Final Report

Information Retrieval System for Meeting Minutes and Papers Gathered from Automated Web-scraping

CMT400 - 60 credits

Author: Ivonna Prince

Supervisor: Padraig Corcoran

Moderator: Steven Arthur

Degree: MSc Artificial Intelligence

Institution: School of Computer Science and Informatics, Cardiff University

September 2020

Contents

Abstract

The automated web scraping has been around almost as long as the Internet itself. It is been used to gather public and sometimes private data for information retrieval systems and further analyses. Many such systems are widely used on a daily bases. However, the surface web search engines often do not offer case-specific filtering systems. This project aims to build an entire pipeline of document retrieval, storage, cleaning and querying system to build a fully functional search engine specific for the needs of Audit Wales. It is capable of scraping the modern web from a sample of public body websites and extract and normalise the attributes of interest. It cleans and stores data in cloud services or locally and provides a fast information retrieval system. It is developed to be robust to small web changes, easily expansible and resilient to errors. The search engine has a near real-time response rate and is built to expand user queries to improve recall. Instead of using the Boolean word search approach, the search is carried out on word embeddings.

Acknowledgements

I would like to thank Dr. Padraig Corcoran for the continued guidance throughout this project, the AW Data Analytics team and Stephen Lisle for the support and advice.

Commercial Limitations

There are no commercial limitations on the use of the information contained in this dissertation.

1 Introduction

Audit Wales (AW) [1] is the public spending watchdog for Wales. They hold public bodies to account for their spending and assure the people of Wales that the public money is being spent well. One of the groups of staff is performance auditors. Performance auditors assess whether public bodies have made proper arrangements for securing the economy and are making wise use of public money. To make an assessment, a performance audit is carried out. A performance audit can take up to 30 days of which 30% of the time will be spent on a document review. The documents are not available in one place, auditors have to manually visit many different public body websites. Often, the documents have not been uploaded or are difficult to find. It can take up to a day to find a specific document.

This is a very time-consuming process that could be sped up by creating some tools to help auditors gather the documents in one place. Such a tool should automatically gather relevant documents from many public body websites in one, easily accessible place. It should also keep a copy of the original document to prevent cases when documents get lost during a website migration. To further improve the functionality, a search engine should be built to query the gathered documents. It could bring out the relevant paragraphs of search queries from otherwise 100s of pages long reports. Such tools would reduce the time spent on searching public body websites as well as reduce the time spent on reading through documents that

might not be relevant.

The first 4 chapters of this report try to understand the problem and the need for the products developed throughout this project. Chapter 2 defines the aims and objectives of these tools. Chapter 3 reviews the relevant literature and literature that supports the design and algorithm choices made when carrying out implementation. Chapter 4 aims to describe the background of the originating problem, why is it difficult to solve it, what benefits a successful project would hold, previous work within the organisation to attempt to solve the defined problem and what existing problems are known.

Chapters 5 to 7 are going into software architecture and implementation. Chapter 5 explains the intended approach, contrasts different candidate approaches and explains why others were discarded. Chapter 6 states the functional requirements, design choices and the description of intended implementation. It also breaks the project down into 2 separate tools that can work together in the end. Chapter 7 focuses on describing the more interesting parts of the code and how they work.

Chapters 8 to 10 are trying to assess the success of the project. Chapter 8 analyses the performance of the tools. Chapter 9 summarises the achievements and the deficiencies of the tools and proposes recommendations for future work. Chapter 10 discusses the skills learnt throughout this project, the challenges encountered and the process of handing over the code to AW.

2 Aim and Objectives

The aim of this project is to build a web crawling and information retrieval system that works together. A web crawler that can crawl the public body websites defined later in section 6.1.1 should be implemented. It has to extract the desired attributes that auditors may want to filter by the documents to help them find what they are looking for. Then store the item- an object containing the extracted text and relevant attributes- in some storage space. Saving it in a database can improve search speed and prevent data loss which may occur when public

bodies update their websites. When proposing a solution, the compatibility with Amazon Web Services (AWS) has to be kept in mind as it is the cloud service provider used by AW. In general, using cloud services is recommended. It can prevent data loss when local hardware fails, it can be accessed from anywhere and it is easily scalable.

Currently, at the AW the required documents are manually found and read by the auditors. The goal is to improve auditor productivity by building a solution that reduces the time spent on looking for documents and searching them for relevant information. The solution should acquire all documents in one searchable space as described in more detail in Section 4.4.

