundy and Lincoln Wallen. Breadth-first search. In *Catalogue of Artificial Intelligence Tools*, pages 13–13. Springer, 1984.
- [6] Marco Baroni and Motoko Ueyama. Building general-and special-purpose corpora by web crawling. In *Proceedings of the 13th NIJL international symposium, language corpora: Their compilation and application*, pages 31–40, 2006.
- [7] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer networks*, 31(11-16): 1623–1640, 1999.
- [8] Donald Michie, David J Spiegelhalter, CC Taylor, et al. Machine learning. *Neural and Statistical Classification*, 13, 1994.
- [9] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.
- [10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.
- [11] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.
- [12] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- [13] Chen Jin, Luo De-Lin, and Mu Fen-Xiang. An improved id3 decision tree algorithm. In *Computer Science & Education, 2009. ICCSE'09. 4th International Conference on*, pages 127–130. IEEE, 2009.
- [14] Tanaphol Suebchua, Bundit Manaskasemsak, Arnon Rungsawang, and Hayato Yamana. History-enhanced focused website segment crawler. In *Information Networking (ICOIN), 2018 International Conference on*, pages 80–85. IEEE, 2018.
- [15] Requests package, 2019. URL <http://docs.python-requests.org/en/master/>. [Online; accessed 21-March-2019].

- [16] Pyquery, 2019. URL <https://pythonhosted.org/pyquery/>. [Online; accessed 21-March-2019].
- [17] Natural language toolkit, 2019. URL <https://www.nltk.org/>. [Online; accessed 23-March-2019].
- [18] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer, 1998.
- [19] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.
- [20] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.
- [21] Richard Swinburne. Bayes’ theorem. 2004.
- [22] Vikas P Deshpande, Robert F Erbacher, and Chris Harris. An evaluation of naïve bayesian anti-spam filtering techniques. In *2007 IEEE SMC Information Assurance and Security Workshop*, pages 333–340. IEEE, 2007.
- [23] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Proceedings fifth ieee international enterprise distributed object computing conference*, pages 118–127. IEEE, 2001.
- [24] Pickle, 2019. URL <https://docs.python.org/3/library/pickle.html>. [Online; accessed 23-March-2019].
- [25] Scrapy — a fast and powerful scraping and web crawling framework. URL <https://scrapy.org/>. [Online; accessed 16-November-2018].
- [26] Pip package installer, 2019. URL <https://pypi.org/project/pip/>. [Online; accessed 20-March-2019].
- [27] Virtualenv python environment, 2019. URL <https://pypi.org/project/virtualenv/>. [Online; accessed 20-March-2019].
- [28] Tensorflow. URL <https://www.tensorflow.org/>. [Online; accessed 16-November-2018].
- [29] Re, 2019. URL <https://docs.python.org/3/library/re.html>. [Online; accessed 23-March-2019].
- [30] Django web application framework, 2019. URL <https://www.djangoproject.com/>. [Online; accessed 25-March-2019].
- [31] Postgresql: The world’s most advanced open source database, 2019. URL <https://www.postgresql.org/>. [Online; accessed 29-March-2019].

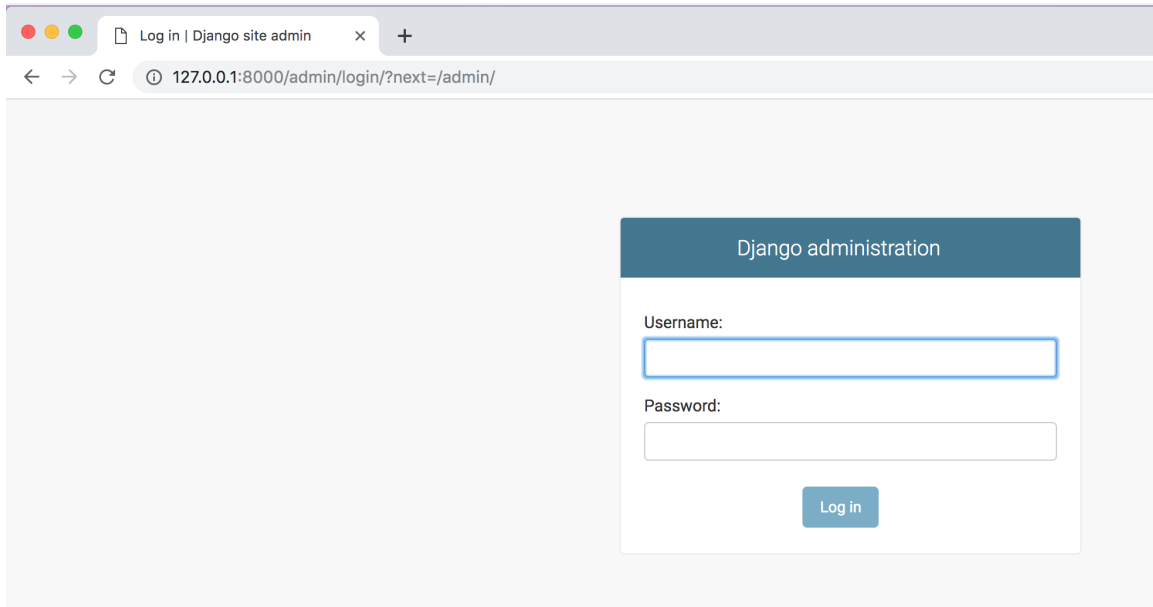
- [32] Michael Buckland and Fredric Gey. The relationship between recall and precision. *Journal of the American society for information science*, 45(1):12–19, 1994.
- [33] Cyril W Cleverdon. On the inverse relationship of recall and precision. *Journal of documentation*, 28(3):195–201, 1972.
- [34] Tao Peng and Lu Liu. Focused crawling enhanced by cbp–slc. *Knowledge-Based Systems*, 51:15–26, 2013.
- [35] David D Lewis et al. Evaluating and optimizing autonomous text classification systems. In *SIGIR*, volume 95, pages 246–254. Citeseer, 1995.
- [36] Ismail Sengor Altingovde and Ozgur Ulusoy. Exploiting interclass rules for focused crawling. *IEEE Intelligent Systems*, 19(6):66–73, 2004.
- [37] Wenxian Wang, Xingshu Chen, Yongbin Zou, Haizhou Wang, and Zongkun Dai. A focused crawler based on naive bayes classifier. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 517–521. IEEE, 2010.
- [38] Houqing Lu, Donghui Zhan, Lei Zhou, and Dengchao He. An improved focused crawler: using web page classification and link priority evaluation. *Mathematical Problems in Engineering*, 2016, 2016.

