Classification of higher education web pages against key information requirements

Chris Green, MSc Computing. School of Computer Science and Informatics, Cardiff University.

Supervised by Professor Irena Spasic.

November 2016.

Abstract

Higher education (HE) providers in the UK currently submit summary information about their courses for display on the Unistats website. A course's information is known as a Key Information Set (KIS). Recent sector consultations have suggested a move away from central presentation, towards mandating HE providers to display KIS on their own websites. In order to regulate this information in the student interest, some judgement may be required on its compliance with regulations. This project investigated the possibility of reducing human effort in regulating the future display of KIS. The main problem associated with regulating information on websites is one of text classification. Supervised Machine Learning (SML) algorithms were investigated for use in text classification within this specific domain. They were assessed according to several performance metrics, and benchmarked against both random and rule-based classification approaches. The current availability of KIS information on HE websites was not favourable in producing a SML classifier to seek other KIS information. However, success was found in using a Naïve Bayes classifier to generally identify course information from non-course information. The main recommendation from this project is that automatic classification is possible in this domain, and will become more viable for KIS information as universities update their pages. A Naïve Bayes classifier for identifying course information, decorated with a key word classifier to flag potential KIS information, could be the best solution.

Acknowledgments

This project is the culmination of an MSc undertaken part-time, with the support and funding of my employer the Higher Education Funding Council for England (HEFCE). Accordingly, I would like to extend my thanks to HEFCE for the significant investment that they have made in my future, and their commitment to improving staff skills through training.

Specific thanks to: Dr Richard Puttock (HEFCE), you have been consistent in allowing me time to expand my knowledge and apply it in a professional context; Professor Irena Spasic (Cardiff University), for helping me shape this project as a novice to machine learning; Ferielle Jarrett, Mike Allaway and Kit Rothwell, you have all provided vital personal support over the last two years of study.

Source Code

All of the source code and other artefacts generated during this project are available at https://github.com/GreenC90/project.

Contents

Introduction

Context

Key Information Sets and Unistats

Since 2007, prospective higher education (HE) students have been able to access information on HE courses through the Unistats web site (HEFCE, 2012). Information on the original Unistats website was primarily from the National Student Survey (NSS) and Destination of Leavers from Higher Education survey (DLHE). In November 2010 HEFCE published an open consultation on public information about higher education, specifically on changes to the information published by institutions (HEFCE et al., 2010). The consultation proposed the creation of a Key Information Set (KIS) for each course at an institution, where applicable. Contents of the proposed KIS were based on research aiming to find out what information prospective students find most useful for determining the relevance of the course to their academic preferences.

Outcomes and responses to the consultation were published in June 2011. They were broadly positive and HEFCE's document provided "next steps" for applicable institutions in England (HEFCE et al., 2011). Through further technical guidance (HEFCE, 2011) and updates, the KIS structure and plans for its display on Unistats were refined. Originally it was intended that universities display each full KIS on their own website, but this was found to be impractical by the sector. Instead, it was decided that a new version of Unistats would be launched in September 2012 and act as a central point for KIS (HEFCE, 2012). Universities would display an extract of KIS data on their own course pages via a widget linking directly to the corresponding Unistats page.

Unistats has remained the central location for KIS data since the redesigned site was launched in 2012. In 2015 the HE funding councils launched a joint consultation on the future of Unistats, the NSS and information to be published by institutions from 2017 (DELNI et al., 2015a). Information published by institutions is the data focus of this project, relevant responses to the 2015 consultation and expected next steps were publicised in August 2016 (HEFCE et al., 2016). Amongst other changes the HE funding bodies agreed: to remove detailed course information from Unistats; transfer responsibility of tuition fee display to institutions; and develop requirements for the way information is presented on institutional websites, these guidelines will be complete in late 2016 for implementation by September 2017.

The exact data items that the funding councils expect institutions to display for the 2017-18 academic year are yet to be finalised; fee information has already been agreed, but as the scope of Unistats is reduced then institutions could viably be asked to display some of the data that was previously available.

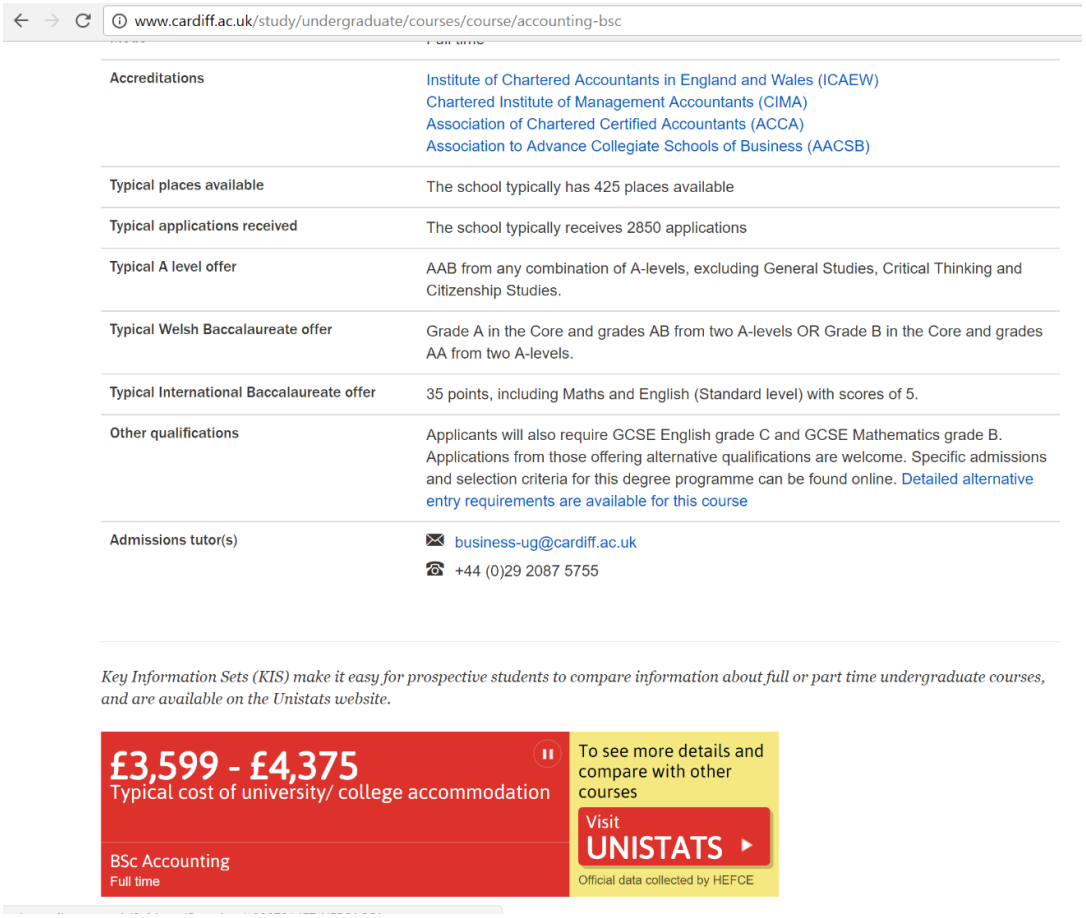


Figure 1: Screenshot from a Cardiff University Accounting BSc, featuring the Unistats widget (CardiffUniversity, 2016).

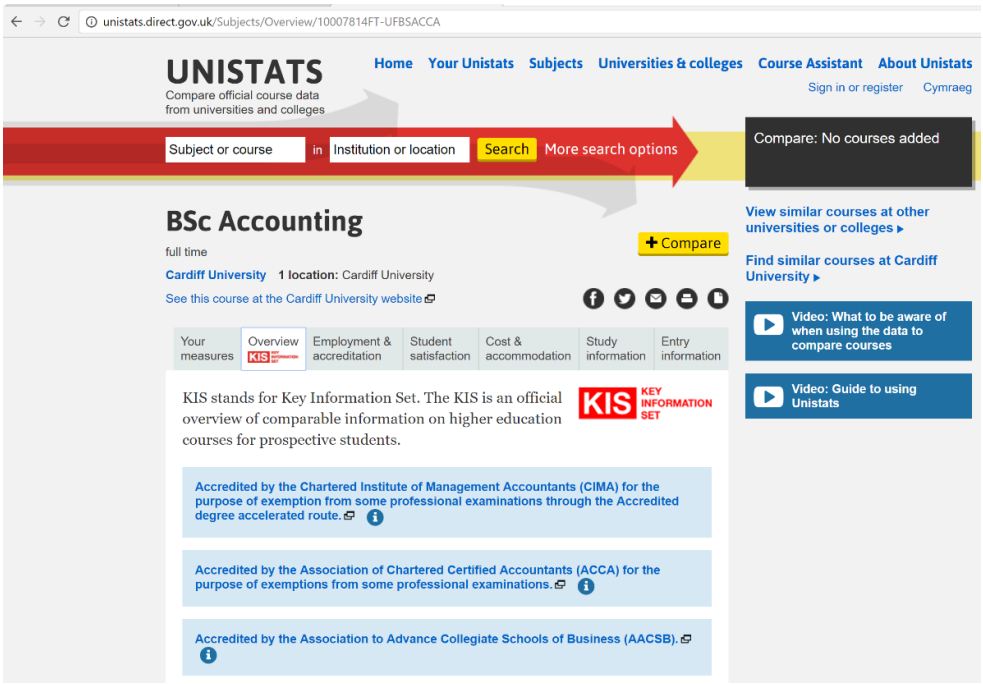


Figure 2: A screenshot from the equivalent Cardiff University Accounting BSc Unistats page (Unistats, 2016).

Resourcing KIS at HEFCE

During the current incarnation of Unistats, HEFCE has collated KIS returns from Further Education Colleges (FECs) in England and Wales. HEFCE runs a KIS submission system, where FECs upload their data with the aim of obtaining valid, signed off data before a deadline for the next academic year. Several members of staff at HEFCE are involved in the design and maintenance of the KIS sign off system, which although burdensome does ensure that all the data to be displayed on Unistats has been through a validation process. Collecting KIS specific data is not an entirely simple task for the institutions involved either, with the majority reporting at least some areas are challenging or burdensome to produce (DELNI et al., 2015b). Moving a portion of detailed information away from a central point, such as Unistats, could theoretically reduce the burden at HEFCE. However, if institutions are required to display certain information in a clear and comparable manner, then presumably a body acting on behalf of potential students would be required to check this was happening.

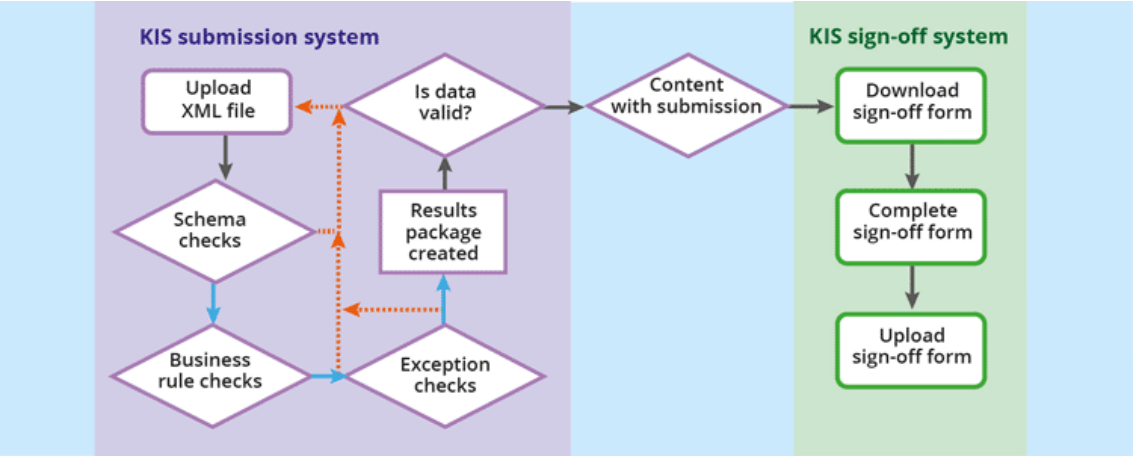


Figure 3: Diagram of HEFCE's KIS submission system, used by Further Education Colleges (HEFCE, 2016).

Assuring Compliance

It is clear the sector is moving away from a single central point for undergraduate HE information. During this transition and afterwards, a problem of how a regulator might ensure compliance arises. The main issues to solve in assuring compliance with information standards are:

1. There are currently 369 institutions on Unistats, even checking the compliance of course information from a single course at each institution is daunting.
2. Checking a single course is unlikely to satisfy a requirement of ensuring compliance.
3. Alternative providers (APs) will also soon be required to comply with information standards, further increasing the number of institutions who need assessment (BIS, 2015).
4. The data and web pages are changeable without notice, meaning any assessment of compliance could quickly become out of date.
5. At minimum, a basically trained eye would be required to make an assessment of compliance.
6. Splitting the assessment up in to more manageable pieces for individuals introduces problems of consistency; how could it be assured that each person assesses page content in the same way?

Competing aims of decentralising data display whilst efficiently assuring compliance provided the context for this project to become relevant. Primary difficulties in a human based approach to this task are size (number of pages), time constraint and classifying potential compliance. Usefully, the size and time constraints of visiting many web pages can be handled through web scraping; and since compliance is related to a textual document, we can use text classification algorithms to assess compliance.

Aim and Deliverables

The overall aim of this project was to produce a system that could reduce the human effort in inferring an institution's compliance with information display requirements. To achieve this, the

system would have to access course information pages, recognise them as such and then make an assessment on each page's compliance. This aim has been translated into two major deliverables:

1. A web scraping bot that could crawl a given domain and return text content from web pages.
2. A document classification model that could, at minimum, classify text content against two classes: "course information" and "not course information". Further classes relating to levels of compliance can be introduced as the model is refined.

Web Scraping Software for Data Collection

Guiding Principles

In order for the classifier to be developed, text data was required from institutional web sites. Given a set of base URLs, this task can be fully automated using Web scraping. Web Scraping is one term given to the process of automatically downloading HTML from the internet and parsing it for content (Brody, 2012). Modern websites tend to be created with common structure between their pages, when this structure is understood then particular data in a web page can be targeted. In this project all human readable content was of interest, so the HTML was parsed not to target particular objects but simply to remove HTML tags. Bots that carry out web scraping will create more work for themselves by collecting links to other pages from their current target, or by understanding the website's URL structure and altering it accordingly. Web scraping is an easily achievable task at the basic level, with languages like Python offering useful libraries such as lxml, requests and Scrapy (Reitz, 2016, Scrapy, 2016). However, the industrial setting of this project lead to a desire for a high level of control over the web scraping bot(s) and also the ability to easily substitute or tune components to better serve future data collections.

The entire code base of the project was influenced by the contents of two books in particular; primarily through reading all of the generic design pattern guidance from Dependency Injection in .NET (Seeman, 2012) and, secondarily, the opening chapters of Agile Modelling (Ambler, 2002). Software engineering and design patterns are topics worthy of entire projects in themselves; there's no implication here that the areas have been fully covered in preparing to write code for this project. Instead, a pragmatic approach was adopted, drawing on the books mentioned, experiences gained through the MSc course and example code bases/advice from those with greater experience at HEFCE. This approach favoured a simplistic design of connecting several discrete components, each responsible for its own task (separation of concerns (Makabee, 2012)), along with unrefined modelling in the early stages. Models were hand sketched, making reference to: possible structuring of components; seams that would join them; data flow through a large idealised application; and tasks different potential users should be able to achieve (see Appendix A). The models did not represent the final code but they did help to shape my thinking and structure my approach in development; which is an aim of agile modelling as described in Ambler's book. Seeman's explanations on Dependency Injection were very compelling and have changed how I structure programs to favour ease of adjustment, maintenance and unit testing. It wasn't within my current technical understanding to implement a full Dependency Injection framework. The currently favoured system at HEFCE is Castle Windsor (CastleProject, 2014), and I would hope to be able to use this in the future. Despite no official framework, I was able to gain many benefits by passing dependencies through a class' constructor and storing them for later use. Whenever a class would be accepting dependencies, I created an interface and specified that as the type to expect. Concrete implementations of functionality inherit the appropriate interface, ensuring they meet the "contract" of expected functions and return types. In this way, any class that correctly inherits the interface can be passed in to the waiting class as a dependency (Jensen, 2009).

When considering the data collection requirements of the project, I wanted and was encouraged by HEFCE towards limiting bot activity. Together it was decided to limit requests to individual domains

to around 500 in any 24 hour period and no more than one request every two seconds. Reasons for this revolved around not wanting to place unwelcome burden on institutional web servers; the numbers are arbitrary but should not tax even a small environment. It was also decided that robots.txt files would not be parsed before access and that bots would be identified with my organisation and name in the user agent string. A full ethical discussion on web scraping is beyond the scope of this project but the approach hasn't yielded any enquiries yet, aggravated or otherwise. At the time of writing, multiple tens of thousands of pages have been collected. I decided that the architecture behind each bot should allow for multiple to be run simultaneously without conflict. Conflict in this case would have meant bots visiting the same pages and making wasteful requests, or visiting the same domain and possibly breaking the rate limiting.