The Search Engine system has to return the relevant documents within 2 seconds and order them by date. On top of that, only the relevant paragraphs of the document must be returned. Filtering options to query for specific committee types or document types (agenda, draft minutes, or final minutes) have to be provided. The querying should be done on a database to improve speed by avoiding querying multiple body websites directly. The documents should be acquired regularly and stored beforehand to be ready to be queried. In this way, the information would be fast to find and it would be easy to determine the presence or absence of important documents.

An intuitive front-end for this application should be built. The information should be displayed in a table-like manner, with URL to the original document, date and the relevant paragraphs as shown in Figure 1. However, given the time for this project, the front-end application will be left for future work.

3 Background material

Web scrapers have been around almost as long as the internet. The first web scraper then called World Wide Web Wanderer was built in 1993 [2]. Since then the use of the internet has become a necessity and the ways websites are built today have changed. The book 'Web Scraping With Python' [3] is a good introductory book on how the web crawlers work

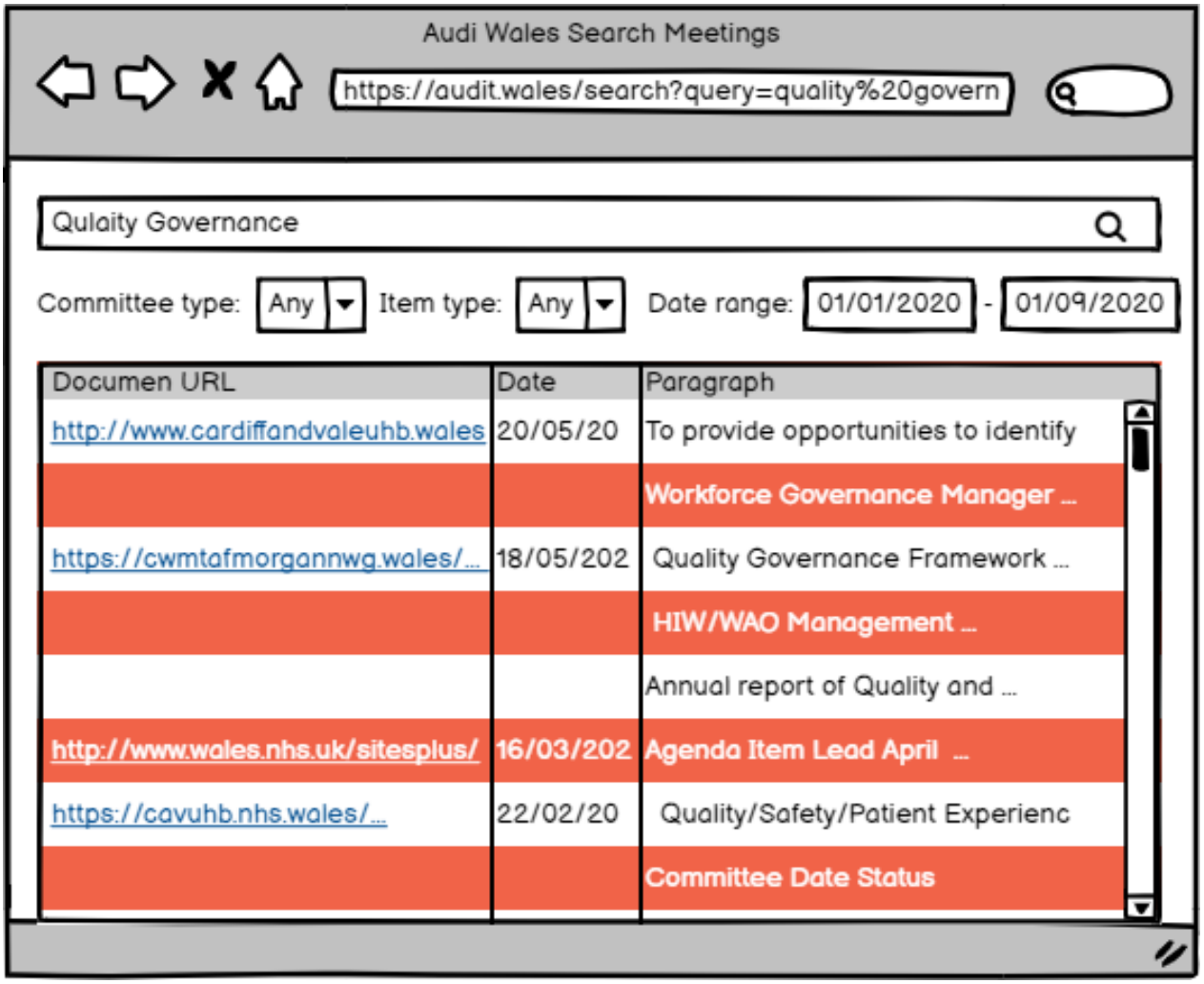


Figure 1: Front-End Mock-Up

and how to build your own for a specific purpose. It explains how to parse HTML, how to use XPaths, regular expressions and lambda functions to extract attributes from the HTML. Then it covers the use of the Scrapy framework for crawling and storing the data. It considers that often different types of documents are encountered such as PDFs and .docx with different document encodings that still can be extracted but may present challenges. Similarly, client-side languages such as JavaScript that make the web dynamic can make web scraping more tricky. For example, the next set of results might be loaded on a scroll event. On top of that, web scrapers might be asked to be throttled or otherwise, the agent might get banned from the website. There are other books, such as 'Python Web Scraping Cookbook' [4] that considers similar issues as well as considers how to make the scraper as a service using ElasticSearch. Many of these issues apply to our project. Specifically, text extraction from documents was found to be tricky. To prevent the scrapers from getting banned, they had to be throttled to 2 requests per minute. To make the scraper compatible with AWS a set of middlewares had to be installed. These and other issues will be described in more detail in section 4.2. Currently, there are no commercial products available specific to the auditor's needs.

The book 'Introduction to Information Retrieval' [5] covers a multitude of challenges associated with information retrieval (IR). It covers building indexes and vocabularies, Boolean searches, ranked searches, evaluation of IR, query expansion, web search basics and other topics. It concludes that ranked unigram Boolean search models in most uses cases are a good solution. It highlights the importance of having a good indexing method to achieve fast response time.

When building a ranked IR system, the weighting algorithm has high importance. Many books and papers such as 'The Probabilistic Relevance Framework: BM25 and Beyond' (S Robertson, H Zaragoza, 2009), 'Field-Weighted XML Retrieval Based on BM25' (MacFarlane, A., Lu, W. and Robertson, S. E., 2006), 'A simple approach to optimize XML Retrieval' (T.Wichaiwong and C.Jaruskulchai, 2010) consider the probabilistic model of Okapi BM25 that have been around since the 1970's. This model is sensitive to term frequency and