# Appendices

## A Kanban board during Web Crawler Project Development



## B Django Admin Login Page



## C Model for the *WebsiteTrainingMapping* table

```
class WebsiteTrainingMapping(models.Model):
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    website = models.ForeignKey(Website, on_delete=models.CASCADE)
    training = models.ForeignKey(Training, on_delete=models.CASCADE)
    done = models.BooleanField(default=False)
```

## D ModelAdmin for the *WebsiteTrainingMapping* table

```
class WebsiteTrainingMappingAdmin(admin.ModelAdmin):  
    list_display = (  
        'website',  
        'training',  
        'done',  
    )
```

# E Web Application Main Page

Django administration

WELCOME, **root**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	<a href="#">+ Add</a>	<a href="#">Change</a>
Users	<a href="#">+ Add</a>	<a href="#">Change</a>

CORE

Errors	<a href="#">+ Add</a>	<a href="#">Change</a>
Relevances	<a href="#">+ Add</a>	<a href="#">Change</a>
Settings	<a href="#">+ Add</a>	<a href="#">Change</a>
Training historys	<a href="#">+ Add</a>	<a href="#">Change</a>
Trainings	<a href="#">+ Add</a>	<a href="#">Change</a>
Website training mappings	<a href="#">+ Add</a>	<a href="#">Change</a>
Websites	<a href="#">+ Add</a>	<a href="#">Change</a>

Recent actions

My actions

✖

<https://www.thisismoney.co.uk/m...>

Website

✖

<https://www.carmagazine.co.uk/>

Website

✖

<https://www.whatcar.com/>

Website

✖

<https://www.autocar.co.uk/car-news>

Website

✖

<https://www.theguardian.com/tec...self-driving-car-unit-valued-at-73bn-as-it-gears-up-for-ipo>

Website

✖

<http://www.exeter.ac.uk/sustainab...>

Website

✖

<https://inews.co.uk/essentials/life...news/why-you-never-share-photos-driving-licence-car-documents-dvla-warning/>

Website

✖

<https://www.telegraph.co.uk/cars/>

Website

✖

<https://edition.cnn.com/2019/04/...cars-missing-car-share-app-trnd/index.html>

Website



# F Websites Page

Django administration

WELCOME, **ROOT** / VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Core > Websites

Select website to change

ADD WEBSITE +

Action:  Go 0 of 2 selected

<input type="checkbox"/>	URL	CREATED	UPDATED	FAIL COUNT	IS CRAWLED	PARENT	DEPTH
<input type="checkbox"/>	<a href="https://www.bbcgoodfood.com/recipes/easter-nest-cake">https://www.bbcgoodfood.com/recipes/easter-nest-cake</a>	April 20, 2019, 10:21 a.m.	April 20, 2019, 10:21 a.m.	0	<div></div>	-	0
<input type="checkbox"/>	<a href="https://www.bbcgoodfood.com/howto/guide/health-benefits-broccoli">https://www.bbcgoodfood.com/howto/guide/health-benefits-broccoli</a>	April 20, 2019, 10:21 a.m.	April 20, 2019, 10:21 a.m.	0	<div></div>	-	0

2 websites

FILTER

By is crawled

All

Yes

No

## G Edit Website Entry Page

Django administration

WELCOME, **root**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Core > Websites > <https://www.bbcgoodfood.com/recipes/easter-nest-cake>

Change website

HISTORY

Url:


Currently: <https://www.bbcgoodfood.com/recipes/easter-nest-cake>

Change:

☐ Is crawled

Fall count:

Parent:



Depth:

Delete

Save and add another

Save and continue editing

SAVE

## H Cross Browser Compatibility Checks

Checks	Browser			
	Google Chrome	Mozilla Firefox	Microsoft Edge	Safari
Login page loads correctly	✓	✓	✓	✓
User is able to login	✓	✓	✓	✓
All pages are able to be accessed and load without fault	✓	✓	✓	✓
Training set is able to be created	✓	✓	✓	✓
Training set is able to be loaded	✓	✓	✓	✓
Websites can be added to a training set	✓	✓	✓	✓
Already crawled websites can be added to a training set	✓	✓	✓	✓
Model is succesfully trained	✓	✓	✓	✓
Trained Websites are logged in Training Historys Table	✓	✓	✓	✓
Settings are able to set and saved	✓	✓	✓	✓
Websites are successfully crawled	✓	✓	✓	✓
Websites from crawl are added to websites table	✓	✓	✓	✓
Websites with a relevancy score exceeding the value in setting are added to website training mappings table	✓	✓	✓	✓
Websites that throw errors are logged in the Errors Table	✓	✓	✓	✓
Crawled websites are being logged in Relevances Table	✓	✓	✓	✓
Update and Delete commands across all tables	✓	✓	✓	✓

## I User Acceptance Form

### Web Crawler User Acceptance Testing

Q1) Did you find the layout of the web application facilitated easy navigation from menu to menu?

Yes ☐ No ☐

Q2) Did you find that information was clearly laid out?

Yes ☐ No ☐

Q3) Did you have all the information you required? If no, please explain what information is missing.

Yes ☐ No ☐

.....  
.....

Q4) Did you find it easy to setup a crawl?

Yes ☐ No ☐

Q5) Did you find it simple to analyse the results of the crawl?

Yes ☐ No ☐

Q6) Can you suggest any improvements that could be made in relation to use of the application?

.....  
.....

## J Food Recipe Training Set

URL	Score
<a href="https://www.allrecipes.com/">https://www.allrecipes.com/</a>	0.76
<a href="https://www.foodnetwork.com/">https://www.foodnetwork.com/</a>	0.85
<a href="https://www.bettycrocker.com/recipes">https://www.bettycrocker.com/recipes</a>	0.79
<a href="https://www.myrecipes.com/">https://www.myrecipes.com/</a>	0.85
<a href="https://www.delish.com/cooking/recipe-ideas/">https://www.delish.com/cooking/recipe-ideas/</a>	0.75
<a href="https://www.foodnetwork.com/recipes">https://www.foodnetwork.com/recipes</a>	0.84
<a href="https://www.tasteofhome.com/">https://www.tasteofhome.com/</a>	0.81
<a href="https://www.tasteofhome.com/recipes/">https://www.tasteofhome.com/recipes/</a>	0.85
<a href="https://www.marthastewart.com/1505788/recipes">https://www.marthastewart.com/1505788/recipes</a>	0.88
<a href="https://www.cooks.com/">https://www.cooks.com/</a>	0.81
<a href="https://www.pillsbury.com/recipes">https://www.pillsbury.com/recipes</a>	0.98
<a href="https://www.simplyrecipes.com/">https://www.simplyrecipes.com/</a>	0.87
<a href="https://www.southernliving.com/food/kitchen-assistant">https://www.southernliving.com/food/kitchen-assistant</a>	0.8
<a href="https://www.wholefoodsmarket.com/recipes">https://www.wholefoodsmarket.com/recipes</a>	0.75
<a href="https://www.myrecipes.com/recipe-finder">https://www.myrecipes.com/recipe-finder</a>	0.95
<a href="https://www.health.com/recipes">https://www.health.com/recipes</a>	0.85
<a href="https://www.serious-eats.com/recipes">https://www.serious-eats.com/recipes</a>	0.97
<a href="http://www.recipe.com/">http://www.recipe.com/</a>	0.85
<a href="https://www.pauladeen.com/recipes/">https://www.pauladeen.com/recipes/</a>	0.75
<a href="https://www.epicurious.com/">https://www.epicurious.com/</a>	0.77
<a href="https://www.beefitswhatsfordinner.com/recipes">https://www.beefitswhatsfordinner.com/recipes</a>	0.94
<a href="https://www.bhg.com/recipes/">https://www.bhg.com/recipes/</a>	0.95
<a href="https://www.kingarthurflour.com/recipes/">https://www.kingarthurflour.com/recipes/</a>	0.79
<a href="https://www.chowhound.com/recipes">https://www.chowhound.com/recipes</a>	0.85
<a href="https://www.epicurious.com/recipes-menus">https://www.epicurious.com/recipes-menus</a>	0.88
<a href="https://www.traderjoes.com/recipes">https://www.traderjoes.com/recipes</a>	0.93
<a href="https://food.ndtv.com/recipes">https://food.ndtv.com/recipes</a>	0.9
<a href="https://www.kroger.com/recipes">https://www.kroger.com/recipes</a>	0.91
<a href="https://www.yummly.com/?prm-v1=1">https://www.yummly.com/?prm-v1=1</a>	0.98
<a href="https://www.allrecipes.com/">https://www.allrecipes.com/</a>	0.81
<a href="https://steemit.com/@foodrecipies">https://steemit.com/@foodrecipies</a>	0.97
<a href="https://www.mrfood.com/">https://www.mrfood.com/</a>	0.84
<a href="https://www.pinterest.com/kellyddecke/foodrecipies/">https://www.pinterest.com/kellyddecke/foodrecipies/</a>	0.75
<a href="https://www.pinterest.com/bricktaylor01/foodrecipies/">https://www.pinterest.com/bricktaylor01/foodrecipies/</a>	0.78
<a href="https://www.facebook.com/FoodRecipies/">https://www.facebook.com/FoodRecipies/</a>	0.8
<a href="http://www.foodrecipies.xyz/">http://www.foodrecipies.xyz/</a>	0.98
<a href="https://www.bbcgoodfood.com/recipes">https://www.bbcgoodfood.com/recipes</a>	0.98
<a href="https://www.allfood.recipes/">https://www.allfood.recipes/</a>	0.81
<a href="https://www.myrecipes.com/world-cuisine/american-recipes/recipes-with-soul">https://www.myrecipes.com/world-cuisine/american-recipes/recipes-with-soul</a>	0.81
<a href="https://www.allrecipes.com/recipe/140286/homemade-dog-food/">https://www.allrecipes.com/recipe/140286/homemade-dog-food/</a>	0.9
<a href="https://www.soulfoodandsoutherncooking.com/">https://www.soulfoodandsoutherncooking.com/</a>	0.8