Main Components

From the guiding technical principles and ethical considerations, it became clear the bot itself could be a simple program, following a loop of accessing a page, parsing links and human readable content which were then stored in a database. The database would be a central point to coordinate the bots as it would have view of all the available information; removing the need for bots to communicate with each other or make decisions for themselves. A low access rate for each bot also meant that a traditional Relational Database Management System (RDBMS) would manage data fast enough to not become a major bottleneck. Relational databases are commonly used by HEFCE and in general for business applications. The need to keep relations between tables up to date can cause performance issues when dealing with many concurrent requests, making any API built on top of the database slow to respond. Through careful structuring of queries and experimenting with the frequency of requests made by bots, a stable database structure capable of running on a basic machine was achieved. Whilst individual bot performance was of little concern, the project did require the capability to collect the 500 daily pages from a variable number of domains; making the overall performance and scalability of the system a higher priority. For this reason, I chose to make the bots communicate with the database through a web API, sending data through HTTP requests. Using a web standard communication protocol means the bots and database can be hosted on one machine for a compact, small collection; then if a larger dataset is required, they can be easily moved to multiple machines communicating via a specialised web server and storing data in a fully tuned database server.

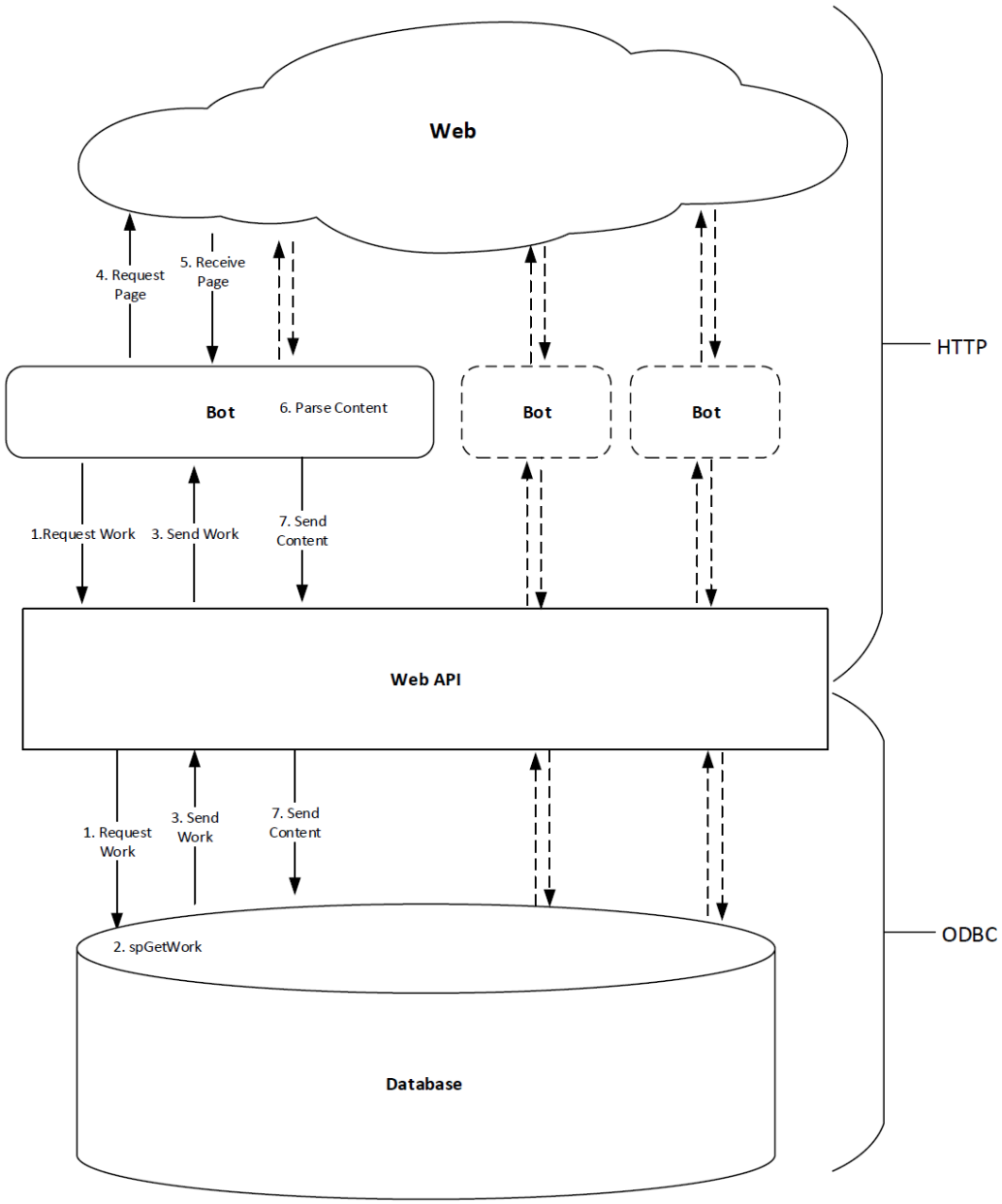


Figure 4: Diagram showing the interactions between the bot(s), web API and database.

Normally in the early stages of any coding task a name for the overarching project is required, especially when written in C# (as this project is) with its heavy use of namespaces. Code in this project was written under the namespace mScrape, as web scraping is a main task and an “m” prefix also allows the course’s award category of MSc to be included.

Web Scraping Bot

mScrape.webBot is the solution containing all the required interfaces and their concrete implementations for the web bot to request pages, parse their contents and communicate with the service API. There is purposefully an easy to follow thread of execution through the steps that the bot takes, this main logic is stored in bot.cs and executed via the Start function.

```
public void Start(){
    while (true){
        // get work from the API

        foreach (page in work){
            // access the page

            // parse human readable content

            // parse links to other pages
        }
        // send the links back to the API

        // send the human readable text back to the API
    }
}
```

Pseudocode 1: mScrape.webBot.bot.cs

Each of the main actions of the bot is handled by an appropriate implementation of an interface, all of these are passed in via the bot's constructor. The pseudocode above is very close to the functional code, demonstrating the clean separation between high level operation and technical implementation that is possible with the design pattern. Because these interfaces were designed to aid in separation of concerns, their public members are self-explanatory. The particular implementations used for most of the project made use of Selenium for page requests, HTML Agility Pack for parsing text and links from the raw HTML and RestSharp for communicating with the API (HtmlAgilityPack, 2012, RestSharp, 2016, SeleniumHQ, 2016). Selenium is worth a further mention as it allowed me to ensure the source code being downloaded was as close to what a human would be served as possible. Most often, Selenium is used for regression testing web applications because it allows the user to automate a number of different web browsers. Using a full browser ensured HTML that would be created on page load had completed before downloading the source. As performance of each bot was already throttled, the overhead of using a browser versus a direct request (or headless browser) was irrelevant.

Communication with the Database

Once data is passed back to the web API by the bots, it moves through further layers of abstraction that represent a data repository and data context. A data repository receives the data from the API and then takes some appropriate action using the data context. A repository was mainly used in this case for future development; the benefits of separating business logic from data access are very desirable, as is the ability to cache data centrally, but at this point the repository simply passes calls straight through to the data context (Microsoft, 2007). Keeping a data context separate from the API means that a new technology for data storage could be more easily substituted into the application; the project currently utilises SQL Server but HEFCE also heavily utilises SAS.

In the current configuration, different calls to the web API relate directly to individual SQL stored procedures; all of the SQL database objects vital for operation are stored in the mScrape.database solution. spGetWork is the procedure that encapsulates all the logic controlling what pages a bot requests, whilst making sure the c.500 page limit isn't broken and that each bot should be crawling a separate domain. Bots "should" be crawling separate domains, but the application could be

configured to mean this wasn't the case. Normal operation would mean that many more domains were being crawled than bots being used, so each call to spGetWork cycles through the list of domains and serves work back to the waiting bot; if only one domain was being searched and many bots were used then all would search that domain and the rate limiting of one page every two seconds would be easily broken. Edge cases such as this would be accounted for if the main focus of the project was an industrial grade web crawler; as the web bot code is a secondary but worthwhile side effect, user interfaces and edge case handling haven't yet been implemented.

When operating with data that has already been classified and stored in the "classified" table, spGetWork is able to prioritise work for bots according to where course information has been found in the past. This is achieved by calculating a rate for how many of the leaves at each node previously served have been classified as course information. For example, if bots have already visited ten pages attached to the parent node `someUniversity.ac.uk/courses` then the hit rate for that node is the number of pages classified as course information divided by ten. If all ten pages were course information then the hit rate is one and other pages beginning `someUniversity.ac.uk/courses` would be crawled as a priority. The number of course information pages found at a node serves to break ties where the hit rate is the same, finally if both of those measures are equal (which is possible when bots request small amounts of work) then a random selection is used. Prioritising work for the bots in this way adds both protection against false course information classification and works well with how many universities structure their web pages. Protection against false positives is offered because as the classifier assesses more pages from the node then the hit rate is likely to quickly drop below that of a node actually representing a root for some course information. With the prevalence of content management systems and a unified web site for each institution covering courses in multiple academic schools, many chose to structure their sites with a common root for course information (either all course information or sometimes at a more granular level like undergraduate versus postgraduate or school). Prioritisation based on previous outcomes therefore causes the bots to gravitate towards course information, which anecdotally tends to link to further course information as pages suggest similar courses in the school or subject. This is of course subject to the system having a classifier that is accurate enough, the tuning of classifiers for this purpose is discussed later.

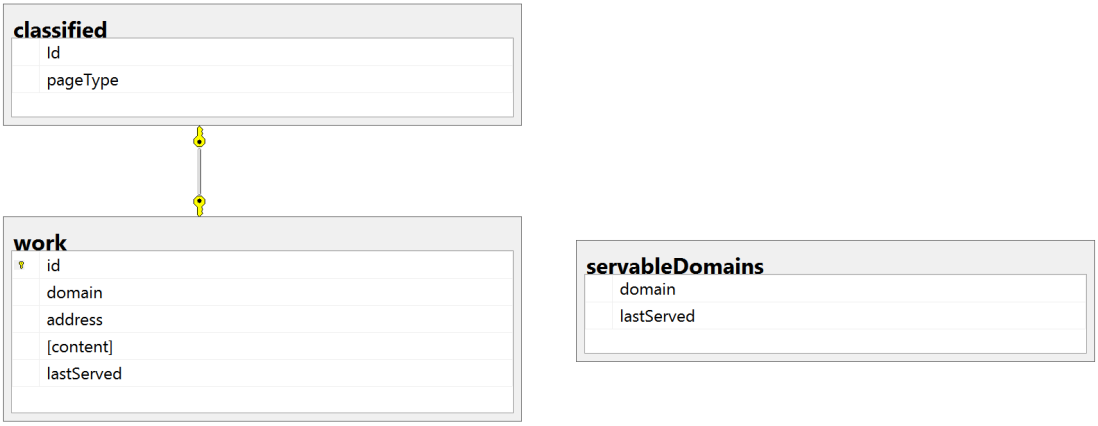


Figure 5: Entity relationship diagram of the underlying database.

Example Web Scrape

For the purpose of clarity it may help to have an example of how the bot would operate on a simple web page.

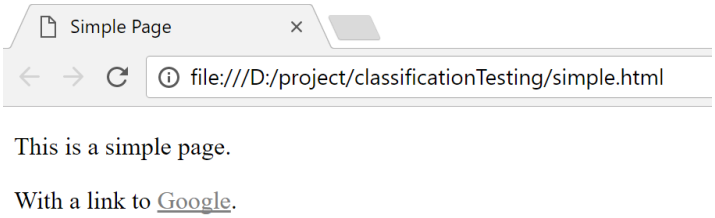


Figure 6: A simple web page rendered in Google Chrome.

The simple page above was hosted statically on my machine, so Chrome defaulted to the "file" protocol rather than HTTP(S), but the effect is the same. Like other web pages, this simple page is built using HTML, Hyper Text Mark-up Language.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Simple Page</title>
</head>
<body>
    <style> a {color: gray} </style>
    <p>This is a simple page.</p>
    <p>With a link to <a href="http://www.google.co.uk">Google</a>.</p>
    <script>console.log("hello, World!")</script>
</body>
</html>
```

Figure 7: HTML for a simple web page.

Despite being simple, this web page demonstrates the parsing issues that the bot deals with before passing the human readable content back to the database. Comparing the rendered page to the source HTML shows there's a lot more information contained in the HTML document then the text for a human to read. In this project, we're specifically interested in what a University provides for people to read though, so all the extra information will need to be removed. First, "style" and "script" tags, which are not rendered on the page are removed.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Simple Page</title>
</head>
<body>
    <p>This is a simple page.</p>
    <p>With a link to <a href="http://www.google.co.uk">Google</a>.</p>
</body>
```

All the human readable content (except the title) of a webpage is contained within children of the “body” tag, so everything not in the “body” tag can be ignored.

```
<body>
    <p>This is a simple page.</p>
    <p>With a link to <a href="http://www.google.co.uk">Google</a>.</p>
</body>
```

Next, all the HTML tags are removed.

```
    This is a simple page.
    With a link to Google.
```

Finally, white space is compressed and the result will be sent back to the database.

```
This is a simple page. With a link to Google.
```

The simple page is now represented in one string, which can be easily stored in a database and later recalled for classification. Actual webpages are obviously more complicated than the example here, but broadly the approach worked to strip non-human readable data from the source code. The approach did seem to not cope with the “iframe” tag very well though. An iFrame is a way nesting HTML documents that perhaps confused the way I had coded this stage of the bot. For an example of a real world web page and the output after the bot parsed it, refer to appendix B.

Text Classification

Algorithms for text classification have been well utilised and explored within machine learning, data mining and related areas (Aggarwhal and Zhai, 2012). A complete dissection and comparison of text classification techniques warrants a project in itself, therefore this section aims to provide a small grounding in the major terms and techniques used. Two excellent sources of information for this section were descriptions provided in Aggarwhal and Zhai's chapter of Mining Text Data, "A Survey of Text Classification Algorithms", and the contextual examples available in Witten et al.'s book "Data Mining, Practical Machine Learning Tools and Techniques". There are many technically complex texts available on this subject, but these two provide a detailed enough explanation to practically implement classifiers.

Classification Overview

The use of algorithms to classify data or objects is often introduced with an example in which species of the iris flower are classified based on sepal and petal measurements (Fisher, 1936). Classification in general is a problem associated with knowing some properties of an object and having a class assigned to the object based on them. Different situations of classification will lend themselves towards "soft" or "hard" versions, where the former implies finding the probability that an object is of a certain class and the latter makes a clear categorical assignment.

In many applications, rule based classifications are used in isolation and will be familiar to anyone with basic programming knowledge. The structure of "if (comparison is true) then… else..." is a fundamental concept in most programming languages. Such a rigid structure of black and white judgements is sufficient in situations where all the variables for comparison are known and accounted for. Traditional rigid data structures with a small number of properties and clear boundaries between classes lend themselves well to this kind of processing. However in a data-rich environment this approach quickly becomes cumbersome and impractical, as the number of rules and exceptions to rules multiplies rapidly (Guruswamy, 2015). Writing a large of amount of strict rule based statements often makes it difficult to assess what impact changes to code will have, leading to difficulties maintaining or updating key pieces of logic. Unfortunately updating the logic will also be a regular requirement if code deals with highly changeable real world data, especially when the data is semi- or unstructured.

In the context of our study, institutional websites may potentially hold more variety than can be processed by simple rules. Rather than manually creating classification rules, we can exploit readily available training data to implement a supervised machine learning approach instead. There is a large variety of such approaches that can be used. According to "no free lunch" theorem, there is not a single machine learning algorithm that will consistently outperform all other methods. In other words, the best performing algorithm depends on the type of problem at hand and the data available. This means that an algorithm appropriate for the given problem and data available, should be chosen after systematically evaluating a range of different algorithms. This can be achieved by cross-validation and comparison of the results achieved on the training data. Evaluation of classifiers is discussed following a brief introduction to some commonly used algorithms in supervised machine learning.

Supervised Machine Learning

Approaches to machine learning can be split into one of three categories: unsupervised, where algorithms assign classifications to data according to similarities, and with no previous knowledge of the domain; supervised machine learning (SML), where a corpus of pre-labelled data is fed into the algorithm for comparison with new data, before categories assigned; and semi-supervised learning, which attempts to overcome shortcomings of the previous two methods through combining them (McNulty, 2015, Aickelin, 2015).

Supervised learning mimics the way that skills are seemingly passed between people in formal education. With enough examples from the domain, showing differing properties and their effect on outcomes then a learner can start to predict future outcomes in the same domain. The main challenge with this approach is providing enough labelled examples to the algorithm to reliably train it. The dataset containing labelled examples from the domain is known as the training data. Training data must contain examples of all the classes the final model is expected to identify. This makes training data specific to the domain that a classifier will operate in. If the classifier will be differentiating types of news article, it must be trained using data containing all the types of news article it may reasonably encounter. Even when the number of classes to identify is small, if the domain features highly variable information then the training data will need to cover a wide area. This increases the number of training instances required, which will all need to be labelled manually and could cause a bottleneck.

In this project, I chose to explore the use of supervised learning algorithms. I found several institutional websites had a structure that allowed automatic labelling of pages. This is a feature particular to the domain and couldn't necessarily be relied upon in a future project, but here it offered the opportunity to collect a relatively large amount of labelled training data with very little human effort. The labelling was achieved through URLs from specifically chosen institutions, there is no guarantee that others would follow the same pattern; this labelling isn't a viable pattern for a classifier itself. As an aside, I did attempt to have the bots crawl sites given keywords in URLs, but tuning the bot for use on one site made it too specific to be of use on others.