document length which is important when searching public body documents. These papers evaluate the BM25 model and offer other versions of it such as BM25F and BM25E that improve performance in the right context. Thus, when choosing a ranking algorithm BM25 was chosen. It proved to return better results than Universal Sentence Encoder (USE) due to the consideration of document length.

In the context of e-commerce the paper 'When users don't say what they mean: BM25 vs Deep Learning for product search' (2017) and Manning et al. (2008) highlights the importance of query expansion. Often the user does not include the exact words they are searching for. Expanding queries to improve results is necessary. The feedback on the proposed search approaches confirmed that the best approach was the one using query expansion.

The Crawling system that will be introduced will be uniquely tailored for the websites that AW is interested in. It will use word embeddings to help identify and normalise desired attribute fields. To reduce the time needed to build and maintain the indexing method, ElasticSearch will be used for the IR system. To improve the results, query expansion via cosine similarity will be used. This will be achieved by using word vectors stored in ES. This approach was decided upon reviewing the relevant literature and taking into account technology widely used at the moment.

4 Problem

Performance audit is a term used to encompass a range of different functions carried out by public audit bodies, such as Audit Wales (AW). Performance auditors assess whether public bodies have made proper arrangements for securing economy, efficiency and effectiveness in their use of resources. AW has the power to make recommendations if they determine that there are opportunities to improve the current arrangements they audit. For example, one such public body could be Cardiff University. Cardiff University receives large amounts of grants and funding for research and therefore it is likely to be audited to determine whether the funds are being spent efficiently and effectively.

As part of these assessments, audit staff will undertake a range of activities including document reviews. Document reviews can be very labour intensive. A typical performance audit lasts around 25 to 30 days, of which 7 to 9 days can be spent searching for, retrieving and reading multiple documents from the websites of each public body (subject to audit). Some documents are difficult to find on the public body's website due to the lack of consistency in web-page structure, poorly designed and badly documented websites. Due to these inconsistencies, auditors indicated that it can sometimes take more than a day to find a specific document. It is even more challenging when most of these websites have poor quality search engines on the public body's website and user interfaces that are not intuitive and difficult to navigate.

Audit staff will attempt to search for these documents using specific keywords. Not all audit staff have experience in document searches or the training to fully utilise the functionality of popular search engines. This means that the website search engines might be able to return the information the auditor is looking for but they do not have the knowledge of how to use it to its full potential.

