Masters' Project
Michaelmas Progress Report

Patrick Lewis
Queens' College

December 2015

1 Introduction

1.1 Modern Scientific Publishing

With the widespread adoption of the internet in the late 1990's and 2000's, there were fundamental sweeping changes in the academic publishing landscape. The information revolution allowed publishers' costs to fall dramatically, and there was a mood shift in the academic sphere away from subscription based models, towards giving open and free access to some or all of journal article contents.

Simultaneously, university department websites began to post records of their recent publications for adverting and informational purposes, usually with hyperlinks to where the article could be fully accessed.

Publishers still protect journal article content and metadata, in some cases, aggressively, in order to protect their intellectual property. The data they hold on article publishing is immensively valuable and powerful, and as such, publishers are reluctant to give access to their massive metadata holdings. There is a well known saying in Data Science from

Tim O'Reilly, "The Guy with The Most Data Wins" [1]. As such, it is unlikely that academic publishing companies will release their data for analysis by the public in the near future.

1.2 Motivation

The publishing data, when collected and analysed using machine learning and big data techniques, can yield valuable insights into the direction of shifting academic focus, where world leading research is being carried out, and offers the possibility of procedural categorisation of research amongst many other possibilities. Large datasets are almost infinitely mine-able and interesting insights can be found 'around every corner'. For an academic institution like the University of Cambridge, there are two main motivations for mining the data of the publishing landscape:

- Purely Academic insights such as examination of the landscape's features , finding structural information and concealed relationships, in order to better understand how scientific information is spread throughout the world
- Actionable insights, such as identifying where effective collaborations could be set up, prediction of where funding may been allocated, and identification of future areas of academic interests so that internal resources can be intelligently applied

1.3 Aims

There are two main aims for the project, broken down below.

- Data Collection - A scalable, legal and useful programmatic approach to scrape the online publishing landscape to collect as much useful data as possible.
- Data Analysis - Once the data has been collected, Machine learning and statistical techniques should be applied to try and find insights within the data.

The data analysis for such a large, rich dataset is a potentially never ending process. The strategy adopted is to focus on answering a smaller simpler research question first, building a structure to robustly test and answer that hypothesis and then to move on to new research questions. As such, the first research question will be to build a system that can analyse the similarity between scientific authors. This will enable us to find where some authors are publishing qualitatively similar work to each other and to investigate the possibility of them collaborating. The framework upon which this research question is used immediately suggests new, similar questions that can be answered. As such, it will be a primary aim of the project to expose a useful framework for future probing to be facilitated (namely creating a chemistry semantic space where concepts can be quantitatively represented).

From a technical point of view, the aims of the project are as follows:

- Provide documented, maintainable code
- Provide a system that is as automated as possible
- Provide a system that will automatically evolve and improve as time goes on
- Provide a system that will not entropically lose data or suffer data degradation
- Provide a system that produces interpretable, useful outputs

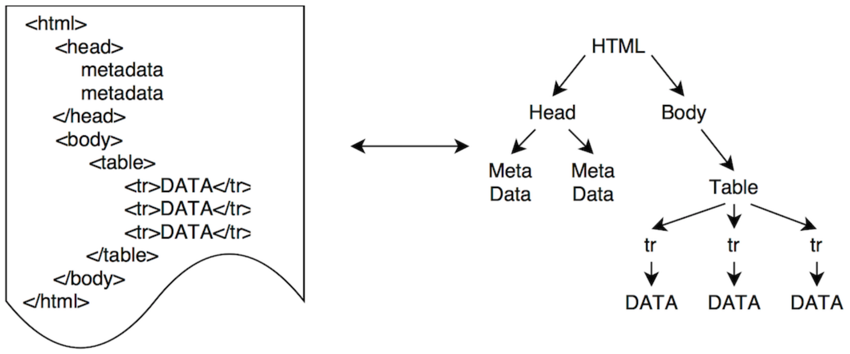
The focus is on ensuring that the result of the project is a system that will continue to be useful after the programmer has left. This will require decent documentation discipline, and systems in place to manage crashing.