Naïve Bayes

In any body of text, each word imparts different amounts of influence over how the larger body might be categorised by a human. This influence varies by the word, their context and location in a document. For example, a word in a title might give a heavy indication as to the category of some text. This thinking is intuitive to humans, but complex to represent in an algorithm. A simpler approach is to treat every word, regardless of context, as if it has equal effect on the outcome of the classification. Training data can be used to determine probabilities for each word being present in a document of a particular class. When a new document is passed through the classifier, then it is classified according to the probabilities of those words present in both the model and new document. This is the approach of Naïve Bayes; the "naïve" refers to the assumption that word order and context have no influence on the outcome (each word is an independent event), and "Bayes" refers to Bayes' rule. The naïve assumption is important, because without it then it is mathematically

incorrect to multiply the probabilities from each word. Bayes' rule is essentially multiplying the probabilities that each word is related to the class in question, which is a much simpler calculation than accounting for word order and/or location with the document.

Baye's rule states that with an hypothesis H and evidence towards that hypothesis E, the probability of H being true is given as:

$$\Pr[H \mid E] = \frac{\Pr[E \mid H]\Pr[H]}{\Pr[E]}$$

Figure 8: Baye's rule, from Witten et al.

The "Pr[|]" notation is used to say the probability of one thing, based on another. So Pr[H|E], is the probability of the hypothesis being true given the evidence. Each classification can be seen as a separate hypothesis, so Pr[H|E] will be calculated for the hypothesis that the document is course information, then non-course information, then Unistats information. Knowing the probability that each word in a group of documents holds for each class (Pr[E|H]) and the "prior probability" (Pr[H]) then Pr[E] can be ignored. The prior probability is the general probability that a hypothesis is true given the past examples, in this project Pr[H] for each hypothesis would be the proportion of each document class in the training set. Pr[E] can be ignored because it will be the same between classes, therefore values obtained purely from the top part of the fraction can be compared.

Witten et al. describe Naïve Bayes as having a reputation for being fast and quite accurate, which would be useful features in a real time application of the finished project, if it is also the case in this domain.

Nearest Neighbour

A nearest neighbour algorithm takes a set of labelled data and delays its work until classification time, when it will attempt to find the closest representation(s) of an unlabelled instance in its labelled set. Imagine a real world example using physical neighbours in a street or small community. A "nearest neighbour" in this sense is not simply the person living in the next house, but is a person whose wider attributes closely relate to yours. If all of person A's characteristics are known and identical to person B's, it is not unreasonable to predict an unknown characteristic of person B to be the same as person A's. This is a simplified nearest neighbour approach, but is an approximation of how the text classification could work. In training the classifier, the model receives a list of classifications that have attached frequencies or Boolean "present/not present" values for a list of words. When an unclassified document is passed to the classifier, then the model will attempt to locate the item(s) in its training data that is most similar. The known classes of the neighbour(s) will be used to classify the unlabelled document.

Nearest neighbour classification is known as "lazy" because of the delay in work, it has advantages in domains where there are huge amounts of data. It is reportedly used to great affect by large online retailers for suggesting products to customers based on people who represent their nearest neighbour |Abernethy, 2010, Data mining with WEKA`, Part 3: Nearest Neighbor and server-side

library~. This has advantages over other machine learning algorithms in that online retailers tend to have a vast amount of data, meaning pre-processing recommendations for customers based on all the available data is inefficient. Nearest Neighbour's "lazy" approach also means recommendations are only generated when required and using fewer points of information, assuming there is an effective search strategy for the nearest neighbours.

Random Tree

A tree algorithm constructs nodes at which decisions are made, outcomes at nodes will process an instance in the direction of a classification (Brownlee, 2016). A simple binary tree can be used to classify objects, the tree is "binary" because at each node a comparison is made that results in a "yes" or "no" answer. It would be trivial to construct a binary tree that grouped models of car, for instance, by having nodes checking for engine size, number of seats, number of doors etc. Random Tree classifiers are based on the traditional decision tree model, except in this case only a randomly chosen selection of attributes are available at each node. A logical extension of the Random Tree is a Random Forest. Random Forest classifiers combine multiple Random Tree classifiers and average the classification result across them.

As a tree method attempts to keep on splitting instances into their respective classes a tree can end up large, with single instances at the ending leaf nodes. The decision nodes immediately above these leaf nodes will be very specific to the instances below them. These nodes are now too specialised and less useful in a production environment. This is one way that Tree methods can become over fitted. A stopping criterion can be used to set a minimum number of instances to have at each leaf during building the tree, or pruning can be used afterwards to remove those branches that have become too specific (Brownlee, 2016).

Support Vector Machine

Joachims' 1998 conference paper on the use of Support Vector Machines (SVM) for classifying text data has been cited over 7.5k times according to Google Scholar; proving the topic of SVM use is popular within the machine learning domain (Joachims, 1998). Joachims lists their benefits for use in text classification based on strengths of: being able to deal with high dimensionality; and, success in domains where classes are linearly separable. In text classification, the number of words (features) chosen in the training and subsequent unseen data is also the number of dimensions in the problem. It is trivial to see that even medium sized documents may have many thousands of words within them. Approaches that do not scale well with increased dimensionality therefore rely on reducing the number of words used to make a classification. In this project the data was limited to a 1000 word vector for each instance. Joachims argues that in text classification there very few irrelevant words and therefore as many as possible should be included; making use of an SVM an obvious choice. Regarding types of document being linearly separable, this refers to classes being distinguishable by a linear "decision surface". In a two dimensional problem represented on a graph, this would be a straight line. Joachims gives the example that many of the Reuters news agency categories are linearly separable, I have not assessed the linear separability of this domain's classes however.

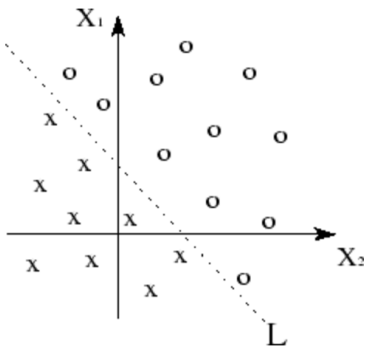


Figure 9: Linearly separable classes in two dimensions (Kawaguchi, 2000)

In the example above the classes "O" and "X" are linearly separable in two dimensions, because a straight (linear) line can be drawn between them. Picturing a similar graph for 1000 dimensions isn't required, the point Joachim's makes is that different document classes don't tend to mix up with each other. A document tends to either be of one type, or another.

Evaluating Classifiers

When a classifier has been trained using labelled data it is important to assess its accuracy against separate test data in order to have an estimate of how well it will perform on unseen data once the system is deployed in practice. The test data should not have taken any part in building the classifier under test. In this project that would mean using entirely different institutional web pages for training and test data. Accuracy in relation to the training data will likely be an overestimation of the classifier's performance, because the classifier has been built with this specific training data in mind. Consider a classifier that is 100% accurate at making categorical assignments against its training data, because it already has full knowledge of every instance's class. Such a classifier could comprise 100 "if" statements for 100 instances to be classified, assigning a class because each instance's ID has been hardcoded into the corresponding "if" statement ("if (ID = 1) then class =…"). This classifier would have seemingly perfect performance against the training data by a simple measure of accuracy. It might seem reasonable to declare this classifier as 100% accurate, but it is clear that this classifier will fail dramatically when fed new data. Even if the new data contains an ID property to be assessed, any correct classification will be based on the chance that the new data contains instances with the same ID and class as the old data. This is an extreme example, but the classifier is suffering from "overfitting"; it has become too specific to one dataset to be useful in a general context. A difference in accuracy against training data compared to test data can indicate to what degree the classifier suffers from overfitting. A smaller difference could indicate greater stability in the classifier, but of course without necessary accuracy then the classifier would still be inappropriate. As this project deals with web pages, there is a very wide scope for the content of the data that will be encountered. Because of this, a classifier with a small reduction in accuracy from training to test could be desirable, likely more so than another that has a higher test accuracy but larger difference to its training accuracy (assuming both have baseline accuracies that are high enough).

Keeping in mind what's previously been said about training versus test data, there is a method known as cross validation that is available to help assess performance against training data more accurately. Cross validation is useful in situations when labelled data is sparse and/or there is a

reasonable degree of randomness in the classifying algorithm (breaking ties, random start points etc.). In cross validation the term "fold" refers to the number of times a classifier will be constructed and tested, the accuracy of each fold is averaged for a final result. For each fold, a proportion of the training data is excluded from creation of the classifier, and then used in place of "true" test data to assess accuracy. For example, a 10 fold cross validation using a dataset of 100 instances: in fold one instances 1-10 are excluded during training and then used as labelled test data; in fold two instances 11-20 are excluded whilst a new classifier is created, then again the accuracy is assessed with the 10 excluded instances. The method continues in this way until all 10 folds have been used to make a classifier and tested. A final accuracy figure obtained through cross validation can still suffer from the issues discussed earlier, but extremes of performance should have been evened out by the averaging process.

So far, we've only discussed classifiers in terms of one measure, which is accuracy; the overall percentage of correct classifications that it makes. At this point a distinction should be made between the four types of result that a classifier can achieve for each instance. The four categories of result are combinations of true/false and positive/negative predictions. In a situation where a classifier was attempting to assign a class labelled by letter, the following table shows the possible results with respect to the class A; this kind of table is known as a confusion matrix.

		Predicted Class	
		A	Not A
Actual Class	A	**True Positive**	False Negative
	Not A	False Positive	**True Negative**

Table 1: Four possible results for assigning a class.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Neagatives + False\ Positives + False\ Negatives}$$

When talking about accuracy both of the "true" outcomes count, their proportion of the total classifications is the accuracy itself. In this project, there's no particular good or bad outcome associated with an incorrect classification of one webpage; at least, not with the current compliance guidelines. Other situations such as clinical diagnoses carry with them more obvious negative consequences for certain types of result. A false positive from a classifier predicting a debilitating disease would warrant further more costly tests, a false negative could mean the person not receiving vital treatment; both of these false results have negative consequences but the latter is clearly worse. In these situations a user can add weight to the predictions made by a classifier. In a clinical example it would have to be much more certain a person was negative for a disease, before classifying them as such.

"Precision" and "recall" are terms used in both information retrieval and classification. In a classification scenario precision is the ratio of true positives to the sum of both true and false positives (Descoins, 2015). Precision is a measure of a classifier's "exactness" at correctly identifying

a class, what proportion of all the instances labelled as “A” were actually “A”? Recall is the ratio of true positives to the sum of true positives and false negatives, what proportion of all the actual “A” class instances did the classifier identify? Using both precision and recall allows us to avoid falling into some particular pitfalls associated with using accuracy alone. Consider a situation where a classifier only ever labelled instances as “A”, regardless of their properties. In this situation all accuracy tells us is the proportion of the dataset that were “A”. Inspecting only recall in this situation could also be misleading, as it will be 1.0 because of course every instance of “A” was classified correctly. Bring in precision though and we’d see a much lower ratio. Similarly a classifier could be created with better precision by only classifying instances as “A” when it was absolutely certain, but now the recall would falter as many real instances of “A” would be missed. Considering the example above regarding clinical diagnoses, it may be desired to favour recall over precision in certain situations. If people labelled “A” are those that will not suffer from a disease, then higher precision will ensure we’re more certain about the people in that group and who will therefore receive no further screening. It would be expected that some people on the borders of classification for “A” would not be labelled as “A”. They would suffer the inconvenience of further tests but in doing so reduce the chance that any people truly “not A”, and therefore may have the disease, were overlooked. A classifier that made no mistakes would have precision and recall of 1.0, but this is unlikely in any real-world application; in reality there is a balance to be made between recall and precision. One way of assessing the balance between recall and precision is using the F-measure.

$$Precision = \frac{True\ Postives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

This version of F-measure is also known as F_1 and equally weights precision and recall in the calculation. Optimising for F-measure can help create a classifier balanced in both recall and precision, which may or may not be desired in the final application.

First Stage Classifier Implementation

Guiding Principles

I investigated a data mining software package called Weka; available from the University of Waikato in New Zealand (Hall et al., 2009, UniversityOfWaikato, 2009). Weka is a popular open source framework, with implementations of machine learning algorithms that can be used in Java code by programmers; Hall et al. (2009) has been cited over 12k times according to Google Scholar. Weka is not only a framework usable in code, but comes with a fully-fledged GUI program for investigating data and applying machine learning algorithms. Some of those involved with writing the software have also released an accompanying book, exploring data mining from a practical rather than technical or mathematical perspective (Witten et al., 2011). Witten et al.'s book provided a grounding in the knowledge required to build a classifier. I had previously read the opening chapters of a more technical text but found the explanations too abstract (Alpadym, 2014).

One potentially large issue with using Weka's libraries for the classifier was it being coded in Java. As HEFCE generally use .NET technologies I wanted to interact with the library through C#. If this wasn't possible then a classifier in Java could easily exchange data with the system, thanks to its design featuring a web API. Fortunately Weka documentation pointed towards using some third party software called IKVM (Frijters, 2014); using examples and guidance from this documentation I was able to port the Weka.jar file in to a .dll for use with C# (WEKA, 2009a, WEKA, 2009b).

Having no previous experience in machine learning algorithms, I collected a dataset for building models and testing them, then created several classifiers to try and gauge their baseline effectiveness in this application. The following sections describe the data and models used to explore the possibilities of classifying course information.

Training and Testing Data

Using the web bot, data was collected from a range of institutional websites, Unistats and non-sector websites. Data from non-sector websites was included to help not over fit the classifier to university web pages. Inclusion of non-sector data wasn't strictly required as the bot would be limited to specific domains in the future, but this wasn't certain during early stages of classifier development. Unistats data was included on the possibility that an institution may have used words similar enough to trigger a classification, although it was expected that no universities would have done this judging by previous sector consultations.

The university websites selected for data collection were chosen because manual inspection of their sites revealed a common node for course information; allowing for mass classification of pages based on their URLs. Gaining enough labelled data to build a classifier is a common issue in data mining, as discussed by Witten et al. A clustering based approach to labelling training data is explored later. URL based labelling worked for this training data because they were manually inspected; the web bot did, for a short period, attempt to access course information based on words in the URL (like course or undergraduate) but the results were very unpredictable and too specific to how a website was structured. The bot was tuned to help find course information on individual

websites, resulting in a mix of course and non-course information from these universities (ncl.ac.uk, dur.ac.uk and exeter.ac.uk). For others, it was specifically guided away from course information or left to randomly collect pages (shef.ac.uk, ucl.ac.uk). It is expected that some non-course information pages would have been incorrectly labelled in this approach, but the sheer number of correct classifications should drown out this noise.

Data for testing classifiers was collected from four universities and also mass labelled using URLs as either course information or not course information. The test data didn't contain pages from the Unistats site as a production version of the classifier would not be used against it. Including Unistats pages in the training data did mean the potential for a page to be classified in that category. When this did occur these pages were inspected manually to see if the classification was accurate, but it was not expected to be so.

Domain	Total Pages	Not Course Info	Course Info	Unistats
www.ncl.ac.uk	988	395	593	0
www.dur.ac.uk	900	315	585	0
www.exeter.ac.uk	798	454	344	0
www.bristol.ac.uk	763	463	300	0
www.cardiff.ac.uk	720	426	294	0
www.lboro.ac.uk	671	419	252	0
www.plymouth.ac.uk	541	433	108	0
www.liverpool.ac.uk	524	296	228	0
unistats.direct.gov.uk	501	171	0	330
www.shef.ac.uk	499	482	17	0
www.bbc.co.uk	494	494	0	0
www.tri247.com	493	493	0	0
www.ucl.ac.uk	464	458	6	0
www.theguardian.com	425	425	0	0
www.huffingtonpost.co.uk	414	414	0	0
www.hefce.ac.uk	247	247	0	0
wonkhe.com	99	99	0	0
www.educationuk.org	52	52	0	0
courses.leeds.ac.uk	5	0	5	0
www.nottingham.ac.uk	2	0	2	0
Total	**9600**	**6536**	**2734**	**330**

Table 2: Domains used in training data and number of pages as classified through URLs.

Domain	Total Pages	Not Course Info	Course Info	Unistats
www.birmingham.ac.uk	874	390	484	0
www.surrey.ac.uk	655	440	215	0
www.brookes.ac.uk	434	306	128	0
www.canterbury.ac.uk	92	69	23	0
Total	**2055**	**1205**	**850**	**0**

Table 3: Domains used for test data.