<a href="https://www.moneycrashers.com/homemade-dog-food-treat-recipes/">https://www.moneycrashers.com/homemade-dog-food-treat-recipes/</a>	0.77
<a href="https://www.yummly.com/recipes/southern-black-soul-food?prm-v1=1">https://www.yummly.com/recipes/southern-black-soul-food?prm-v1=1</a>	0.77
<a href="https://www.parenting.com/baby/homemade-baby-food-recipes">https://www.parenting.com/baby/homemade-baby-food-recipes</a>	0.82
<a href="https://foodrecipies-harindersabharwal.blogspot.com/">https://foodrecipies-harindersabharwal.blogspot.com/</a>	0.99
<a href="http://www.mexican-authentic-recipes.com/">http://www.mexican-authentic-recipes.com/</a>	0.9
<a href="https://www.bbcgoodfood.com/recipes/2364/mushroom-risotto">https://www.bbcgoodfood.com/recipes/2364/mushroom-risotto</a>	0.88
<a href="https://www.greatbritishchefs.com/ingredients/pasta-recipes">https://www.greatbritishchefs.com/ingredients/pasta-recipes</a>	0.79
<a href="https://www.deliciousmagazine.co.uk/ingredients/pasta-recipes/">https://www.deliciousmagazine.co.uk/ingredients/pasta-recipes/</a>	0.92
<a href="https://www.bbcgoodfood.com/recipes/category/cakes-baking">https://www.bbcgoodfood.com/recipes/category/cakes-baking</a>	0.96
<a href="https://www.oliviascuisine.com/authentic-brazilian-recipes/">https://www.oliviascuisine.com/authentic-brazilian-recipes/</a>	0.93
<a href="https://www.bbc.com/food/recipes/moroccanlambtagine_6696">https://www.bbc.com/food/recipes/moroccanlambtagine_6696</a>	0.93
<a href="https://www.desertsun.com/">https://www.desertsun.com/</a>	0.95
<a href="https://www.allrecipes.com/recipes/1788/healthy-recipes/low-sodium/">https://www.allrecipes.com/recipes/1788/healthy-recipes/low-sodium/</a>	0.81
<a href="https://www.facebook.com/Food-Recipes-228248023852823/">https://www.facebook.com/Food-Recipes-228248023852823/</a>	0.9
<a href="https://apkpure.com/sambhar-recipes-in-hindi/">https://apkpure.com/sambhar-recipes-in-hindi/</a>	0.81
<a href="https://www.wikihow.com/Make-a-Medieval-Feast">https://www.wikihow.com/Make-a-Medieval-Feast</a>	0.85
<a href="https://www.foodnetwork.com/topics/low-fat-recipes">https://www.foodnetwork.com/topics/low-fat-recipes</a>	0.97
<a href="https://recipes.sparkpeople.com/cookbooks.asp?cookbook=524597">https://recipes.sparkpeople.com/cookbooks.asp?cookbook=524597</a>	0.75
<a href="https://www.foodnetwork.com/magazine">https://www.foodnetwork.com/magazine</a>	0.97
<a href="https://www.allrecipes.com/recipes/87/everyday-cooking/vegetarian/">https://www.allrecipes.com/recipes/87/everyday-cooking/vegetarian/</a>	0.86
<a href="https://www.foodnetwork.com/topics/vegetarian-recipes">https://www.foodnetwork.com/topics/vegetarian-recipes</a>	0.86
<a href="http://www.vegetarianrecipes.net/">http://www.vegetarianrecipes.net/</a>	0.86
<a href="https://www.cookinglight.com/food/top-rated-recipes/best-vegetarian-recipes">https://www.cookinglight.com/food/top-rated-recipes/best-vegetarian-recipes</a>	0.82
<a href="https://www.delish.com/content/vegetarian-recipes/">https://www.delish.com/content/vegetarian-recipes/</a>	0.98
<a href="https://www.vegetariantimes.com/recipes">https://www.vegetariantimes.com/recipes</a>	0.95
<a href="https://www.myrecipes.com/vegetarian-recipes">https://www.myrecipes.com/vegetarian-recipes</a>	0.92
<a href="https://www.foodandwine.com/slideshows/vegetarian-recipes">https://www.foodandwine.com/slideshows/vegetarian-recipes</a>	0.98
<a href="https://www.marthastewart.com/1513023/vegetarian-vegan-recipes">https://www.marthastewart.com/1513023/vegetarian-vegan-recipes</a>	0.79
<a href="https://www.allrecipes.com/recipes/265/everyday-cooking/vegetarian/main-dishes/">https://www.allrecipes.com/recipes/265/everyday-cooking/vegetarian/main-dishes/</a>	0.96
<a href="https://www.instagram.com/explore/tags/vegetarianrecipes/">https://www.instagram.com/explore/tags/vegetarianrecipes/</a>	0.91
<a href="https://www.bbcgoodfood.com/recipes/category/vegetarian">https://www.bbcgoodfood.com/recipes/category/vegetarian</a>	0.95
<a href="https://www.jamieoliver.com/recipes/category/special-diets/vegetarian/">https://www.jamieoliver.com/recipes/category/special-diets/vegetarian/</a>	0.9
<a href="https://www.tasteofhome.com/course/vegetarian-recipes/">https://www.tasteofhome.com/course/vegetarian-recipes/</a>	0.96
<a href="https://www.101cookbooks.com/vegetarian_recipes">https://www.101cookbooks.com/vegetarian_recipes</a>	0.76
<a href="https://www.countryliving.com/food-drinks/g1186/vegetarian-recipes-0309/">https://www.countryliving.com/food-drinks/g1186/vegetarian-recipes-0309/</a>	0.82
<a href="http://www.vegetarianrecipes.net/about/">http://www.vegetarianrecipes.net/about/</a>	0.88
<a href="https://www.womansday.com/food-recipes/food-drinks/g2373/vegetarian-recipes/">https://www.womansday.com/food-recipes/food-drinks/g2373/vegetarian-recipes/</a>	0.89
<a href="http://www.eatingwell.com/recipes/18005/lifestyle-diets/vegetarian/">http://www.eatingwell.com/recipes/18005/lifestyle-diets/vegetarian/</a>	0.88
<a href="https://www.bbc.com/food/diets/vegetarian">https://www.bbc.com/food/diets/vegetarian</a>	0.76
<a href="https://www.realsimple.com/food-recipes/">https://www.realsimple.com/food-recipes/</a>	0.97
<a href="https://ohmyveggies.com/category/vegetarian-recipes/">https://ohmyveggies.com/category/vegetarian-recipes/</a>	0.9
<a href="https://thevegetarianrecipes.blogspot.com/">https://thevegetarianrecipes.blogspot.com/</a>	0.76
<a href="https://www.marthastewart.com/360627/kid-friendly-vegetarian-recipes">https://www.marthastewart.com/360627/kid-friendly-vegetarian-recipes</a>	0.91
<a href="https://www.delish.com/cooking/g826/vegetarian-soup/">https://www.delish.com/cooking/g826/vegetarian-soup/</a>	0.83