2 First Steps

2.1 Generating XPaths

The initial focus was on developing a strategy to scrape records off an unseen webpage. Webpages are written in a tree-like structure using a markup language called HTML, (HyperText Markup Language). A very basic webpage is shown in Figure 1.When a webpage is accessed, the html code is sent over the internet to the user, and the browser e.g. Firefox, interprets it and displays the webpage in a human readable format. When scraping a page for information, the html code itself is scraped. The programmer must write a program that accesses the useful parts of the webpage and stores the required information.

HTML and XPaths



```
<html>
    <head>
        metadata
        metadata
    </head>
    <body>
        <table>
            <tr>DATA</tr>
            <tr>DATA</tr>
            <tr>DATA</tr>
        </table>
    </body>
</html>
```

Figure 1: Tree representation of HTML code

In the above simplified example, it is shown that the html code represents a tree structure, and the useful data is at the bottom of the ‘body’ branch. In order to store the DATA fields, the programmer must provide what is known as the XPath to that node in the tree. The XPath is just a set of directions that the program follows to find the data, and in the example above, the DATA fields are accessed via XPath:

```
//HTML/Body/Table/Tr/*
```

The XPath is a required part of any scraping strategy. The first problem with scraping potentially millions of webpages, is how the XPath (potentially very complicated, and certainly different for every single institution) can be generated. The initial approach was to write a tree analysis suite that would give the XPath of the most 'probable' data entries. This worked by finding the most repeated substructure within the entire tree, and by using heuristics, parameters could be adjusted to automatically generate the XPath to the useful data with decent efficiency. The core of the algorithm is shown below:

```
1. Start at trunk of tree
2. Count the number of descendants of each child of the current node
3. (a) Check if all child notes are all within a threshold of
       similarity to each other
   (b) Check there are more than a required threshold number of child
       nodes
   (c) Check the average similarity between child nodes is above a
       certain threshold
   (d) Check that the proportion of child nodes that are considered
       'similar' is above a certain threshold
4. If all of (a)-(d) are true, this node contains the required data
   and the XPath has been found.  Otherwise, move down to the child
   node with the greatest number of descendants and return to step 2
```

The thresholds mentioned in steps 3.(a)-(d) are adjustable parameters set at reasonable values. This approach was successful for webpages with very large numbers of records on them, as the algorithm searches for the largest, most frequently repeating structural unit in the tree. If there are not vast numbers of records on the page, the method can fail. As such, this strategy is not flexible enough for the required purposes. Despite this it is still useful and still the quickest automated way to generate XPaths for webpages with many hundreds of records on them. A different strategy has been been adopted, which is discussed in detail in section 3.

2.2 CAMSIM and BATHSIM Mini Projects

A significant part of the first few weeks of the project were dedicated to selecting and learning the appropriate technologies to achieve the goal. The CAMSIM and BATHSIM