Domain	Path	Page Type
courses.leeds.ac.uk	http://courses.leeds.ac.uk	courseInfo
courses.uwe.ac.uk	http://courses.uwe.ac.uk	courseInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/Institutions	notCourseInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/Search/SubjectCode	notCourseInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/subjects/cost	unistatsInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/subjects/employment	unistatsInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/subjects/entry	unistatsInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/Subjects/Overview	unistatsInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/subjects/satisfaction	unistatsInfo
unistats.direct.gov.uk	https://unistats.direct.gov.uk/subjects/study	unistatsInfo
wonkhe.com	http://wonkhe.com/blogs	notCourseInfo
www.bbc.co.uk	http://www.bbc.co.uk/news	notCourseInfo
www.birmingham.ac.uk	http://www.birmingham.ac.uk/undergraduate/courses	courseInfo
www.bristol.ac.uk	http://www.bristol.ac.uk/study/undergraduate/2017	courseInfo
www.bristol.ac.uk	http://www.bristol.ac.uk/study/undergraduate/search/	notCourseInfo
www.brookes.ac.uk	https://www.brookes.ac.uk/courses/undergraduate	courseInfo
www.canterbury.ac.uk	https://www.canterbury.ac.uk/study-here/courses/undergraduate	courseInfo
www.cardiff.ac.uk	http://www.cardiff.ac.uk/study/undergraduate/courses	courseInfo
www.dur.ac.uk	https://www.dur.ac.uk/courses	courseInfo
www.educationuk.org	http://www.educationuk.org/global/	notCourseInfo
www.exeter.ac.uk	http://www.exeter.ac.uk/undergraduate/courses-by-subject/	notCourseInfo
www.exeter.ac.uk	http://www.exeter.ac.uk/undergraduate/degrees/	courseInfo
www.hefce.ac.uk	http://www.hefce.ac.uk	notCourseInfo
www.huffingtonpost.co.uk	http://www.huffingtonpost.co.uk/	notCourseInfo
www.lboro.ac.uk	http://www.lboro.ac.uk/study/undergraduate/courses	courseInfo
www.liverpool.ac.uk	https://www.liverpool.ac.uk/study/undergraduate/courses	courseInfo
www.ncl.ac.uk	http://www.ncl.ac.uk/undergraduate/degrees	courseInfo
www.nottingham.ac.uk	http://www.nottingham.ac.uk/ugstudy/courses	courseInfo
www.nottingham.ac.uk	https://www.nottingham.ac.uk/ugstudy/courses	courseInfo
www.plymouth.ac.uk	https://www.plymouth.ac.uk/courses/undergraduate	courseInfo
www.plymouth.ac.uk	https://www.plymouth.ac.uk/courses?course_index=	notCourseInfo
www.shef.ac.uk	https://www.shef.ac.uk/prospectus	courseInfo
www.surrey.ac.uk	http://www.surrey.ac.uk/undergraduate	courseInfo
www.theguardian.com	https://www.theguardian.com/education/	notCourseInfo
www.tri247.com	http://www.tri247.com	notCourseInfo
www.ucl.ac.uk	http://www.ucl.ac.uk/prospective-students/undergraduate/degrees/	courseInfo

Table 4: URLs used for mass labelling, note inclusion of sector pages not from universities like wonkhe.com and hefce.ac.uk.

Native C# Classifiers

Since there is significant overhead in developing a trained classifier it's worth first checking that usable results can't be generated via simpler means; a "simplicity first" approach is also advocated by Witten et al.. This also has the effect of offering some baseline performance with which to compare more complex classification algorithms. All the native classifiers were built on the training data, repeated ten times when there was a random element to the classification. Where a classifier was in essence "trained" (as opposed to purely random) then it was also assessed against the test data. The native C# classifiers use string functions designed to operate on an unbroken string, so they consume the sing string representing each website.

Random classifier

The most basic of classifiers that could be used in this situation is a random classifier that takes no account of the text and simply assigns a category to pages with equal probability. Results from this classifier represent an absolute baseline minimum for any future classifier.

Total	Average Success	Min	Max
9600	**30.6%**	24.9%	39.1%

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	2026	2191	2320
	courseInfo	985	780	968
	unistatsInfo	6	154	170

	Recall	Precision	F
notCourseInfo	0.310	0.671	0.421
courseInfo	0.285	0.248	0.263
unistatsInfo	0.516	0.050	0.092

Table 5: Averaged results from a random classifier against training data.

As could be expected, over ten runs choosing one of three categories (notCourseInfo, courseInfo, unistatsInfo) the classifier achieved a success rate approaching one third.

Random Proportional classifier

One advancement on a random classifier would be to classify pages based on their likelihood of existing in the dataset. For this classifier proportions of the three categories were found in the training data and then used to assign a category based on a random integer falling within an equivalent range. A 1000 integer range was used, giving an accuracy 0.1% when representing the proportions of page types in the training data.

Average Correct	Total	Average Success	Min	Max
5266	9600	**54.9%**	49.2%	59.1%

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	4474	1829	233
	courseInfo	1837	792	105
	unistatsInfo	210	120	0

	Recall	Precision	F
notCourseInfo	0.685	0.686	0.685
courseInfo	0.290	0.289	0.289
unistatsInfo	0.000	0.000	0.000

Table 6: Averaged results from a random proportional classifier against training data.

Results from this classifier reach over 50% accuracy, although it would be expected to reduce closer to 50% in more runs against this data. This is because the classifier has been trained against the data, so the ranges in which the random integer may fall (to assign a category) very closely represent the actual distributions in the data. This approach should eventually yield a 50% success rate on this data because the decision on success has now been reduced to a 50/50 choice, was the integer in the correct range or not? In the previous random classifier 33% would be the eventual end point as the classifier was choosing one of three categories with no preference based on knowledge of the data.

Whilst a 50% success rate may seem a useful starting point, this classifier would suffer from overfitting when assessing other data. This classifier could be fed 100 known course information pages and it would still classify some of them as not course information, regardless of their content. Future success of this classifier relies on the training data having proportions of page types that closely represent the proportions of page types in unseen data. In ten runs against the test data the classifier had much increased variance in its accuracy, but still did approach 50% success on average. However, the variance of this classifier means that average success rate is an inaccurate representation of its real world performance. Against unlabelled data a user couldn't know where in the wide possible range of performance the classifier is sat.

Average Correct	Total	Average Success	Min	Max
988	2055	**48.1%**	19.9%	81.0%

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	781	299	122
	courseInfo	646	207	0

	Recall	Precision	F
notCourseInfo	0.650	0.547	0.594
courseInfo	0.243	0.409	0.305

Table 7: Averaged classification data for a random proportional classifier, against test data.

Any Keyword

As content from each page is available to a native classifier as well as for training through Weka, it can be used to help classify pages. Using simple string functions in C# the appearance of keywords was used as an indicator of page type; these key words were selected manually. Separate arrays of words were used to classify course information and the more detailed Unistats information. If any word from the array was present in the page content then the page was classified accordingly, with Unistats trumping course information when both classifications applied. The following are results from using some different combinations of keywords against the training data. These results were from a single run, as there was no random element to the classification.

```
private string[] courseKeyWords = { "course" };
private string[] unistatsKeyWords = { "satisfied" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	2663	3706	167
	courseInfo	24	2448	262
	unistatsInfo	0	91	239

	Recall	Precision	F
notCourseInfo	0.407	0.991	0.577
courseInfo	0.895	0.392	0.545
unistatsInfo	0.724	0.358	0.479

Table 8a

```
private string[] courseKeyWords = { "course", "study" };
private string[] unistatsKeyWords =
      { "satisfied", "average salary" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	1704	4660	172
	courseInfo	13	2443	278
	unistatsInfo	0	74	256

	Recall	Precision	F
notCourseInfo	0.261	0.992	0.413
courseInfo	0.894	0.340	0.493
unistatsInfo	0.776	0.363	0.494

Table 8b

```
private string[] courseKeyWords = { "study" };
private string[] unistatsKeyWords = { "satisfied" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	2485	3884	167
	courseInfo	29	2443	262
	unistatsInfo	6	85	239

	Recall	Precision	F
notCourseInfo	0.380	0.986	0.549
courseInfo	0.894	0.381	0.534
unistatsInfo	0.724	0.358	0.479

Table 8c

```
private string[] courseKeyWords = { "study" };
private string[] unistatsKeyWords = { "average salary" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	2504	4007	25
	courseInfo	29	2677	28
	unistatsInfo	6	87	237

	Recall	Precision	F
notCourseInfo	0.383	0.986	0.552
courseInfo	0.979	0.395	0.563
unistatsInfo	0.718	0.817	0.765

Table 8d

Using an Any Keyword approach generally classified course information well, at least 0.89 recall in all cases, with one reaching 0.97. Precision for non-course information was also notable, but low recall reduced its F-measure. Reasonable success was also had identifying Unistats information, with each model scoring over 0.7 for recall. The last model scored well for both recall and precision on Unistats information, which is reflected in its F-measure. The difference in the precision from using “average salary” to identify Unistats information versus “satisfied”, shows how key the selection of words for use in the classifier is. Recall was broadly the same between these last two models, but precision was 0.358 in the first compared with 0.817 in the second.

The Any Keyword approach does suffer from not being selective enough, scoring low for recall in identifying pages that were not course information. The classifier’s preference was to classifying lots of pages as course information when they were not. Recall for non-course information dropped lower still when two words were included in the course information array (c.0.4 to 0.26). The criteria for classification as course information were being too easily met. It’s also likely that much of the success the classifier did have was due to the wide range of non-course information included in the

training data. Pages from outside the higher education domain would be much less likely to contain words like “graduate” and “study”. In a production setting the model would only be used against pages from within higher education, so non-course information pages from these sites would be more difficult to pick out.

The keywords from table 8d were used to classify the test data, as these produce the best F scores against the training data.

```
private string[] courseKeyWords = { "study" };
private string[] unistatsKeyWords = { "average salary" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	111	1094	0
	courseInfo	19	824	7

	Recall	Precision	F
notCourseInfo	0.092	0.854	0.166
courseInfo	0.969	0.430	0.595

Table 9: Any Keyword classifier run against the test data

Expectations that the recall for non-course information was related to the contents of the training data were correct. The test data only contains pages from universities, so whilst recall for classifying course information was extremely high (0.969) non course information was very low at 0.092. The classifier was assessing the vast majority of pages as course information. The percentage of course information in the data is 41.3% and the classifier correctly identified 45.5% of instances, which is an accuracy very close to that of mass labelling the data as course information. This classifier was not sensitive enough to properly distinguish between the closely related pages of the test data.

Seven instances featured the string “average salary” and so were classified as Unistats information, three pages from one institution and one from another. The group of three were classified more than once, owing to a style of website not accounted for in the database design. Different URLs were used to represent sections of the page that should be open to the user, so the source code was the same between them. The sentences that triggered the classifications were a very small part of the pages and I’m not sure if a machine learning classifier alone would be sensitive enough to pick the same up. Calling these pages Unistats information is likely a stretch, but it does disprove my earlier suspicion that no such information would be found. Although given the test data contains 2055 instances these seven instances represent only 0.3% of the data.

All Keyword

A simple modification to the Any Keyword classifier yielded an All Keyword classifier, where every word in a keyword array would be required to gain classification. Again, a small range of keywords was used to build the classifier against the training data, with the most successful at this stage also being used with the test data.

```
private string[] courseKeyWords = { "course", "study" };
private string[] unistatsKeyWords =
      { "satisfied", "average salary" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	3467	3049	20
	courseInfo	40	2682	12
	unistatsInfo	6	104	220

	Recall	Precision	F
notCourseInfo	0.530	0.987	0.690
courseInfo	0.981	0.460	0.626
unistatsInfo	0.667	0.873	0.756

Table 10a

```
private string[] courseKeyWords = { "course", "study", "graduate" };
private string[] unistatsKeyWords = { "satisfied" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	3898	2471	167
	courseInfo	45	2427	262
	unistatsInfo	90	1	239

	Recall	Precision	F
notCourseInfo	0.596	0.967	0.738
courseInfo	0.888	0.495	0.636
unistatsInfo	0.724	0.358	0.479

Table 10b

```
private string[] courseKeyWords = { "course", "study" };
private string[] unistatsKeyWords = { "satisfied" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	3444	2925	167
	courseInfo	40	2432	262
	unistatsInfo	6	85	239

	Recall	Precision	F
notCourseInfo	0.527	0.987	0.687
courseInfo	0.890	0.447	0.595
unistatsInfo	0.724	0.358	0.479

Table 10c

By definition, the All Keyword classifier becomes more selective as words are added to the keyword arrays; this allowed it to exclude non-course information better than the Any Keyword classifier. Common to all of the All Keyword classifiers was high recall and low precision for course information and high precision with low recall for non-course information; showing that whilst this classified improved on the Any Keyword approach, it still allowed a lot of non-course information to be classified as course information. The keywords from table 10a produced the most consistent F scores, so these words were used to test the classifier with the test data.

```
private string[] courseKeyWords = { "course", "study" };
private string[] unistatsKeyWords =
      { "satisfied", "average salary" };
```

		Classified		
		notCourseInfo	courseInfo	unistatsInfo
Actual	notCourseInfo	373	832	0
	courseInfo	20	824	6

	Recall	Precision	F
notCourseInfo	0.310	0.949	0.467
courseInfo	0.969	0.498	0.658

Table 11: Results from the All Keyword classifier using the test data.

Despite more success with non-course information versus Any Keyword, All Keyword's recall dropped 0.22 against the test data. As was the case with Any Keyword, this is likely due to the more challenging make-up of the non-course information in the test data. Six course information pages were classified as Unistats information, these were the same repeated three pages from the Any Keyword classifier.

Supervised Machine Learning Classifiers

The GUI provided with the Weka framework allowed testing of different machine learning algorithms. Guided by Witten et al.'s book several algorithms were created using the same training data as the native C# classifiers, these were then also assessed against the test data. All of the models were built and validated using 10 fold cross validation.

Results for the Weka built models feature three fewer instances than the native C# classifiers, an error in the way Weka connected to the database storing the training data meant that not all instances were returned. Weka not only generates an accuracy rate for correctly classified instances, but also a confusion matrix and statistics like recall and precision.

Problem Representation

In this project the data is stored in a central database for each web page, the content of each page is one single text string. A separate executable to that of the bot described earlier requests an amount unclassified work from the database. The database sends this work to the classifier executable through the web API. Again, this means that the classifier can be hosted on another machine specialised to the task if required. The work is an array of objects representing the previously scraped web pages. Each object has a "content" property, which is a single text string containing the human readable text from the web page. The process by which this text string is arrived at, is described earlier in the "Example Web Scrape" section and a real world example is given in Appendix B.

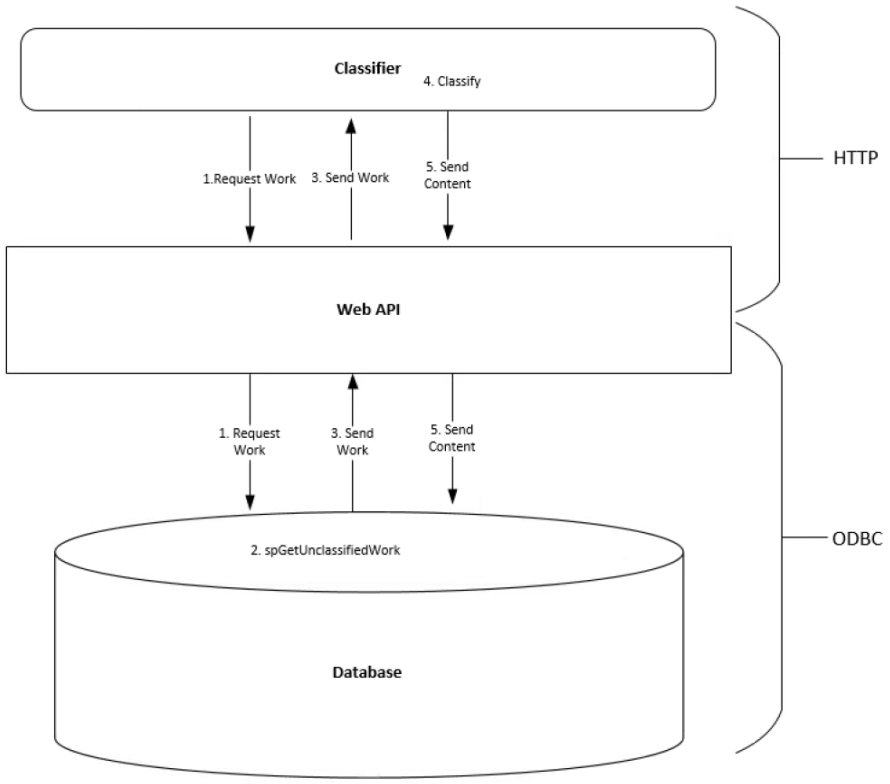


Figure 10: Data flow between the classifier and database.

A single text string cannot be used in a SML method and expected to produce a usable result. This is because the SML method does not read text, it expects groups of numerical values that can be operated on. In order to train a classifier, we must first represent the data (or problem) in a way that it can understand. The Weka framework uses a concept of vectorization, which can turn the content of a text document into a collection of numbers. Producing one vector (collection of numbers) for each document in a dataset is seen as passing the documents through a filter. There are many filters in the Weka framework, the StringToWordVector filter takes input of one or more documents (each in one string) and produces a collection of vectors that represent all of them. All of the vectors from a group of documents passed simultaneously through the filter will have the same word elements, even if not every document features every word.

Document ID	Document Class	dog	cat	cow	pig	…
1	A	1	0	1	0	…
2	B	1	1	1	1	…
…	…	…	…	…	…	…
n	A	0	0	1	0	…

Table 12: Example representation of document vectors, one and zero represent presence or absence of the word in the document.

Conceptually a group of vectors is similar to a table of information, where the left hand column is a unique ID for each document and subsequent columns each represent a word. If the document contains the word, then a non-zero numerical value will exist in the cell where the document's row and the word's column intersect. One final column is used to represent the documents class, in labelled data this has a value and in unlabelled data it is empty. The classifier's purpose is to compare the vectors it saw in training with those in unlabelled data and fill in the empty class column. Choosing a large enough vector size (number of words to keep) to allow all the documents in a dataset to be represented is key. In this project, 1000 words were used to make the vectors as this would give good coverage for the size of document without making the vectors too large to process easily. The 1000 words making up a vector were those that were the most popular across the document(s) going through the filter at that time. What numerical value is used in the vectors is selected whilst tuning the filter.