<a href="https://www.bettycrocker.com/recipes/health-and-diet/vegetarian-recipes">https://www.bettycrocker.com/recipes/health-and-diet/vegetarian-recipes</a>	0.78
<a href="https://www.waitrose.com/home/recipes/quick_and_easy_recipes/">https://www.waitrose.com/home/recipes/quick_and_easy_recipes/</a>	0.97
<a href="https://www.peta.org/recipes/">https://www.peta.org/recipes/</a>	0.89
<a href="https://www.reddit.com/r/vegetarianrecipes/">https://www.reddit.com/r/vegetarianrecipes/</a>	0.8
<a href="https://www.cookinglight.com/food/vegetarian/simple-vegetarian-recipes">https://www.cookinglight.com/food/vegetarian/simple-vegetarian-recipes</a>	0.75
<a href="https://www.allrecipes.com/recipes/1227/everyday-cooking/vegan/">https://www.allrecipes.com/recipes/1227/everyday-cooking/vegan/</a>	0.85
<a href="https://www.foodnetwork.com/topics/vegan">https://www.foodnetwork.com/topics/vegan</a>	0.86
<a href="https://www.realsimple.com/food-recipes/recipe-collections-favorites/">https://www.realsimple.com/food-recipes/recipe-collections-favorites/</a>	0.89
<a href="https://www.goodhousekeeping.com/food-recipes/healthy/g807/vegan-recipes/">https://www.goodhousekeeping.com/food-recipes/healthy/g807/vegan-recipes/</a>	0.8
<a href="https://www.101cookbooks.com/vegan_recipes">https://www.101cookbooks.com/vegan_recipes</a>	0.94
<a href="https://www.jamieoliver.com/recipes/category/special-diets/vegan/">https://www.jamieoliver.com/recipes/category/special-diets/vegan/</a>	0.95
<a href="https://www.peta.org/recipes/">https://www.peta.org/recipes/</a>	0.91
<a href="https://www.cookinglight.com/food/vegetarian/vegan-recipes">https://www.cookinglight.com/food/vegetarian/vegan-recipes</a>	0.85
<a href="https://www.reddit.com/r/veganrecipes/">https://www.reddit.com/r/veganrecipes/</a>	0.86
<a href="https://www.vegansociety.com/resources/recipes">https://www.vegansociety.com/resources/recipes</a>	0.77