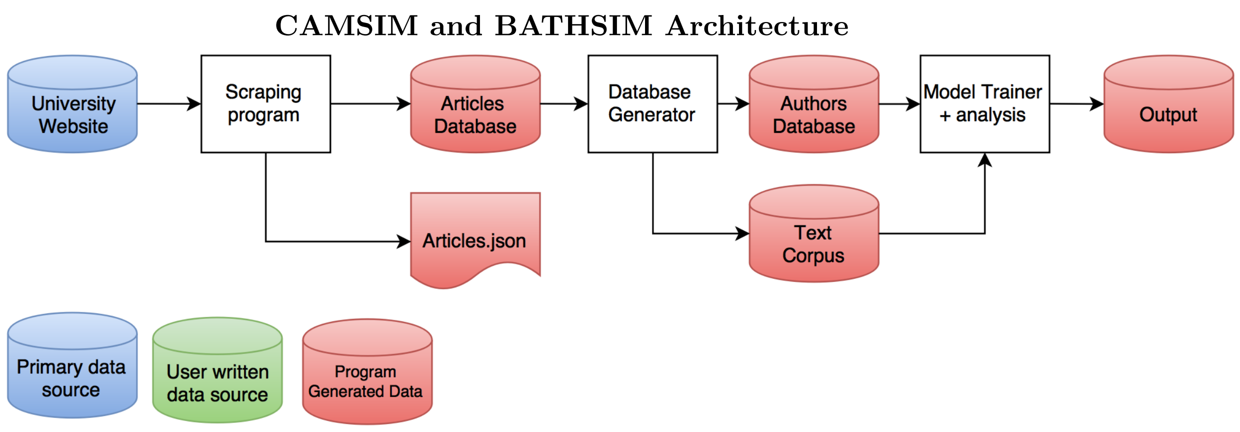


Figure 2: Flow Chart Diagam representing CAMSIM and BATHSIM structure

projects were small proof of concept projects to explore how the different technologies work together, and to prototype a system architecture. The techologies used are:

1. Python 2.7.10 programming language. A modern and flexible interpreted programming language widely adopted by the scientific community, Python 2 has many useful libraries available, optimised for use.
2. MongDB database - MongoDB is becoming one of the most popular databases for data intentsive applications [2]. MongDB is a schema-less noSQL database - it records objects as individual, unstructured 'documents' rather than as rows in tables. This is well suited to the task as complete data is not always available during large scraping operations. There are also well supported python drivers for MongoDB.
3. Anaconda Environment - Anaconda is a python distribution/virtual environment, allowing quick use of external python packages, and fast generation of distributions to deploy on new machines.
4. Scrapy Scraping suite - A well supported end to end scraping framework for Python, allowing fast, scalable web scraping.
5. Gensim - A Natural Language Processing and Machine Learning suite for Python. Their libraries are C optimised and widely used and verified.

BATHSIM and CAMSIM projects use all of the above technologies with an architecture shown in Figure 2. The programatic flow is represented above. BATHSIM and CAMSIM have the same structure, and the only difference is the source data for CAMSIM is provided by University of Chemistry Department at `http://www.ch.cam.ac.uk/publications` whereas BATHSIM's source is the data provided by the Bath University Chemistry department, at `http://opus.bath.ac.uk/view/divisions/dept=5Fchem.html`. A custom written scraping program collects the data which is placed into a database and exported as a human readable .JSON text file. A second program then creates databases of individual authors and a corpus of all the collected titles and abstracts (explained further in section 4.2). As discussed in the section 4, this corpus is used to train a Word2Vec style model that generates a vector representation of an author's published work. The dot product cosines between authors' vectors are computed and outputted as a measure of their similarity.

3 Collection Strategy

The main problem encountered so far is how to scrape large numbers of websites for records without writing tailored time consuming scraping scripts and finding XPaths for every website visited, (approach used in CAMSIM and BATHSIM). The possible solution is using 'regular expression' pattern matching techniques on DOIs. DOIs are explained in detail in section 3.1. The proposed collection strategy, which is nearly complete, is then described in section 3.2.

3.1 DOI

DOIs (document object identifiers) are identifier strings used access journal articles. DOIs can issued by a number of accredited issuing bodies, although the vast majority of scientific articles are issued by CrossRef, a not-for-profit body comprised from Publishers International Linking Association (PILA), a loose association of many academic publishers [3]. A DOI is usually included in a citation or record, and by pre-pending a DOI string with the url stub

```
http://dx.doi.org/
```

, DOI.org will redirect the request to the desired academic journal entry on the publisher's website. All articles published by large established publishers since the adoption of the DOI system will have a DOI registered against them.

DOIs have a somewhat loose structure, and the 'anatomy' of a DOI is shown in figure 3.