When working with documents of varying lengths it's clear that simple frequencies of words might not represent a documents class well. A very long document could mention a the word "dog" ten times but not actually be about dogs, whereas a short poem specifically about dogs could use the word only two or three times. A similar issue arises when considering words that are mentioned in very few of the documents making up a collection. In this project mentioning the word KIS would likely indicate some information relating to Unistats or display of the appropriate information. Applying the correct weight to words in both of these situations is addressed using a technique known as TF-IDF. TF stands for Term Frequency and IDF is Inverse Document Frequency, they deal with the issues in the order presented. In the long text featuring "dog" ten times, the TF for "dog" will be lower than the short poem featuring the word two or three times. TF is a simple measure of how frequently each word appears in a document. If only a handful of pages feature "KIS" in all our

training data then “KIS” then the IDF for “KIS” will be high. IDF is a measure of how many of the documents in a collection contain a particular word. In this project the TF-IDF was used as the measure for words in a vector. The TF-IDF is the TF and IDF for a word multiplied together.

$$TF = \frac{Number\ of\ times\ a\ word\ appears\ in\ the\ document}{Total\ number\ of\ words\ in\ the\ document}$$

$$IDF = \ln\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ the\ word\ in}\right)$$

$$TFIDF = TF \cdot IDF$$

on: QueryResult-weka.filters.unsupervised.attribute.StringToWordVector-R1-W1000-prune-rate-1.0-T-I-N1-L-stemmerweka.core.stemmers.NullStemmer-stopwords-handlerweka.core.stopwords.

266: eligible Numeric	267: email Numeric	268: employability Numeric	269: employers Numeric	270: employment Numeric	271: end Numeric	272: energy Numeric	273: engage Numeric	274: engagement Numeric	275: engineering Numeric	276: england Numeric	277: english Numeric	278: enough Numeric
0.0	0.552273	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.209402	0.0	0.0	0.0
0.0	0.552663	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.552743	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.553519	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.842003	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.211465	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.554374	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2132	0.0	0.843303	0.0
0.0	0.554369	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.843295	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.213396	0.0	0.0	0.0
0.0	0.554481	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.213435	0.0	0.843466	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.554823	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.214184	0.0	0.843987	0.0
0.0	0.555038	0.0	0.0	1.213586	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.214771	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.215157	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.449244	0.0	0.0	0.0	0.0	0.0	0.0	1.563818	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.845531	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.555928	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.216602	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.216666	0.0	0.0	0.0
0.0	0.556084	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.564382	0.0	0.0	0.0	0.0
0.0	0.0	1.115958	1.453121	1.217806	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.5571	0.0	0.0	0.0	0.0	0.0	0.0	1.56527	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.219504	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.848076	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.848076	0.0
0.0	0.557528	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.557579	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.220214	0.0	0.0	0.0
0.0	0.557586	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.84819	0.0
0.0	0.557661	1.117345	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.220442	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.557805	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.558247	0.0	0.0	0.0	0.0	0.0	0.0	1.568492	0.0	1.5716	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.558387	1.118799	1.456821	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.717897	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.558855	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 13: Excerpt from the group of vectors produced by the training data.

In table 13 you can see an example of the sort of data produced by vectorising actual web pages. The excerpt is showing the data for several web pages between elements 266 “eligible” and 278 “enough”. The numbers are the IF-TDF calculated for the word in the context of that document. The lower values of “email” suggest it is mentioned infrequently on individual pages but is featured on lots of them (this small excerpt shows around 20 pages featuring “email”). “Engagement” has around three times the score of “email”, suggesting it is mentioned in fewer of the pages and perhaps more often in individual pages when it is used.

The StringToWordVector filter also allows the user to specify treatment of stop words and how to tokenize words. Stop words are common words that don't help convey the message of text, like "and", "the" or "as". Stop words also tend to be very common, so easily end up consuming space in a document vector as well as not adding information relating to the document's class. In this project, data did not have stop words removed, it may have been investigated if models were not working. A tokenizer's job is to split the text into pieces referred to as tokens that then make up the each vector. An alphabetic tokenizer will ignore any words containing non-alphabetic characters. Use of the alphabetic tokenizer was particularly useful with this project's data, because it added a second chance to remove HTML tags not handled by the bot.

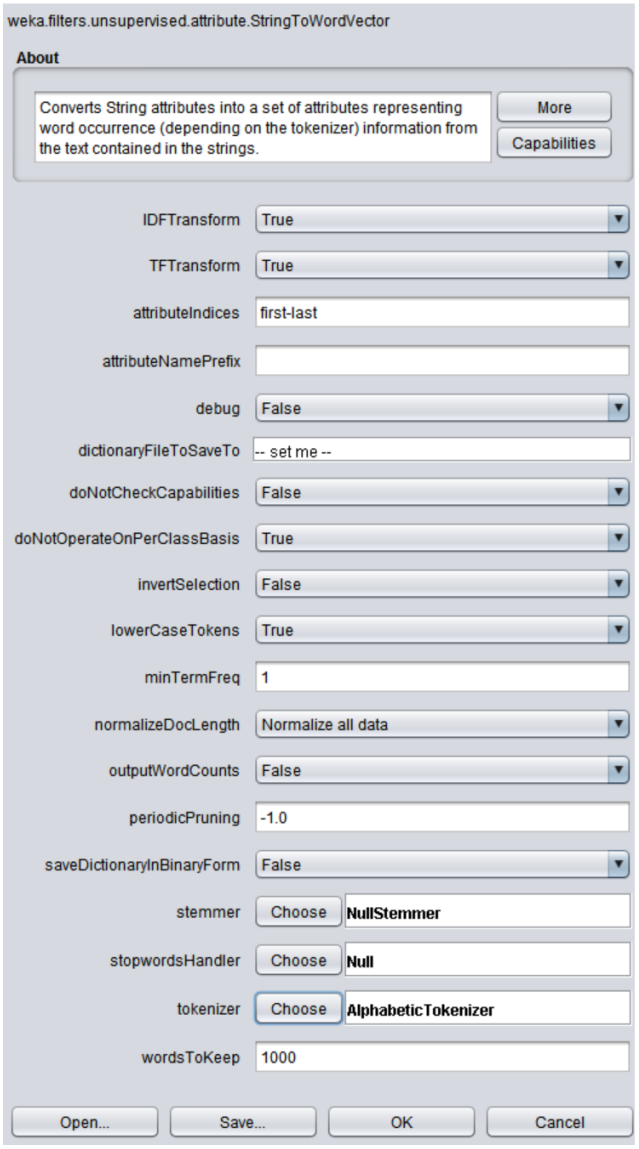


Figure 11: Settings from the StringToWordFilter.

Every piece of data from the training and testing datasets, as well as any future unseen data, was given the treatment described above. This allowed consistent representation of the problem between algorithms and datasets. Feeding data that had not been alphabetically tokenized into a classifier not built with similar data would produce incorrect results. An obvious issue arises with using new data in a classifier built with a particular structure of document vector. The 1000 words chosen from a training dataset are unlikely to be the same 1000 words used in a vector created from unseen data. To help with this, Weka provides an intermediate class that will map between vectors and allow tokens not present in both the classifier and unseen data to be ignored. This is another reason to try and use a vector size that is relatively large, to allow a good chance that unseen data can also be represented in the tokens the classifier is used to dealing with. As the variability of the text domain increases then the size of vector may also need to increase to allow for this.

Naïve Bayes

```
Correctly Classified Instances        8227               85.7247 %
Incorrectly Classified Instances      1370               14.2753 %

Precision  Recall   F-Measure Class
0.948      0.850    0.896     notCourseInfo
0.749      0.880    0.809     courseInfo
0.508      0.815    0.626     unistatsInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 5551  771  211 |    a = notCourseInfo
  277 2407   50 |    b = courseInfo
   26   35  269 |    c = unistatsInfo
```

Table 13: Results from a Naïve Bayes classifier against the training data.

Accuracy from the training data for a Naïve Bayes classifier was clearly higher than that achieved through the native C# classifiers, with an accuracy of 85.7%. Aside from the higher accuracy and recall, the striking thing about use of this classifier is its consistently high recall for each class, with all being over 0.8. Precision was very high for non-course information, the combination of good precision and recall show a clear improvement for this class over the C# classifiers. Precision of Unistats information was low though, 0.508 meaning that almost half the pages identified as containing Unistats information did not. The F-measure scores across the classes were promising, although some improvement to the performance of Unistats classification would be desirable.

```
Correctly Classified Instances        1672               81.5212 %
Incorrectly Classified Instances       379               18.4788 %

Precision  Recall   F-Measure Class
0.828      0.864    0.846     notCourseInfo
0.797      0.747    0.771     courseInfo


=== Confusion Matrix ===
```

```
    a    b    c   <-- classified as
 1038  161    3 |    a = notCourseInfo
  215  634    0 |    b = courseInfo
    0    0    0 |    c = unistatsInfo
```

Table 14: Results from test data when classified using the Naïve Bayes model

Against the test data this classifier marginally improved recall of non-course information and reduced recall in identifying course information, the opposite effect was seen for precision. These changes did reduce the F-measure for both classes, although not too dramatically. Despite the reduction in F-measure for both classes, overall accuracy dropped only 4.2%, from 85.7% to 81.5%. Three pages were identified as Unistats information but on closer inspection these were examples where the scraping bot had failed and saved an error string as the page contents.

Nearest Neighbour (k = 1)

"k" is the variable used in Weka when describing the number of neighbours used to arrive at a classification for the unlabelled data; hence this type of classifier in Weka is known as IBk, where k can be controlled by the user. For this classifier I chose a k value of one, meaning that once the nearest neighbour is found, its own classification is assigned to the unlabelled data. When k is larger than one and nearest neighbours have different classes then the algorithm must decide between them; when k is larger than one, it should be set to a value that allows ties to be broken.

```
Correctly Classified Instances        8581               89.4134 %
Incorrectly Classified Instances      1016               10.5866 %

Precision  Recall   F-Measure Class
 0.927      0.928    0.928     notCourseInfo
 0.851      0.854    0.852     courseInfo
 0.581      0.555    0.567     unistatsInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 6063  367  103 |    a = notCourseInfo
  370 2335   29 |    b = courseInfo
  104   43  183 |    c = unistatsInfo
```

Table 15: Results from a Nearest Neighbour classifier, using k = 1.

Nearest Neighbour produced the highest overall accuracy seen so far with the training data at 89.4%. Precision and recall for the classifier show a very high accuracy at identifying non-course information and high accuracy for course information, but its ability to identify Unistats information was low. Looking at the confusion matrix it can be seen that the false negatives related to Unistats information were split in a roughly 2:1 ratio between not-course information and course information. Whilst no particular effort has been made to identify to the model that Unistats information can be considered a subset of course information, it is concerning that so many pages taken directly from the existing Unistats site could be classified in this way. F-measures were better for course and non-course information compared to Naïve Bayes, but the nearest neighbour approach did struggle a little more to identify Unistats information.

```
Correctly Classified Instances        1564               76.2555 %
```

```
Incorrectly Classified Instances       487               23.7445 %

Precision  Recall   F-Measure Class
0.722      0.967    0.827     notCourseInfo
0.910      0.473    0.623     courseInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 1162   40    0 |    a = notCourseInfo
  447  402    0 |    b = courseInfo
    0    0    0 |    c = unistatsInfo
```

Table 16: Results from Nearest Neighbour (k = 1) when classifying the test data

A 13% drop in accuracy when using the test data may suggest some overfitting with this method. Recall shows that the success for identifying course information dropped below 0.5, although a high precision does mean that when a page was classified as course information then this was generally correct. The reduction in recall reduced the F-measure for course information to lower than that of Naïve Bayes.

With this data, Nearest Neighbour (k = 1) is less accurate than Naïve Bayes but also loses its "lazy" advantage, as every page the bot encounters will need to be classified. Spending time training a model that can classify quickly will be an advantage over each instance taking more time, especially with a classifier like Naïve Bayes taking under ten seconds to construct against the current training data.

Nearest Neighbour (k = 4)

A Nearest Neighbour classifier using four neighbours was also created, with the hope that this would increase performance; four neighbours were used to allow ties between the three classes to be broken.

```
Correctly Classified Instances        8825               91.9558 %
Incorrectly Classified Instances       772                8.0442 %

Precision  Recall   F-Measure Class
0.936      0.956    0.946     notCourseInfo
0.903      0.880    0.891     courseInfo
0.659      0.521    0.582     unistatsInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 6246  218   69 |    a = notCourseInfo
  307 2407   20 |    b = courseInfo
  117   41  172 |    c = unistatsInfo
```

Table 17: Training results for Nearest Neighbour (k = 4)

This Nearest Neighbour classifier improved slightly on the overall accuracy compared to the one neighbour version. General performance as measured by F-score increased slightly for course and

non-course information but dropped for Unistats, which is unfortunate as this was already the worst performing class.

```
Correctly Classified Instances        1587               77.3769 %
Incorrectly Classified Instances       464               22.6231 %

Precision  Recall   F-Measure Class
0.756      0.906    0.824     notCourseInfo
0.815      0.587    0.682     courseInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 1089  113    0 |    a = notCourseInfo
  351  498    0 |    b = courseInfo
    0    0    0 |    c = unistatsInfo
```

Table 18: Nearest Neighbour (k = 4) results against the test data

Using four neighbours produced a similar drop in accuracy between training and test, dropping 14% compared to 13% with one neighbour; although this classifier was still more accurate against the test data compared to using one neighbour. Course information recall was 0.587, which is an improvement over the one neighbour algorithm; recall of non-course information was slightly reduced. Overall this algorithm fared slightly better than one neighbour, which is intuitive as it was consulting with more neighbours.

Random Tree

The Random Tree classifier in Weka can have the number of available attributes set manually or selected by default. The default selection chooses log2(#predictors) + 1, in this case I'm taking #predictors to be the total number of attributes. "k" is again used in Weka's implementation, but this time to represent the number of attributes to inspect; with 1000 words in each instance's vector, the number inspected at each node is 11.

```
Correctly Classified Instances        8388               87.4023 %
Incorrectly Classified Instances      1209               12.5977 %

Precision  Recall   F-Measure Class
 0.913      0.914    0.913     notCourseInfo
 0.819      0.820    0.819     courseInfo
 0.555      0.536    0.545     unistatsInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 5970  452  111 |    a = notCourseInfo
  462 2241   31 |    b = courseInfo
  109   44  177 |    c = unistatsInfo
```

Table 19: Results from training the random tree classifier against the training data.

Overall accuracy was again higher in this classifier than any seen in the native C# approach. Similar to Nearest Neighbour, Random Tree also showed strong performance in recall and precision for non-

course information and course information classes, but dropped off when identifying Unistats information.

```
Correctly Classified Instances        1102               53.7299 %
Incorrectly Classified Instances       949               46.2701 %

Precision  Recall   F-Measure Class
0.569      0.897    0.696     notCourseInfo
0.189      0.028    0.049     courseInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 1078  103   21 |    a = notCourseInfo
  816   24    9 |    b = courseInfo
    0    0    0 |    c = unistatsInfo
```

Table 20: Results from the Random Tree classifier against the test data.

Performance against the test data dropped well below that of the other Weka classifiers discussed so far; I suspect severe over fitting, given the reduction in accuracy from 87.4% to 53.7%. Recall, precision and the confusion matrix show that this classifier heavily favoured assigning the class of non-course information to pages, resulting in the lowest recall and precision rates for course information yet of 0.028 and 0.189. Given the very low success at identifying course information, the Unistats classifications were ignored.

Given the stronger performance of the other classifiers in their initial states, pruning was not undertaken; if this classifier had already been a strong contender then time would have been used to prune during a refinement stage.

Random Forest

A Random Forest of 100 Random Trees was constructed in Weka using the training data.

```
Correctly Classified Instances        8827               91.9767 %
Incorrectly Classified Instances       770                8.0233 %

Precision  Recall   F-Measure Class
0.938      0.957    0.947     notCourseInfo
0.902      0.877    0.889     courseInfo
0.657      0.533    0.589     unistatsInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 6253  215   65 |    a = notCourseInfo
  309 2398   27 |    b = courseInfo
  107   47  176 |    c = unistatsInfo
```

Table 21: Results of a Random Forest classifier built using the training data

Training results for Random Forest were similar to those of a single Random Tree in overall accuracy, the precision of course information and Unistats information was improved. Recall for Unistats information still remained low at 0.533.

```
Correctly Classified Instances        1202               58.6056 %
Incorrectly Classified Instances       849               41.3944 %

Precision  Recall   F-Measure Class
0.586      1.000    0.739     notCourseInfo
0.000      0.000    0.000     courseInfo


=== Confusion Matrix ===

    a    b    c   <-- classified as
 1202    0    0 |    a = notCourseInfo
  849    0    0 |    b = courseInfo
    0    0    0 |    c = unistatsInfo
```

Table 22: Random Forest Classifier on the test data

This Random Forest classifier suffers from extreme over fitting, to the point that every instance was classified as not course information. With the overfitting of both this classifier and the individual random tree, I suspect pruning would be vital to properly use these classifiers in production.