---

## K Food Recipe Test Set 1: Random URLs

URL	Score
<a href="https://www.nationalgeographic.com/animals/">https://www.nationalgeographic.com/animals/</a>	0.36
<a href="https://www.youtube.com/watch?v=gCYcHz2k5x0">https://www.youtube.com/watch?v=gCYcHz2k5x0</a>	ERROR
<a href="https://a-z-animals.com/animals/">https://a-z-animals.com/animals/</a>	0.44
<a href="https://kids.nationalgeographic.com/animals/">https://kids.nationalgeographic.com/animals/</a>	0.47
<a href="https://www.imdb.com/title/tt4953128/">https://www.imdb.com/title/tt4953128/</a>	0.25
<a href="https://animals.sandiegozoo.org/animals">https://animals.sandiegozoo.org/animals</a>	0.17
<a href="https://www.huffpost.com/impact/topic/animals">https://www.huffpost.com/impact/topic/animals</a>	0.28
<a href="https://kids.sandiegozoo.org/animals">https://kids.sandiegozoo.org/animals</a>	0.36
<a href="https://www.hbo.com/animals">https://www.hbo.com/animals</a>	0.41
<a href="https://www.thefreedictionary.com/Animals">https://www.thefreedictionary.com/Animals</a>	ERROR
<a href="https://www.pexels.com/search/animals/">https://www.pexels.com/search/animals/</a>	0.11
<a href="http://www.animalplanet.com/wild-animals/">http://www.animalplanet.com/wild-animals/</a>	0.3
<a href="https://www.nytimes.com/topic/subject/animals">https://www.nytimes.com/topic/subject/animals</a>	0.29
<a href="https://en.wikipedia.org/wiki/Animal">https://en.wikipedia.org/wiki/Animal</a>	0.2
<a href="https://nationalzoo.si.edu/animals">https://nationalzoo.si.edu/animals</a>	0.56
<a href="https://detroitzoo.org/animals/zoo-animals/">https://detroitzoo.org/animals/zoo-animals/</a>	0.43
<a href="https://www.merriam-webster.com/dictionary/animal">https://www.merriam-webster.com/dictionary/animal</a>	0.11
<a href="https://www.britannica.com/topic-browse/Animals">https://www.britannica.com/topic-browse/Animals</a>	0.3
<a href="https://www.msn.com/en-us/video/animals">https://www.msn.com/en-us/video/animals</a>	0.27
<a href="https://kids.nationalgeographic.com/">https://kids.nationalgeographic.com/</a>	0.31
<a href="https://www.npr.org/sections/animals/">https://www.npr.org/sections/animals/</a>	0.31
<a href="https://www.theguardian.com/world/animals">https://www.theguardian.com/world/animals</a>	0.18
<a href="https://www.imdb.com/title/tt4426738/">https://www.imdb.com/title/tt4426738/</a>	0.19
<a href="https://www.rottentomatoes.com/m/animals_2014/">https://www.rottentomatoes.com/m/animals_2014/</a>	0.46
<a href="https://www.popsci.com/animals">https://www.popsci.com/animals</a>	0.49
<a href="https://www.animaljam.com/welcome">https://www.animaljam.com/welcome</a>	0.24
<a href="https://a-z-animals.com/">https://a-z-animals.com/</a>	0.34
<a href="https://www.coloring.ws/animals.html">https://www.coloring.ws/animals.html</a>	0.11
<a href="https://www.nps.gov/ever/learn/nature/animals.html">https://www.nps.gov/ever/learn/nature/animals.html</a>	0.22
<a href="https://www.nationalgeographic.com/animals/">https://www.nationalgeographic.com/animals/</a>	0.39
<a href="https://www.youtube.com/watch?v=gCYcHz2k5x0">https://www.youtube.com/watch?v=gCYcHz2k5x0</a>	ERROR
<a href="https://a-z-animals.com/animals/">https://a-z-animals.com/animals/</a>	0.36
<a href="https://kids.nationalgeographic.com/animals/">https://kids.nationalgeographic.com/animals/</a>	0.22
<a href="https://www.imdb.com/title/tt4953128/">https://www.imdb.com/title/tt4953128/</a>	0.19
<a href="https://animals.sandiegozoo.org/animals">https://animals.sandiegozoo.org/animals</a>	0.43
<a href="https://www.huffpost.com/impact/topic/animals">https://www.huffpost.com/impact/topic/animals</a>	0.52
<a href="https://kids.sandiegozoo.org/animals">https://kids.sandiegozoo.org/animals</a>	0.18
<a href="https://www.hbo.com/animals">https://www.hbo.com/animals</a>	0.19
<a href="https://www.thefreedictionary.com/Animals">https://www.thefreedictionary.com/Animals</a>	ERROR
<a href="https://www.pexels.com/search/animals/">https://www.pexels.com/search/animals/</a>	0.32
<a href="http://www.animalplanet.com/wild-animals/">http://www.animalplanet.com/wild-animals/</a>	0.19