Anatomy of a DOI

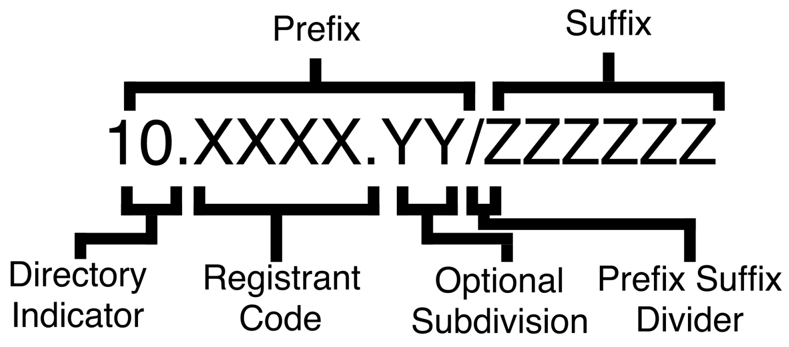


Figure 3: Doi structure. The structure consists of a numeric prefix (X and Y must be integers) and alphanumeric suffix (Z can be any UTF-8 encoded Character)

DOIs consist of 2 sections, the prefix and suffix. The prefix is subdivided into a 'Directory Indicator' (always integer '10') separated from a 'Registrant Code' assigned by the issuing body. Registrant codes are numeric and can be of any length, however they are almost invariably 4 integers long. Registrant codes can have further subdivisions separated by full stops. The suffix is provided by the registrant themselves. It can have any form as long as it is encodable by UTF-8. The DOI specification somewhat ambiguous as there is no vigorous way to terminate a DOI, and on a webpage, the end of a DOI suffix must be interpreted from the context. This provides some scraping difficulties, but can be overcome with careful consideration of outlier cases.

3.2 Cherry Data Collection Program

DOIs can be recognised programmatically from HTML source code through the use of regular expression pattern matching techniques. This suggests a way to collect article data without knowledge of the XPath. The new strategy for creating a widespread scraping with well structured output data is summarised below.

1. `Download page source of university website`
2. `Use pattern matcher to extract DOIs from the source HTML code`
3. `Verify scraped DOIs by using CROSSREF API service`
4. `Deposit valid DOIs in DOI Database`
5. `Use collected DOIs and DOI.org re-routing service to access publisher webpages for articles`
6. `Extract useful, freely available article data: Title, Authors, Date, Abstract, Affiliations`

7. `Store these full records in a database, for use in data analytics`

This process allows widespread scraping as XPaths are only required for the publisher webpages, rather than the university websites. There are much fewer publisher website formats than potentially infinite formats for university websites, and a conversion rate of >90% has already been achieved on a sample dataset of DOIs to full records, from only ∼20 publisher websites XPath sets. The algorithm above is to be implemented by the system architecture shown in figure 4.