Support Vector Machine

In Weka a support vector machine is known by the acronym SMO, the following are results from training and cross validating an SMO classifier on the training data.

```
Correctly Classified Instances        8840               92.1121 %
Incorrectly Classified Instances       757                7.8879 %

Precision  Recall   F-Measure Class
0.948      0.947    0.947     notCourseInfo
0.899      0.884    0.891     courseInfo
0.623      0.727    0.671     unistatsInfo

=== Confusion Matrix ===

    a    b    c   <-- classified as
 6184  238  111 |    a = notCourseInfo
  284 2416   34 |    b = courseInfo
   56   34  240 |    c = unistatsInfo
```

Table 23: Results from an SMO classifier against training data.

As with many of the other machine learning classifiers, the SMO produced highly favourable statistics for non-course information during training, with very good performance also found in classifying course information. The F-measure for Unistats information was the highest compared to the other machine learning classifiers discussed, but still a little lower than would be wanted in a final model.

```
Correctly Classified Instances        1536               74.8903 %
Incorrectly Classified Instances       515               25.1097 %
```

```
Precision  Recall   F-Measure Class
0.725      0.921    0.811     notCourseInfo
0.820      0.505    0.625     courseInfo

=== Confusion Matrix ===

    a    b    c   <-- classified as
 1107   94    1 |    a = notCourseInfo
  420  429    0 |    b = courseInfo
    0    0    0 |    c = unistatsInfo
```

Table 24: Results from SMO classifier against test data.

The SMO classifier saw a drop in accuracy against the training data similar to that of all the other machine learning classifiers, with the exception of Naïve Bayes. Recall for course information dropped significantly to 0.505, reducing F-measure for the class with it. F-measures for both classes in the test data showed the SMO to be very similar in performance to a k=1 nearest neighbour approach.

First Stage Selection of a Classifier

Having tested basic implementations of different native C# and Weka algorithms, a decision on narrowing the scope classifiers being used was required. From the training and testing data I decided that Naïve Bayes had been the most successful based on accuracy and F-measures produced in testing compared to training. All of the other Weka classifiers produced better results on the training data but had a larger reduction in metrics against the testing data. A Naïve Bayes approach gave a lower initial accuracy in training but was then more consistent on the previously unseen data; I value this characteristic highly as it reduces the requirement for training data and unseen data to be alike. Variability is inherent in the domain that the classifier will be operating in, so producing strong results on test data suggests it will also be able to cope in a production setting with unlabelled data. Because of this reason I chose to investigate refining the Naïve Bayes classifier further.

Second Stage Classifier Implementation

The Naïve Bayes classifier implemented in stage one showed a good level of performance, but also had room for improvement; particularly around identifying Unistats data. The second stage of implementation briefly explored changes in training data that could lead to improved performance.

Naïve Bayes Using Cleaned Training Data

As mentioned previously, the training data contained information drawn from pages outside of the sector. In this attempt at refinement, the out of sector pages were removed from the training data. This was to test the effect of the noise caused by out of sector websites in the data. The subset was used to train a Naïve Bayes classifier to make a distinction between course information and non-course information. Unistats itself was excluded from this data, in favour of possibly using some other mechanism to indicate possible Unistats information (discussed later).

```
Correctly Classified Instances        6402               89.9536 %
Incorrectly Classified Instances       715               10.0464 %

Precision  Recall   F-Measure Class
0.933      0.901    0.917     notCourseInfo
0.850      0.896    0.873     courseInfo


=== Confusion Matrix ===

    a    b   <-- classified as
 3951  432 |    a = notCourseInfo
  283 2451 |    b = courseInfo
```

Table 25: Results from a Niave Bayes classifier, trained using only institutional web pages.

When trained using data without non-sector websites, the Naïve Bayes approach produced a higher overall accuracy of 89.9% versus 85.7% of the original classifier. Both classes had individual recalls very near to 0.9.

```
Correctly Classified Instances        1651               80.4973 %
Incorrectly Classified Instances       400               19.5027 %

Precision  Recall   F-Measure Class
0.819      0.856    0.837     notCourseInfo
0.783      0.733    0.757     courseInfo

=== Confusion Matrix ===

    a    b   <-- classified as
 1028  173 |    a = notCourseInfo
  227  623 |    b = courseInfo
```

Table 26: Results from the Naïve Bayes classifier against the original test data

Surprisingly, this new Naïve Bayes classifier was slightly less accurate against the same test data. The F-measure for both classes were also slightly reduced. These drops in metrics suggest that the non-sector web sites might help with avoiding over fitting. From this result, it would appear that the Naïve Bayes previously created is a better fit for production use.

Expectation Maximisation Clustering and Naïve Bayes

Witten et al. discuss the use of EM (Expectation Maximisation) clustering in combination with a Naïve Bayes classifier, as a potential solution to the common problem of lacking unlabelled data to build models from. Until this point, all the labelling on the model and test data that has been used in the project has been conducted based on the page's URL. If there is a natural distinction between the content of course information and non-course information pages, a clustering algorithm should be able to partition the two. Creating a Naïve Bayes model from data labelled according to EM clustering would allow me to also see if my previous automated labelling had caused an impact. If I had misunderstood the structure of the websites contributing to the model then many more non-course information pages could have been wrongly classified than I expected, causing inaccuracies in the model; EM clustering would offer a view on class based purely on the content of each page.

```
Incorrectly clustered instances :   1060.0   14.8939 %
```

Table 27: Accuracy of EM clustering on the reduced training data.

Using the reduced training data as described in the previous Naïve Bayes model, the EM clustering algorithm disagreed with the URL based classifications 14.9% of the time. The total size of the training data is 7117 instances, so 1060 disagreements is not insignificant but I would expect the models to be comparable in performance. The labelled data created from the EM clustering was used to train a Naïve Bayes classifier, which was also run against the standard test data.

Correctly Classified Instances	1636	79.766 %
Incorrectly Classified Instances	415	20.234 %

Table 28: Results from a Naïve Bayes classifier, trained using data labelled through EM clustering.

An overall accuracy of 79.8% suggests that any impact caused through disagreements between EM clustering and URL based labelling did not have a large affect, although this comparison is using test data labelled using the latter method. I would expect the URL based model to fair better than the EM clustered one, it performing very similarly suggests both methods produce workable models. I was intrigued as to whether the performance would hold or be switched if the test data was also labelled through EM clustering rather than URLs, so ran the EM clustering algorithm on the test data.

Incorrectly clustered instances :	976.0	47.5865 %

Table 29: Results from the EM clustering applied to the test data

Unfortunately the EM clustering and URL labelling approach disagreed 47.6% of the time, perhaps the reduced number of instances (c.2000) and only having four different universities made it difficult for the clustering to spot the difference between course information and not. Because of this large disagreement I did not continue to test this refinement of Naïve Bayes against this test data.

Summary

Method and Results

In keeping with the “no free lunch” theorem, a range of text classifiers have been built and tested. Their purpose was to aid in the identification of course information on university websites, in particular course information that may meet Key Information Set (KIS) standards. KIS information is currently displayed in the Unistats website, but this is likely to change in the near future. The timing of the project meant that changes to regulation around KIS and Unistats were not completely finalised, or implemented by universities. This meant the training data for classifiers had to use Unistats itself to learn what KIS data could look like. The main disadvantage of this was Unistats being only one example of how to present the data to a reader. In time, each University will have slightly different wordings and expressions, as they do on their current course information pages. To more closely represent the classifier’s production environment, the test data only contained information from university websites and not Unistats. All of the training and test data was automatically labelled using inspection of root URLs.

	Training		Test	
Classifier	Accuracy	Average F	Accuracy	Average F
Random	30.6%	0.259	-	-
Random Proportional	54.9%	0.325	48.1%	0.450
Any Key Word	69.3%	0.627	53.5%	0.381
All Key Word	72.6%	0.691	63.9%	0.523
Naïve Bayes	85.7%	0.777	81.5%	0.809
Nearest Neighbour (k=1)	89.4%	0.782	76.3%	0.725
Nearest Neighbour (k=4)	91.6%	0.806	77.4%	0.753
Random Tree	87.4%	0.759	53.7%	0.373
Random Forest	92.0%	0.808	58.6%	0.370
Support Vector Machine	92.0%	0.836	74.9%	0.718
Naïve Bayes (Cleaned Data)	89.6%	0.895	80.5%	0.797

Table 30: Summary results from the constructed classifiers.

Initially, classifiers based on purely random chance were created, this offered a baseline performance against the data. Any subsequent classifier would have to outperform these by a clear margin to be worth considering. Before using machine learning techniques, basic classifiers built on native C# string functions were created. This gave a baseline of performance achievable by any programmer who is not familiar with machine learning methods. In both of these native classifiers, performance was easily tuned to training data but suffered in testing. Such a difference is representative of a main challenge in this project, which is the variability between text different universities use on their websites.

Despite difficulties in the data, several machine learning classifiers were created that performed well in both training and testing. The two tree based approaches were clearly over fitted to the training data. It is likely they could be built and used for this classification task but time, inexperience and other better performing classifiers meant pruning activities were not undertaken. Aside from the tree based algorithms, all the machine learning classifiers out performed random selection and simple rule based approaches.

A Naïve Bayes classifier was chosen from the classifiers as being most likely to offer the best performance in the final application. This decision was based on having the strongest performance in the F-measure metric, whilst also having the lowest drop in performance from training to testing. Two methods were briefly explored to increase performance of the classifier by adjusting the training data. The first was to remove data from outside the higher education sector from the training data. This was initially included before the context of project had been fully decided and could have added unnecessary noise to the classifier. Removing the non-sector data did not increase the performance however. Finally an attempt was made to build and test a Naïve Bayes classifier with data labelled through a clustering method. The idea was to remove false labels in the original training data caused by the automatic labelling method. This approach produced workable clusters for the training data, but not the test data. Because of this it would have been impossible to test any classifier made from the training data.

Potential Benefits

The initially created Naïve Bayes classifier would likely be of a good enough quality to use in production, especially when Universities become more explicit with display of KIS data featuring words currently used on Unistats. There are two examples I can offer for how the classifier can reduce manual workload, the first is from identifying Unistats data in training. 530 pages were identified during training as Unistats information, of which roughly half were correct. Assuming a person inspecting the same web pages would correctly identify all course information, there were 3064 pages of potential Unistats information. There were 330 actual Unistats pages, which are the closest approximation to how a University might display the information. Unistats pages made up roughly 11% of all the course information. Use of the classifier to identify Unistats information would reduce the manual search space to the 530 pages labelled as such. This means that if a person was required to manually inspect some of these pages, they would find Unistats information in every other page rather than every tenth page.

The second example is from identifying only course information in the test data. With the number of higher education providers subject to KIS regulations increasing, many new providers of varying size and complexity are going to be encountered. It is likely that at least some of the providers new to HEFCE and KIS will not structure their websites in a common way, both in URLs and the prominence of their course information. Even discounting any capability to identify Unistats information on webpages, having a classifier able to reliably identify course information would save manual effort. All that is required is for the domain of the provider to be identified and the web bot described earlier will be able to crawl the domain finding and prioritising course information. This would be simple for a human but, as mentioned in the context section, quickly scales beyond a reasonable task and also may need to be performed again at short notice. A bot can be run at all hours of the

day, storing the information it finds in a database ready to be inspected by the analysts at HEFCE. The Naïve Bayes classifier initially developed had a precision suggesting up to 80% of the pages collected and classified as course information would be correct. A human would likely achieve a higher precision, but their time would be more expensive and motivation likely to fade with such a task.

Project Issues

A personal preference of mine is to use large projects to forward my understanding in an area that was previously unknown to me. This project was my first time studying machine learning, which has meant decisions made earlier in the project I have sometimes later thought incorrect. In many cases these did not seem to detract from the impact that the project could have. However, I do feel like the project has been a starting point on solving the particular issue, rather than a comprehensive answer. This is unsurprising now that I understand how vast machine learning is.

In particular, I feel the success of the Naïve Bayes classifier is partly due to its robustness against issues that I may have inadvertently introduced. For instance, the data used a TF-IDF weighting so was on some level a measure of frequency. Reflecting later on Naïve Bayes, in order to expand areas of this report, I now realise I should have explicitly selected a Naïve Bayes classifier meant for use with data containing frequency values. According to Witten et al., I should have selected a multinomial Naïve Bayes implementation. In this case the classifier created is still usable, so either: the standard Naïve Bayes implementation in Weka is intelligent enough to cope with the mistake, and self-select a multinomial approach; or, the non-zero TF-IDF weights were simply treated as 1 and therefore True with a Boolean approach. Regardless, this example highlights the risk of using someone else's implementation of any algorithm. There are benefits to be had in saving time to implementation, but without writing similar yourself it can be difficult to fully understand what has been created. Keeping this in mind, it is not unlikely that some of the other classifiers tested would be much more successful once I have more experience in the field; the failure of the Tree based algorithms, for example.

Generally I believe the data collected was of a high enough quality for the task, but with more understanding of the algorithms I might have treated it differently between implementations. With the claimed strengths of an SVM in highly dimensional problems for instance, I might have recreated the vectors to offer a much higher word count. As it happened, I settled on an approach of not changing the treatment of the training data, in order to gain experience and knowledge of all the classifiers I managed to test. To experiment with the data processing for each algorithm would have been an overwhelming task for my first machine learning project.

Future Development

Decorated Naïve Bayes Classifier

In order to classify pages as holding data, the previously described Naïve Bayes classifier will be reliant on universities displaying information in a wording similar to that of Unistats. It would be

better not to rely on this occurring, particularly in the early implementation of new regulations when universities may change their sites incrementally. To account for this, I would recommend decorating the Naïve Bayes classifier with a simple C# classifier. Decoration is the programming principal of wrapping one object in another to perform some intermediate task. In this case intercepting the classification and change a course information class to Unistats based on some rule. In the decorated Naïve Bayes classifier, each instance given the class "courseInfo" would be inspected again by a simple key word classifier. The key word classifier would change the "courseInfo" class if a key word or several key words were found. This approach increases the chance of identifying pages displaying a little, but not complete KIS data. It also mimics the reality of these classes more accurately, in that Unistats information is a class within course information. Whilst the keyword classifiers did not have high overall performance, the highest F-measures for identifying Unistats information were achieved with the Any and All Keyword classifiers. Naïve Bayes has proved itself very good at differentiating course and non-course information, so that power should also be utilised. Using a simple classifier to supplement the more complex one would allow an operator of the production system to easily adjust the words that indicate potential Unistats information. This is a benefit because it combines two separately powerful approaches and because the project is happening at time of regulatory change. It was originally thought that the final requirements for data display would be clear during the project, but they were not. As the data on websites changes and is recollected then an analyst could easily change the key words and search for new display of KIS data. The design of the data collection system also allows reclassification to happen without explicitly recollecting the data, so an iterative process could be used on each collection to produce a list of sites worthy for human investigation. Once the first few universities were found to be correctly displaying the new data, these could be used to redevelop the Naïve Bayes classifier and remove the simple keyword search.

General System Development

In order to use the data collection system, the operator must currently run SQL queries on the central database to set behaviour like domains to crawl and 24 hour page limits. They also have to adjust source code if they want to change the amount of work requested by either the web scraping bots or the classifier. Given the system's existing web API and database backend, adding a graphical user interface (GUI) would be feasible. A simple web application could be built to communicate with the database through the web API. The database would store a table of system settings that could be adjusted through the web application. Bots would operate on a separate machine, which administrators would have access to in order to troubleshoot issues; Selenium does tend to leave a browser open when a connection is dropped. The GUI would become another part of the system that could be distributed and hosted from an appropriate machine in the network, but available to all internal users with the right authorisation.

University websites are potentially very large, with several observed in the project featuring expansive news and staff sections. I believe the system in its current configuration would eventually find and stick to course information, but there is certainly potential for it to become stuck in other highly connected areas of a website for some time. This issue occurred during testing the web scraping bot with the final classifier and was mainly related to starting the bot at the university's homepage. From this point if the bot found its way to a news section, for instance, it would quickly

amass a large amount of links to other news articles and have no or very few links to course pages. It was simple to counter this effect by starting the bot on a course information page or an “A-Z” course search. The latter is, by definition, highly linked to course information pages. For an example based on Lincoln University see appendix C. A production version of the scraping system could help this issue by perhaps adding more variability to the way URLs are selected. Early in development I briefly explored string similarity concepts like “fuzzy matching” and “Soundex”. If no course information has been found yet, the production system could choose URLs with top levels that are statistically distant from each other. This could help the bot select between different areas of website that normally appear in the style of `someUniversity.ac.uk/news/` or `someUniversity.ac.uk/courses/`.

References

Aggarwhal, C. C. and Zhai, C. (2012) 'A Survey of Text Classification Algorithms', *Mining Text Data*. New York: Springer, pp. 163-222.

Aickelin, U. 2015. Machine Learning Methods. *In:* Riley, S. (ed.) *Computerphile.* YouTube.

Alpadym, E. (2014) *Introduction to Machine Learning.* Massachusetts: MIT.

Ambler, S. 2002. Agile Modelling: Effective Practices for eXtreme Programming and the Unified Process. New York: John Wiley & Sons.

BIS (2015) *Consultation on alternative providers of higher education - Improving quality and value for money.* London: BIS.

Brody, H. (2012) *I Don't Need No Stinking API: Web Scraping For Fun And Profit*. Available at: https://blog.hartleybrody.com/web-scraping/ (Accessed: 21st October 2016).

Brownlee, J. (2016) *Classification And Regression Trees For Machine Learning*: Machine Learning Mastery. Available at: http://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/ (Accessed: 3rd November 2016).