<a href="https://www.nytimes.com/topic/subject/animals">https://www.nytimes.com/topic/subject/animals</a>	0.17
<a href="https://en.wikipedia.org/wiki/Animal">https://en.wikipedia.org/wiki/Animal</a>	0.55
<a href="https://nationalzoo.si.edu/animals">https://nationalzoo.si.edu/animals</a>	0.35
<a href="https://detroitzoo.org/animals/zoo-animals/">https://detroitzoo.org/animals/zoo-animals/</a>	0.4
<a href="https://www.merriam-webster.com/dictionary/animal">https://www.merriam-webster.com/dictionary/animal</a>	0.15
<a href="https://www.britannica.com/topic-browse/Animals">https://www.britannica.com/topic-browse/Animals</a>	0.25
<a href="https://www.msn.com/en-us/video/animals">https://www.msn.com/en-us/video/animals</a>	ERROR
<a href="https://kids.nationalgeographic.com/">https://kids.nationalgeographic.com/</a>	0.34
<a href="https://www.npr.org/sections/animals/">https://www.npr.org/sections/animals/</a>	0.17
<a href="https://www.theguardian.com/world/animals">https://www.theguardian.com/world/animals</a>	0.58

---

## L Food Recipe Test Set 2: Food Recipe URLs

URL	Score
<a href="https://www.allrecipes.com/recipes/201/meat-and-poultry/chicken/">https://www.allrecipes.com/recipes/201/meat-and-poultry/chicken/</a>	0.95
<a href="https://www.foodnetwork.com/topics/chicken">https://www.foodnetwork.com/topics/chicken</a>	0.51
<a href="https://www.cookinglight.com/food/quick-healthy/easy-chicken-recipes">https://www.cookinglight.com/food/quick-healthy/easy-chicken-recipes</a>	0.57
<a href="http://chickenrecipes.org/">http://chickenrecipes.org/</a>	0.73
<a href="https://www.myrecipes.com/chicken-recipes">https://www.myrecipes.com/chicken-recipes</a>	0.82
<a href="https://www.countryliving.com/food-drinks/g680/chicken-recipes-0109/">https://www.countryliving.com/food-drinks/g680/chicken-recipes-0109/</a>	0.94
<a href="https://www.tasteofhome.com/ingredients/chicken-recipes/">https://www.tasteofhome.com/ingredients/chicken-recipes/</a>	0.93
<a href="https://www.bettycrocker.com/recipes/main-ingredient/chicken-recipes">https://www.bettycrocker.com/recipes/main-ingredient/chicken-recipes</a>	0.71
<a href="https://www.delish.com/chicken-breast-recipes/">https://www.delish.com/chicken-breast-recipes/</a>	0.7
<a href="https://www.simplyrecipes.com/recipes/ingredient/chicken/">https://www.simplyrecipes.com/recipes/ingredient/chicken/</a>	0.76
<a href="https://www.allrecipes.com/recipes/659/meat-and-poultry/chicken/chicken-breasts/">https://www.allrecipes.com/recipes/659/meat-and-poultry/chicken/chicken-breasts/</a>	0.67
<a href="https://www.pillsbury.com/recipes/ingredient/chicken">https://www.pillsbury.com/recipes/ingredient/chicken</a>	0.76
<a href="https://www.marthastewart.com/1504489/chicken-recipes">https://www.marthastewart.com/1504489/chicken-recipes</a>	0.88
<a href="https://www.delish.com/cooking/recipe-ideas/g2972/chicken-weeknight-dinners/">https://www.delish.com/cooking/recipe-ideas/g2972/chicken-weeknight-dinners/</a>	0.67
<a href="https://www.bhg.com/recipes/chicken/">https://www.bhg.com/recipes/chicken/</a>	0.58
<a href="https://www.simplyrecipes.com/recipes/ingredient/chicken/page/2/">https://www.simplyrecipes.com/recipes/ingredient/chicken/page/2/</a>	0.76
<a href="https://www.allchickenrecipes.com/">https://www.allchickenrecipes.com/</a>	0.84
<a href="https://www.marthastewart.com/1502529/baked-chicken-recipes">https://www.marthastewart.com/1502529/baked-chicken-recipes</a>	0.48
<a href="http://dish.allrecipes.com/top-5-fried-chicken-recipes/">http://dish.allrecipes.com/top-5-fried-chicken-recipes/</a>	0.54
<a href="https://www.skinnytaste.com/main-ingredient/chicken-recipes/">https://www.skinnytaste.com/main-ingredient/chicken-recipes/</a>	0.81