Cherry Program Architecture

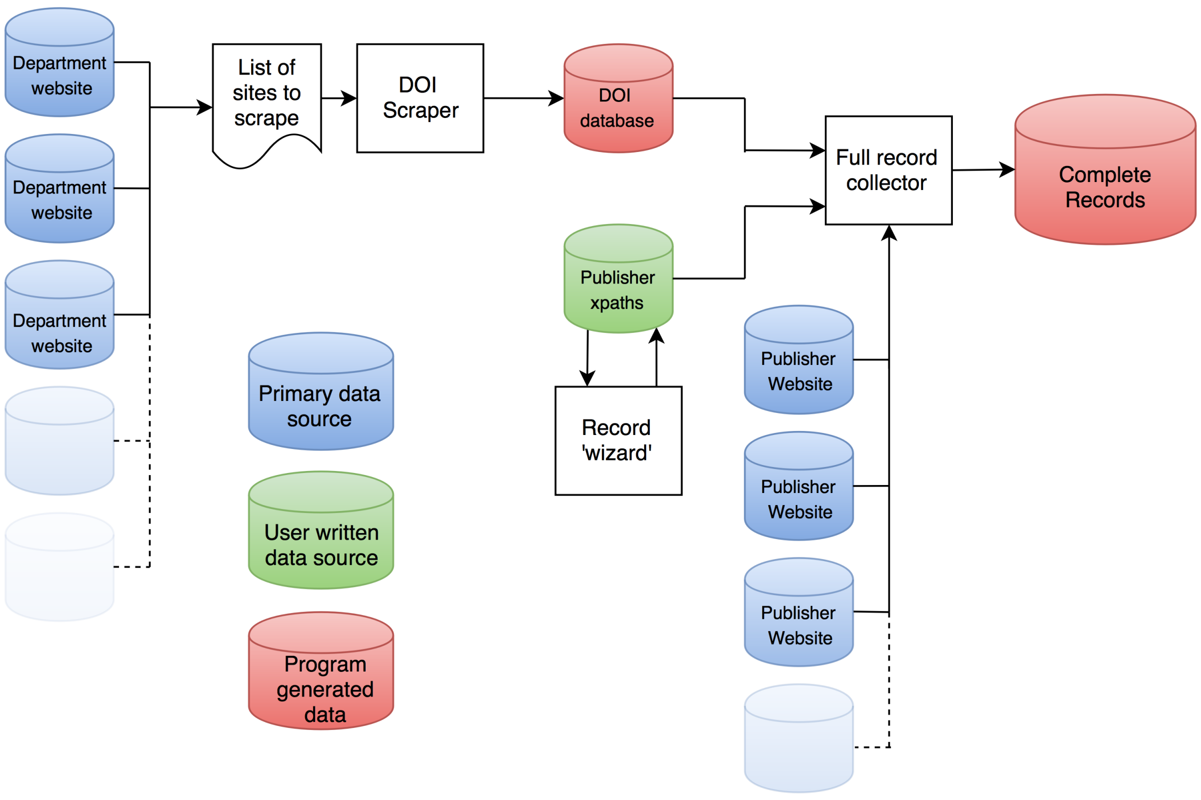


Figure 4: Flow Chart Diagram representing data collection strategy structure

The system has been named Cherry, as the process is reminiscent of picking cherries from an ‘html tree’, and then eating them. The naming convention of fruit runs through into the next sections. The Cherry data collection system is 90% complete at time of writing, with the programs referred to in 4 as ‘DOI SCRAPER’ being complete and ‘Full Record Collector’ being put through thorough testing. The ‘Publisher XPaths’ database is an ongoing project, and can never be ‘complete’. It is the only brittle part of the program and will necessarily need to added to in order to incorporate more publishers and updated when publisher websites change. ‘Record Wizard’ is a proposed program to help write the XPaths and the small normalisation processes required to collect full records for a given publisher website. As an auxiliary program not vital to the system to run, it is of lower priority and will be implemented when the system is up and running.

4 Analysis

4.1 Algorithm

The core aim is to be able to represent research interests quantitatively so that programatic comparisons and analysis can be facilitated. The algorithm that will be used for this purpose is a recently published Natural Language Processing deep learning techique known as Word2Vec[5][6]. The Word2Vec algorithm is a based off 'continuous Skip-Gram' and 'Continuous Bag of Words' architectures. These two learning algorithms are highly optimised, research level Natural language processing algorithms. Word2Vec, after training, takes in individual words and outputs vector representations of those words. The 'Continuous Bag of Words' (henceforth CBOW) learns by attempting to predict the current word based on the word's context (specifically the 2 words preceding and the 2 words following) during training and the 'skip-gram' model learns by attempting to predict the surrounding words given the current word. They could be considered as 1mirror image' techniques of each other.