CardiffUniversity (2016) *Accounting (BSc)*: Cardiff University. Available at: http://www.cardiff.ac.uk/study/undergraduate/courses/course/accounting-bsc (Accessed: 19th October 2016).

CastleProject (2014) *Windsor*. Available at: http://www.castleproject.org/projects/windsor/ (Accessed: 8th September 2016).

DELNI, HEFCE, HEFCW and SFC (2015a) *Review of information about learning and teaching, and the student experience: Consultation on changes to the National Student Survey, Unistats and information provided by institutions.* Bristol: HEFCE.

DELNI, HEFCE, HEFCW and SFC (2015b) *UK review of information about higher education, Report on the review of the Key Information Set and Unistats* Bristol: HEFCE.

Descoins, A. (2015) *Why accuracy alone is a bad measure for classification tasks, and what we can do about it*: Tryo Labs. Available at: https://tryolabs.com/blog/2013/03/25/why-accuracy-alone-bad-measure-classification-tasks-and-what-we-can-do-about-it/.

Fisher, R. A. (1936) 'The Use of Multiple Measurements in Taxonomic Problems', *Annals of Eugenics (Now: Annals of Human Genetics),* 7(2), pp. 179-188.

Frijters, J. (2014) *IKVM.NET*. Available at: http://www.ikvm.net/ (Accessed: 15th September 2016).

Guruswamy, K. (2015) *Data Science - Machine Learning vs Rules Based Systems*. Available at: https://www.linkedin.com/pulse/data-science-machine-learning-vs-rules-based-karthik-guruswamy (Accessed: 22nd October 2016).

Hall, M., Eibe, F., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. (2009) 'The WEKA Data Mining Software: An Update', *SIGKDD Explorations,* 11(1).

HEFCE (2011) *Key Information Set, publication of technical guidance and further information.* Bristol: HEFCE.

HEFCE (2012) *Key Information Sets and Unistats, Overview and Next Steps.* Bristol: HEFCE.

HEFCE (2016) *Data collection for colleges*. Available at: http://www.hefce.ac.uk/lt/unikis/fecdata/ (Accessed: 19th October 2016).

HEFCE, SFC, HEFCW and DfE (2016) *Review of information about learning and teaching, and the student experience: Summary of responses to consultation on changes to the National Student Survey, Unistats and information provided by institutions.* Bristol: HEFCE.

HEFCE, UUK and GuildHE (2010) *Public information about higher education: Consultation on changes to information published by institutions.* Bristol: HEFCE.

HEFCE, UUK and GuildHE (2011) *Provision of information about higher education: Outcomes of consultation and next steps.* Bristol: HEFCE.

HtmlAgilityPack (2012) *HTML Agility Pack*. Available at: https://htmlagilitypack.codeplex.com/ (Accessed: 8th September 2016).

Jensen, M. (2009) *Program to an interface, not an implementation*. Available at: http://www.fatagnus.com/program-to-an-interface-not-an-implementation/ (Accessed: 8th September 2016).

Joachims, T. 'Text Categorization with Support Vector Machines: Learning with Many Relevant Features', *European Conference on Machine Learning*: Springer.

Kawaguchi, K. (2000) *Linear Separability and the XOR Problem*. Available at: http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html (Accessed: 13th November 2016).

Makabee, H. (2012) *Separation of Concerns*. Available at: https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/ (Accessed: 8th September 2016).

McNulty, E. (2015) *What's The Difference Between Supervised And Unsupervised Learning?*: Dataconomy. Available at: http://dataconomy.com/whats-the-difference-between-supervised-and-unsupervised-learning/ (Accessed: 22nd October 2016).

Microsoft (2007) *The Repository Pattern*: Microsoft. Available at: https://msdn.microsoft.com/en-us/library/ff649690.aspx (Accessed: 9th September 2016).

Reitz, K. (2016) *HTML Scraping*. Available at: http://docs.python-guide.org/en/latest/scenarios/scrape/ (Accessed: 8th September 2016).

RestSharp (2016) *RestSharp*. Available at: http://restsharp.org/ (Accessed: 8th September 2016).

Scrapy (2016) *Scrapy*. Available at: https://scrapy.org/ (Accessed: 8th September 2016).

SeleniumHQ (2016) *Selenium*. Available at: http://www.seleniumhq.org/ (Accessed: 8th September 2016).

Unistats (2016) *Accounting - Unistats* (Accessed: 19th October 2016).

UniversityOfWaikato (2009) *Weka 3: Data Mining Software*. Waikato, New Zealand. Available at: http://www.cs.waikato.ac.nz/ml/weka/index.html (Accessed: 13th September 2016).

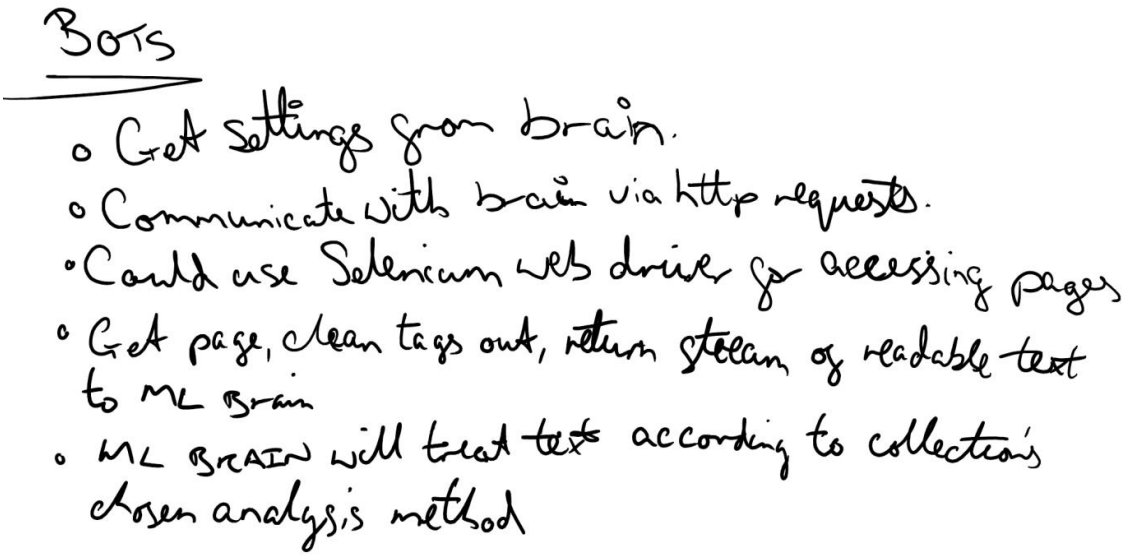
WEKA (2009a) *IKVM with WEKA Tutorial*. Available at: https://weka.wikispaces.com/IKVM+with+Weka+tutorial (Accessed: 15th September 2016).

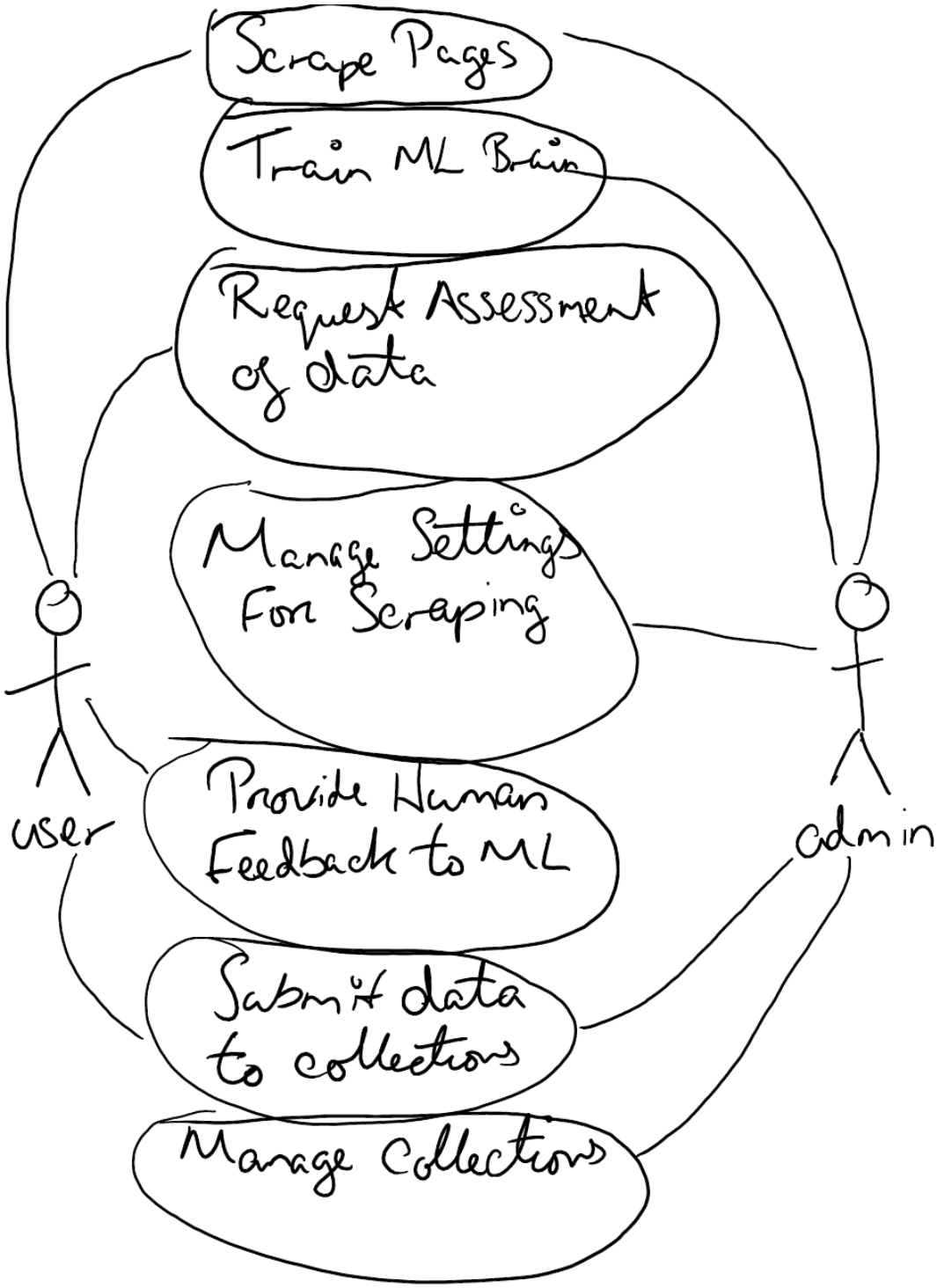
WEKA (2009b) *Use WEKA with the Microsoft .NET Framework*. Available at: https://weka.wikispaces.com/Use+WEKA+with+the+Microsoft+.NET+Framework (Accessed: 15th September 2016).

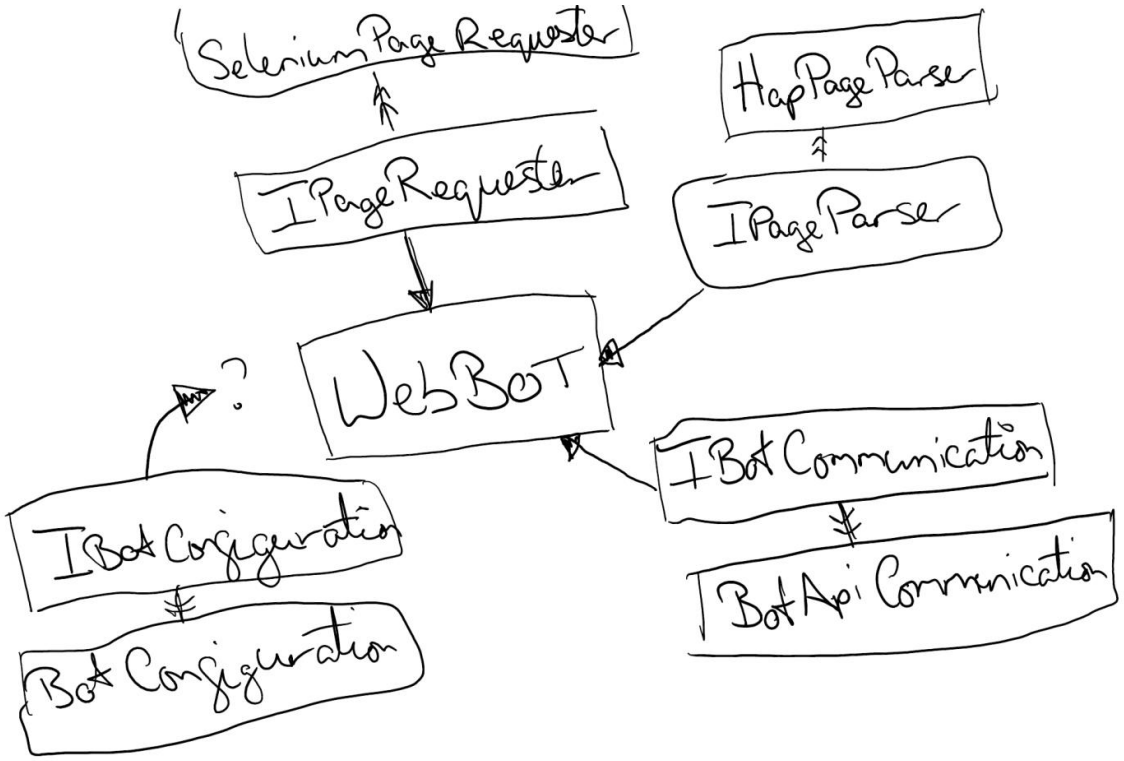
Witten, I. H., Eibe, F. and Hall, M. A. (2011) *Data Mining: Practical Machine Learning Tools and Techniques.* Third edn.

Appendix A – Initial Rough Modelling

The following are some of the early hand drawn models used to clarify my approach and structure the first coding explorations.




Scrape Pages
Train ML Brain
Request Assessment of data
Manage Settings For Scraping
Provide Human Feedback to ML
Submit data to collections
Manage Collections
user
admin