<a href="http://www.eatingwell.com/recipes/18241/ingredients/meat-poultry/chicken/">http://www.eatingwell.com/recipes/18241/ingredients/meat-poultry/chicken/</a>	0.62
<a href="https://food.ndtv.com/lists/10-best-indian-chicken-recipes-693207">https://food.ndtv.com/lists/10-best-indian-chicken-recipes-693207</a>	ERROR
<a href="https://www.pinterest.com/explore/chicken-recipes/">https://www.pinterest.com/explore/chicken-recipes/</a>	0.83
<a href="https://www.readyseteat.com/recipes/easy-chicken-recipes">https://www.readyseteat.com/recipes/easy-chicken-recipes</a>	0.96
<a href="https://www.facebook.com/tastychickenrecipes">https://www.facebook.com/tastychickenrecipes</a>	ERROR
<a href="https://www.bbcgoodfood.com/recipes/collection/chicken">https://www.bbcgoodfood.com/recipes/collection/chicken</a>	0.73
<a href="https://www.eatthis.com/healthy-chicken-recipes/">https://www.eatthis.com/healthy-chicken-recipes/</a>	0.79
<a href="https://www.thespruceeats.com/chicken-dishes-4162802">https://www.thespruceeats.com/chicken-dishes-4162802</a>	0.58
<a href="https://www.allrecipes.com/recipe/85815/thai-red-chicken-curry/">https://www.allrecipes.com/recipe/85815/thai-red-chicken-curry/</a>	0.92
<a href="https://socviewer.com/tag/thaicurryrecipe">https://socviewer.com/tag/thaicurryrecipe</a>	0.44
<a href="https://www.onceuponachef.com/recipes/thai-shrimp-curry.html">https://www.onceuponachef.com/recipes/thai-shrimp-curry.html</a>	0.95
<a href="https://www.epicurious.com/recipes/food/views/thai-shrimp-curry-109161">https://www.epicurious.com/recipes/food/views/thai-shrimp-curry-109161</a>	0.88
<a href="https://www.foodnetwork.com/topics/curry">https://www.foodnetwork.com/topics/curry</a>	0.52
<a href="https://www.simplyrecipes.com/recipes/ingredient/curry/">https://www.simplyrecipes.com/recipes/ingredient/curry/</a>	0.72
<a href="https://www.allrecipes.com/recipe/25881/basic-curry-sauce/">https://www.allrecipes.com/recipe/25881/basic-curry-sauce/</a>	0.96
<a href="https://www.onceuponachef.com/recipes/chicken-curry.html">https://www.onceuponachef.com/recipes/chicken-curry.html</a>	0.9
<a href="https://www.bonappetit.com/recipes/family-meals/slideshow/curry-recipes">https://www.bonappetit.com/recipes/family-meals/slideshow/curry-recipes</a>	0.9
<a href="https://www.jamieoliver.com/recipes/category/dishtype/curry/">https://www.jamieoliver.com/recipes/category/dishtype/curry/</a>	0.59
<a href="https://www.delish.com/cooking/recipe-ideas/recipes/">https://www.delish.com/cooking/recipe-ideas/recipes/</a>	0.93
<a href="https://www.bbcgoodfood.com/recipes/collection/curry">https://www.bbcgoodfood.com/recipes/collection/curry</a>	0.81
<a href="https://cookieandkate.com/2015/thai-red-curry-recipe/">https://cookieandkate.com/2015/thai-red-curry-recipe/</a>	0.61

<a href="https://www.spendwithpennies.com/chicken-curry/">https://www.spendwithpennies.com/chicken-curry/</a>	0.91
<a href="https://www.recipetineats.com/category/collections/curry-recipes/">https://www.recipetineats.com/category/collections/curry-recipes/</a>	0.93
<a href="https://www.justonecookbook.com/simple-chicken-curry/">https://www.justonecookbook.com/simple-chicken-curry/</a>	0.52
<a href="https://www.bbc.com/food/recipes/simple_chicken_curry_95336">https://www.bbc.com/food/recipes/simple_chicken_curry_95336</a>	0.87
<a href="https://www.allrecipes.com/recipes/250/main-dish/pizza/">https://www.allrecipes.com/recipes/250/main-dish/pizza/</a>	0.49
<a href="https://www.foodnetwork.com/recipes/articles/50-easy-pizzas">https://www.foodnetwork.com/recipes/articles/50-easy-pizzas</a>	0.58
<a href="https://www.tasteofhome.com/course/dinner-recipes/pizza-recipes/">https://www.tasteofhome.com/course/dinner-recipes/pizza-recipes/</a>	0.8
<a href="https://www.delish.com/cooking/g269/homemade-pizza-recipes/">https://www.delish.com/cooking/g269/homemade-pizza-recipes/</a>	0.58

---