As mentioned above, the Word2Vec algorithm, once trained on a corpus of many sentences, takes in individual words and converts them into n-dimensional vectors in a language space (n is a user specified value). These vectors encapsulate the syntactic and semantic meaning behind the words. The vectors can be manipulated in surprisingly elaborate ways, and the model can be evaluated by performing vector additions to represent concepts (more on this below). An example is shown in figure 5

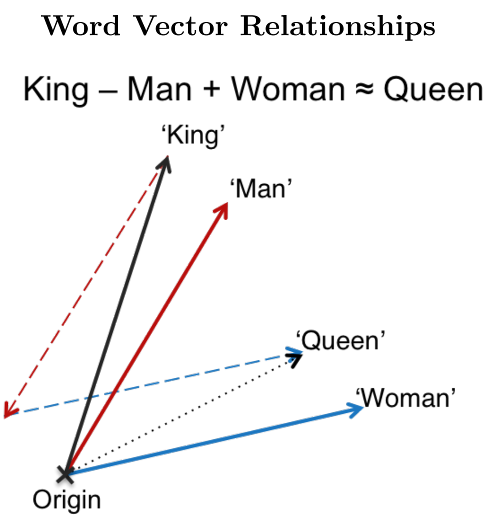


Figure 5: Schematic Representation of how concepts can be represented in word vector space. Word2Vec is able to replicate this behaviour. The vector found by vec('King')-vec('Man')+vec('Woman') is approximately equal to vec('Queen'). The model has been tested on thousands of similar examples[5]

The CBOW and Skipgram algorithms have different strengths but vigorous testing of which will be the best choice for the task cannot be completed until a large set of training and testing data have been amassed. The CBOW appears to be better for syntactic relationships, and Skipgram better for semantic relationships[5]:

- Syntactic relationship: *good* is to *better* as *rough* is to ... (expected answer: 'rougher') [7]
- Semantic relationship: *clothing* is to *shirt* as *dish* is to ... (expected answer: some specific type of dish, e.g.'bowl') [7]

The purpose of the task is to compare semantic chemical concepts, so it may prove that skipgram is a better fit for the purposes, however it will be an interesting question to compare outputs from both models.

The models themselves learn with the following basic method. Sentences of words are assumed to contain words that are semantically and syntactically related. If two words appear in the same sentence, they can assumed to be in some way 'similar'. Word vectors are randomly initialised, and then iteratively optimised, so that weights of word-word connections in the internal network are strengthened if that word pair appear often in sentences together. The procedure is somewhat similar to Neural Network techniques, with only an input and output layer (no non-linear hidden layers). The training complexity is greatly improved over Neural Network techniques.[5].

4.2 Data processing

The 'Cherry' database generated from the process described in section 3.2 contains much free text (the titles and abstracts of the articles). These are already composed of concept linked sentences and can easily be fed in as training data into the Word2Vec algorithms. The more data used in training, the better the model will be. This requires a database with a great many records[1][5].

A training corpus is prepared from the 'Cherry' database in a processing pipeline described in figure 6.

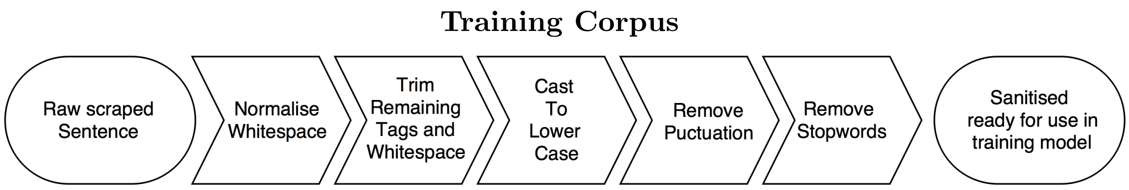


Figure 6: figure showing how data is sanitised and aggregated into the training corpus for training the Word2Vec model

The training corpus is then used to train a Word2Vec model, which in turn can be used to generate vectors to represent e.g. articles, concepts and author research interests (this list is not exhaustive). The analysis suite that will perform these tasks is summarised in the architecture diagram in figure 7.