Site

address
lastScraped
content
pageId

Collection

CollectionId
Collection
Analysis method

Collection members

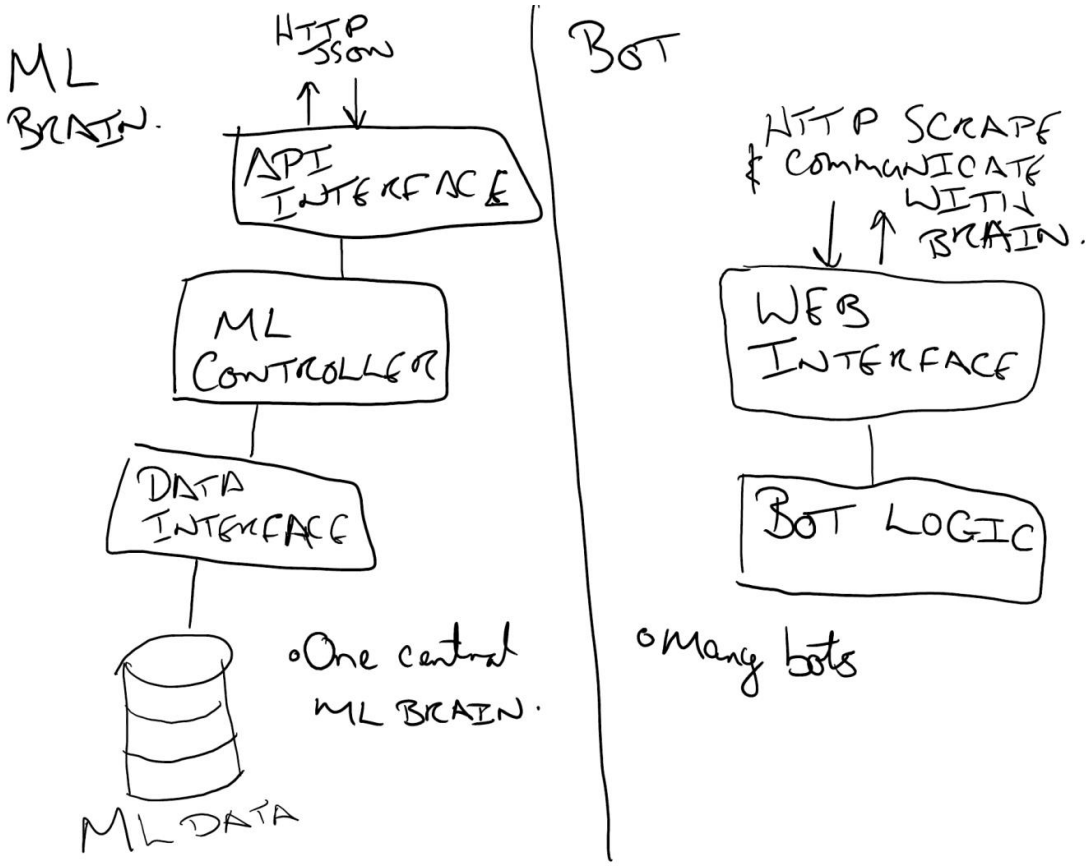
collection id
Page id

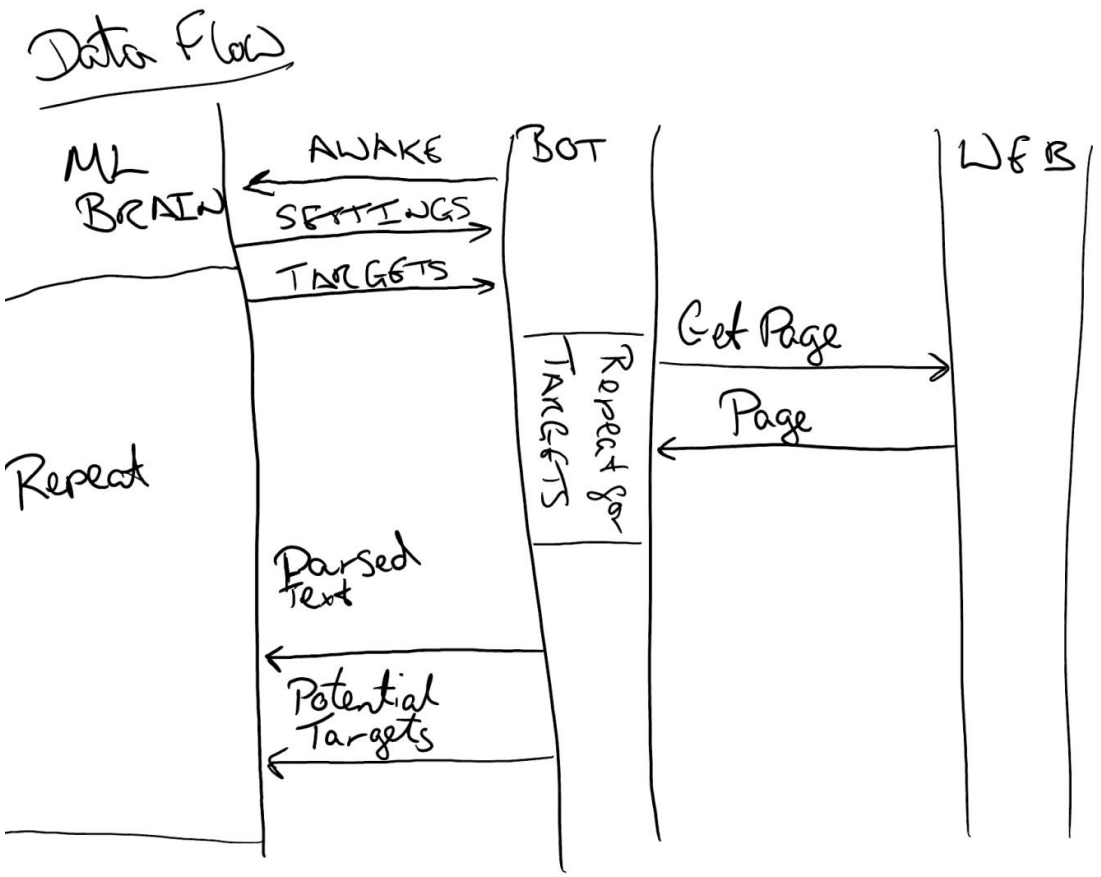
Result type

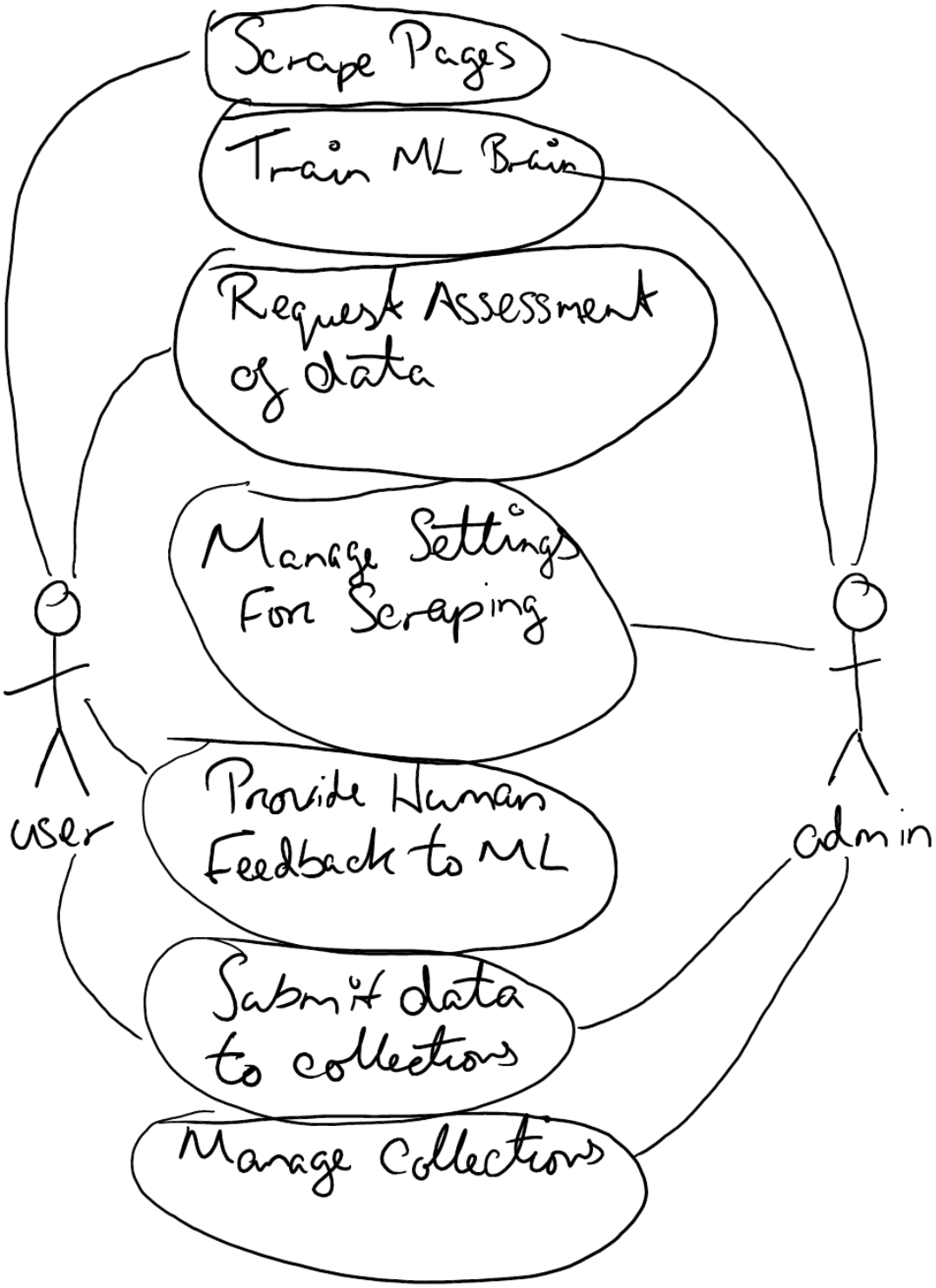
Collection id
Result id
Result

Result

Page id
Collection id
Result id.
HumanChecked

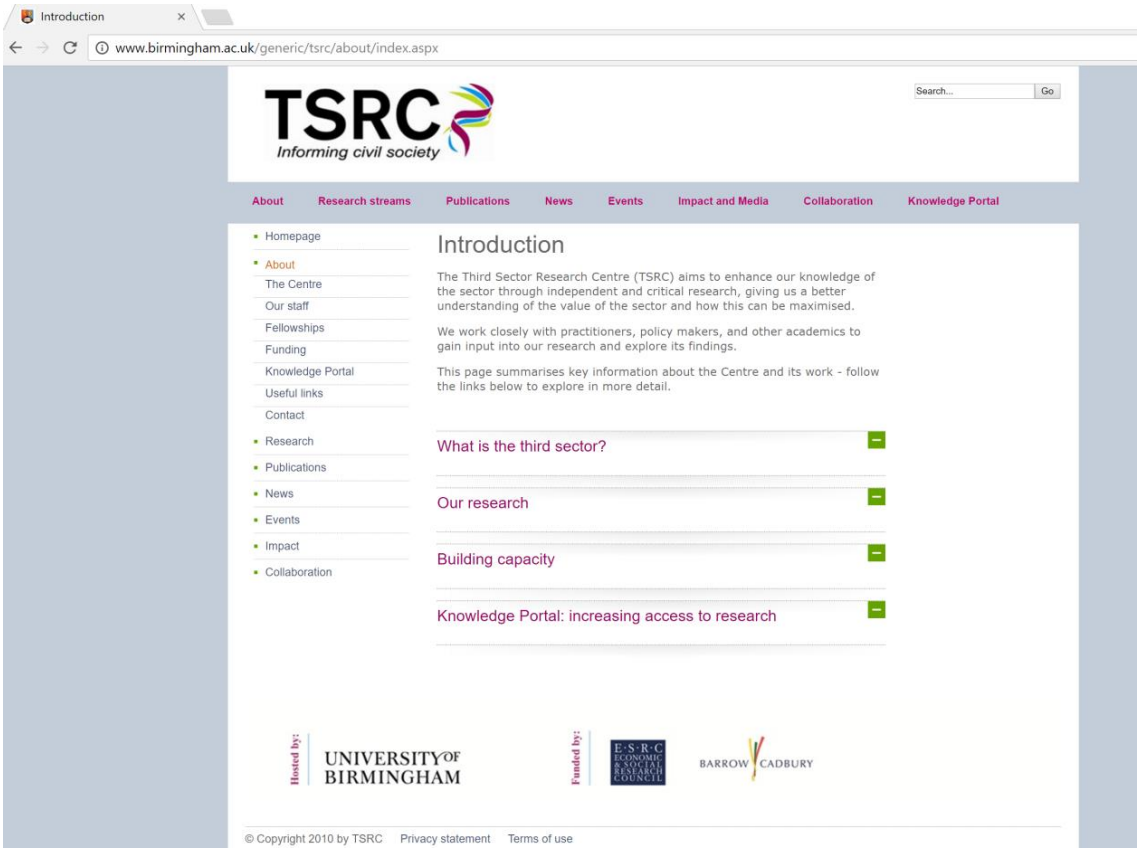

ML BRAIN.
HTTP JSON
API INTERFACE
ML CONTROLLER
DATA INTERFACE
ML DATA
•One central ML BRAIN.
BOT
HTTP SCRAPE & COMMUNICATE WITH BRAIN.
WEB INTERFACE
BOT LOGIC
•Many bots


Data Flow
ML BRAIN
AWAKE
SETTINGS
TARGETS
BOT
WEB
Get Page
Page
Repeat for TARGETS
Repeat
Parsed Text
Potential Targets


Scrape Pages
Train ML Brain
Request Assessment of data
Manage Settings For Scraping
Provide Human Feedback to ML
Submit data to collections
Manage Collections
user
admin

Appendix B – Real Web Page Scraping

This page from the university of Birmingham is relatively simple, but the HTML code underneath is still large and complex. It's clear the bot doesn't remove all of the extra information from the HTML that it could do, but comparing the original source code for the page and what was stored in the database shows the bot did a passable job.



The page as rendered by Google Chrome.

The first 88 lines of HTML making up the page, the total number of lines is 274.

About Introduction The Centre Our Staff Fellowships Funders Volunteering at TSRC Useful links Contact Us Research streams - Below the radar - Quantitative analysis - 'Real Times' - Service delivery - Theory and policy Previous research: - Economic and social impact - Environment - Equalities - Evidence reviews - Social enterprise - Workforce and workplace Publications Research papers Journal articles Major reports Other publications News Latest news Newsletter subscription Events Impact and Media Media coverage log Below the radar Self-help housing Social enterprise Collaboration Joint research Evaluations Knowledge sharing Collaborative research ESRC seminar series Knowledge Portal About the Portal tsrc>About <iframe src="//www.googletagmanager.com/ns.html?id=GTM-MM4Z3Q" height="0" width="0" style="display:none;visibility:hidden"></iframe> Browser does not support script. Homepage About The CentreOur staffFellowshipsFundingKnowledge PortalUseful linksContact ResearchPublicationsNewsEventsImpactCollaboration Introduction The Third Sector Research Centre (TSRC) aims to enhance our knowledge of the sector through independent and critical research, giving us a better understanding of the value of the sector and how this can be maximised. We work closely with practitioners, policy makers, and other academics to gain input into our research and explore its findings. This page summarises key information about the Centre and its work - follow the links below to explore in more detail. What is the third sector? The third sector is a debated term. We take a broad definition, to include all organisations operating outside the formal state or public sphere that are not trading commercially for profit in the market. This means charities and voluntary organisations, community groups, social enterprises, cooperatives and mutuals. Whilst these organisations are exceptionally diverse, they share a broad common theme of being value driven. Our research Our research is divided into research streams focused on different areas of the sector. All research papers are published on our website, along with summaries and short ‘briefing papers’. Browse our research streams Building capacity TSRC is reviewing existing research and building comprehensive databases to create lasting resources for third sector research. We are also working to grow the sector's potential to use and conduct research, through collaborative research projects. Knowledge Portal: increasing access to research TSRC is committed to helping people access research on the sector. Our Knowledge Portal enables people to search for and access a wide range of research from a number of sources. Knowledge Portal © Copyright 2010 by TSRC Privacy statement Terms of use Browser does not support script. </form>

The HTML after parsing by the bot, notice some HTML tags remain.

Appendix C – Bot Traversal Using Lincoln University

Lincoln University was used because it didn't feature in the training or testing data of the project. The website does also feature a common root for course pages, which helps the web scraping system in its current configuration.

The web bot was started on a course information page and left to scrape around 50 pages before the classifier began its work.

Address
http://www.lincoln.ac.uk/home/course/biobioub/
http://www.lincoln.ac.uk/course/bvsbvsum
http://www.lincoln.ac.uk/course/zoozooub
http://www.lincoln.ac.uk/home/campuslife/studentsupport/careersservice/
http://www.lincoln.ac.uk/course/eqsabwum
http://www.lincoln.ac.uk/course/biochmum
http://www.lincoln.ac.uk/home/contacts/
http://www.lincoln.ac.uk/home/webteam/
http://www.lincoln.ac.uk/home/minicom/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/howtoapply/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/feesandfunding/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/parentsguide/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/askaquestion/
http://www.lincoln.ac.uk/virtualopenday
http://www.lincoln.ac.uk/home/studyatlincoln/scholarships/
http://www.lincoln.ac.uk/home/studyatlincoln/partnerinstitutions/
http://www.lincoln.ac.uk/home/studyatlincoln/typesofcourses/
http://www.lincoln.ac.uk/home/research/
http://www.lincoln.ac.uk/home/course/biobioub/?d=2016-17
http://www.lincoln.ac.uk/home/course/biobioub/?d=2017-18
https://www.lincoln.ac.uk/howtoapply
http://www.lincoln.ac.uk/opendays
https://www.lincoln.ac.uk/virtualopenday
http://www.lincoln.ac.uk/2ak/schoolstaff
http://www.lincoln.ac.uk/course/eqsabwub
http://www.lincoln.ac.uk/home/studyatlincoln/shortcourses/
http://www.lincoln.ac.uk/home/accessibility/
http://www.lincoln.ac.uk/course/zoozooum
http://www.lincoln.ac.uk/home/studyatlincoln/
http://www.lincoln.ac.uk/course/biochmum?d=2016-17

http://www.lincoln.ac.uk/course/BUSIBMUB/
http://www.lincoln.ac.uk/course/CMPCMSUM/
http://www.lincoln.ac.uk/course/CONCONUB/
http://www.lincoln.ac.uk/course/ECOFINUB/
http://www.lincoln.ac.uk/course/EGRELCUB/
http://www.lincoln.ac.uk/course/ELECNSUB/
http://www.lincoln.ac.uk/course/ENLHSTUB/
http://www.lincoln.ac.uk/course/FDSOPMUB/
http://www.lincoln.ac.uk/course/GEPGEPUB/
http://www.lincoln.ac.uk/course/ISGISGUB/
http://www.lincoln.ac.uk/course/ISTISTUB/
http://www.lincoln.ac.uk/course/JOUINVUB/
http://www.lincoln.ac.uk/home/international/internationalscholarships/
http://www.lincoln.ac.uk/home/opendays/postgraduatevisits/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/howtoapply/faq/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/howtoapply/personalstatements/
http://www.lincoln.ac.uk/home/studyatlincoln/undergraduatecourses/teachingandlearning/researchengagement/
https://www.lincoln.ac.uk/home/applicants/
https://www.lincoln.ac.uk/home/international/erasmusopportunities/
https://www.lincoln.ac.uk/home/research/

The URLs listed above are the pages scraped on the Lincoln website in order, before the classifier had assessed any content. Included are pages that would clearly not be course information based on the URL, like "…home/research/", "…opendays" and "…schoolstaff". Once the classifier had assessed these pages it found one of the pages from the "…/course/" node to be course information. From this point on the bot stuck to course URLs. This was the case despite subsequent classifications assessing very few of the pages returned to be course information.

	lastNode	nCourseInfo	nPages	hitRate
1	http://www.lincoln.ac.uk/course/	9	311	0.02893891
2	http://www.lincoln.ac.uk/home/course/	1	4	0.25
3	http://www.lincoln.ac.uk/course/GEPGEPUB/	1	1	1

Several returned 404 pages after appearing to be old links, but many more were clearly incorrectly classified. In this case, the good selection of starting point meant that poor performance of the classifier was compensated for. The pages scraped in the first 50 (pre classification) were not close enough to course information to get the class and cause further scraping; high precision in classifying course information stopped the bot going down the wrong path. A production system would benefit from being able to override the classifications and then include them into another iteration of the automatic classifier.