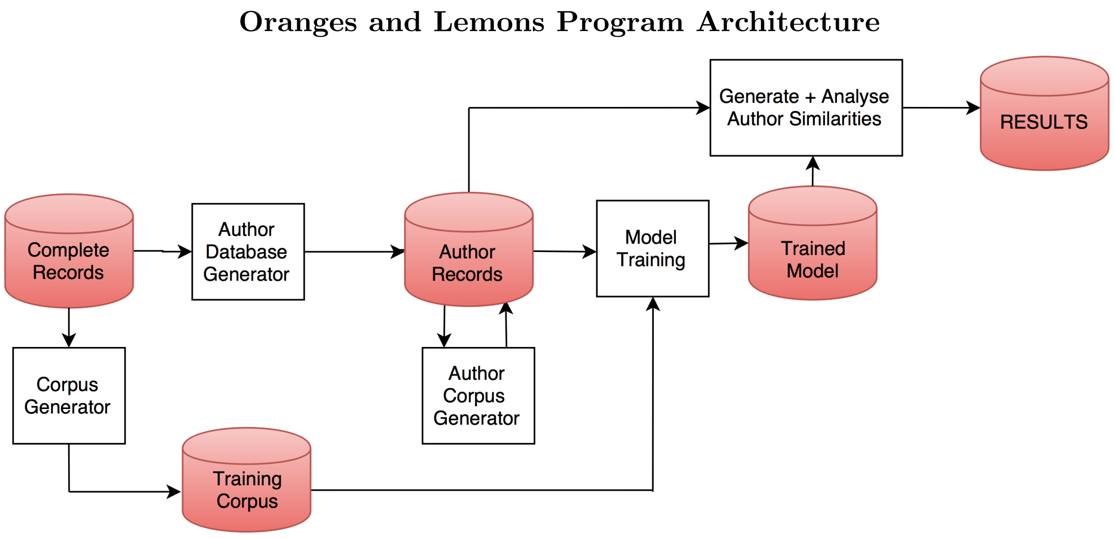


Figure 7: Flow Chart Diagram representing data analysis, training and transformation programmatic structure

Figure 7 includes the necessary programing structure to fulfil the analysis in described in section 4.2.1. The naming convention of fruit was continued from the collection suite described in 3.2. ‘Oranges’ is the name given to the corpus generation and model training processes, whereas ‘Lemons’ is the process required to fulfil the analysis in section 4.2.1. Names for analyses outlined in sections 4.2.2, 4.2.3 and 4.3 will be assigned when the tasks themselves are addressed.

4.2.1 Author similarity pairs

By collecting together articles associated with an author and sanitising the titles and abstracts with the same process descibed in figure 6, we can create a set of words that represent the author’s research interests.Each of these words will each have a vector associated with them, $\mathbf{w_i}$ extracted from the trained Word2Vec model. These can be aggregated to form an average vector of the author’s research interests, $\mathbf{a}$:

$$\mathbf{a} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{w_i} \text{ where N is number of author representation words}$$

The similarity between two authors α and β can measured by finding the cosine similarity between their vectors:

$$\mathbf{a}^{(\alpha)}.\mathbf{a}^{(\beta)} = \sum_{\gamma=1}^{D}\nu_\gamma^{(\alpha)}\nu_\gamma^{(\beta)} = |\mathbf{a}^{(\alpha)}||\mathbf{a}^{(\beta)}|cos(\theta)$$

where the dimensionality of the vector space is D and the components of a vector $\mathbf{a}$ are $(\nu_1, \nu_2 ... \nu_D)^T$. The cosine similarity $cos(\theta)$ is easily computable:

$$cos(\theta) = \frac{\sum_{\gamma=1}^{D}\nu_\gamma^{(\alpha)}\nu_\gamma^{(\beta)}}{\left(\sum_{\gamma=1}^{D}(\nu_\gamma^{(\alpha)})^2\right)^{1/D}\left(\sum_{\gamma=1}^{D}(\nu_\gamma^{(\beta)})^2\right)^{1/D}}$$

which can be well optimised in coding using standard computational libraries. For an author pair, a cosine similarity of $\approx$1 can be interpreted as the two authors having very similar research interests. These high cosine similarities may reveal interesting links between authors currently separated from each other.

This analysis can be achieved programmatically by the structure outlined in figure 7. The ‘Lemon’ analysis process generates a database of authors from the Cherry database described in section 3 and creates a ‘mini corpus’ of representative published words for each author in the database. The analysis suite then computes author vectors and pairwise generates cosine similarities between authors. The process of creating author pairs is not exhaustive as the pairwise comparison complexity will scale with O(n^2). For a set of 1000 authors, to fully explore similarities will require $n(n-1)/2 \approx 500,000$ cosines to be computed. For a very large database, it may be impractical to generate all similarity pairs. Batch processing and intelligent storage of previously computed pairs may however enable thorough searching of the space. The programmatic mechanics of 'Lemons' have been written and a rudimentary form was used in the proof of concepts CAMSIM and BATHSIM (see section 2.2). The models used for vector generation did not have sufficient training data to output useful results. This will change with the completion of 'Cherry' when much more data can be collected.

4.2.2 K-means clustering of research landscapes

K-Means clustering is a categorisation technique in machine learning where k cluster centres are optimised to have least distance from a set of data points. After some dimensional reduction of a set of research interest vectors produced from the 'Cherry' database, k-means clustering could be used to identify research clusters. When the means have been found, new (unseen) articles could be assigned to a topic based on the proximity to the closest cluster centre.

4.2.3 Tracking of Research interests with time resolution

By calculating two sets of k-means cluster centres from two sets of vectors from articles published from two different years, and by tracking how the cluster centres have moved over time, it may be possible to infer how the focus of research has changed over time. As an extension, if the data is of high enough quality, it may be possible to extrapolate where the clusters may move to in the future.

4.3 Network visualisation

Using tools such as Gephi, pairwise author similarity should be able to be visualised as a network graph. This may elucidate non-obvious connections between authors and departments.

5 Next Steps

The next steps suggest themselves naturally. The immediate priority is to finish the 'Cherry' framework and set to work giving it websites to digest. After that has been finished, the focus shifts into analysis, and gluing existing scripts together to get the 'Oranges' and 'Lemons' programs running. The analyses suggested in 4.2.2 4.2.3 and 4.3 are stretch goals of the project, and will probably fall outside the time frame. There is much potential of the method laid out in this document, but so little time to explore it. The portability, distributivity and usability goals alluded to in section 1.3 may have to be completed after the final report submission date, as the final report will require findings, and due to the time constraints, this must be the priority.

References

[1] O'Reilly (2012) *Tim O'Reilly interviewed by Forbes Editor Jon Bruner* Available at:

`https://www.youtube.com/watch?v=wNLhFi3XhIE`

(accessed 24 November 2015)

[2] *JasperSoft Publishes Second Big Data Index* (2012) Available at

`http://www.jaspersoft.com/press/jaspersoft-publishes-second-big-data-index`

(accessed: 27 November 2015)

[3] *The Formation of CrossRef: A Short History* (2009) Available at:

`http://www.crossref.org/08downloads/CrossRef10Years.pdf`

(accessed 25 November 2015)

[4] *The DOI Handbook - Numbering* (2014) Available at:

`https://www.doi.org/doi_handbook/2_Numbering.html#2.2.2`

(accessed 25 November 2015)

[5] T Mikolov, K Chen, G Corrado, J Dean.(2013) *Efficient estimation of word representations in vector space.* ICLR Workshop, Available at:

`http://arxiv.org/pdf/1301.3781v3.pdf`

(accessed 25 November 2015)

[6] T Mikolov, I Sutskever, K Chen, G Corrado, J Dean .(2013) *Distributed Representations of Words and Phrases and their Compositionality* NIPS, Available at:

`http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-`

(accessed 25 November 2015)

[7] T. Mikolov, W.T. Yih, G. Zweig.(2013) *Linguistic Regularities in Continuous Space Word Representations* NAACL HLT, Available at

`http://www.aclweb.org/anthology/N13-1090`

(accessed 25 November 2015)