A risk assessment report is produced as a result of an audit. There are 5 thematic areas that are covered in these reports; Well Led & Well Governed, Strategic Planning, Use of Financial Resources, Workforce Management and Performance. These areas cover a range of sub-topics, for example, 'well led and well governed' cover such topics as:

- Board and committee effectiveness
  - Quoracy
  - Decision logs
  - Public meetings
  - Terms of reference
  - Standing orders
- Risk management

  - Risk strategy
  - Board assurance framework
  - Risk register
  - Risk appetite
  - Risk management policies and procedures
- Board assurance
- Performance management
- Quality governance
  - Clinical audit
  - Complaints and incidents (patients, staff or visitors)
  - Patient experience
  - Patient outcomes
  - Health and care standards
  - Patient and staff stories
  - Quality improvement
- Management information
- Reporting and scrutiny
- Organisational structures

For the full list of topics for each theme see Figure 27 in the Appendix.

To reduce the scope of this project, it was decided to focus on one area, specifically on Quality Governance.

This project is broken down into two parts: 1) data acquisition and 2) data analysis. The goal is to offer a web scraping system and an information retrieval system that is quicker and less prone to error than manual reading and searching relevant documents. It aims to improve the user's productivity and help to inform audit judgements and recommendations.

4.1 Previous Work Within the Organisation

The organisation has built two applications with a similar purpose already. The Assembly Watch application scrapes papers, using a web crawler, from the Senedd Cymru website and provides some automated analysis of the data. The application's back end uses Python and Selenium, whilst the front-end is implemented with R Shiny. However, the web scrape is limited to documents found on the Senedd Cymru Records website and is only performed after the user selects a date range. This results in several issues. Firstly, the crawl is terminated early if the user refreshes the page, limiting the robustness and user experience of the application. Secondly, the documents crawled are not stored in a database for repeated access. As a consequence, choosing a date range that has been selected previously will require another crawl, which is an inefficient use of time and processing. Finally, the documents need to be downloaded before analysis can be performed, which can leave the user waiting quite a long time depending on the date range. For example, a 6 month search range can take up to 20 minutes to be collected. In order to improve the performance, a proposed solution that can scrape multiple websites regularly, extract relevant metadata and store the retrieved information in a database will be introduced. This will remove the need for redundant crawls and speed up the information retrieval greatly.

The second application is a Document Search Engine built in R Shiny. The app performs a keyword search in the corpus of documents that are stored on the user's local machine. The app extracts exact term and phrase matches and returns all the sentences where the search terms appear. The limitations of this app include a lack of robustness to spelling errors, absence of sentence scoring to determine the best matching results (not a ranked IR system) and no consideration for the semantic meaning of the search phrase. Proposed is a ranked

IR system that considers synonymous words as search terms. Additionally, it is resilient to language differences (e.g. 'color' and 'colour' will be treated as matching) and spelling errors.

4.2 Known Problems with Data Acquisition

Websites are continuously changing, and thus scraping them automatically can be challenging. Web scrapers that can follow all links within a given domain and crawl the entire website are easy to build, but it would be time and resource inefficient. AW is only interested in specific committee papers. To reduce the crawling space, rules that the crawler can interpret and obey can be defined. To define accurate rules to determine which papers the user is interested in, the website must have a consistent structure. Unfortunately, in most cases, there is no consistency, which makes it challenging to find rules that are not too specific nor too generic. More specific rules are likely to stop working upon small website changes, and the more generic rules are likely to scrape irrelevant documents.

The papers of interest of AW are available in either PDF or Microsoft Word formats. PDF is a reliable format that will look the same to all users independently of their machine's operating system, browser etc. PDF documents consist of a stream of instructions describing how to 'draw' on a page, with even text being rendered like an image. This can make extracting text and paragraphs difficult since this content isn't clear from the document format and all the content semantics are lost [6]. To help understand how the machine scrapes a PDF document, it is useful to view the scraped format of a .pdf file. This can be done by opening a .pdf document, selecting all the content and copy-pasting it into a blank Word document. The result will be a chunk of text without any tables, images, or text formatting and often with no paragraphs or line breaks either.

It is possible to extract tables to preserve the added value of matching row and column names, however, the current table extraction packages are slow and still are unable to recognise merged cells and column sub-headers.

Another problem is that documents have no meaningful mark-up, i.e. there is no mark-up equivalent to HTML meta tags such as *property="og:datePublished"*. Whilst for a human it is easy to look at the document and determine when the meeting was held, for a machine it is difficult without a specific tag or a rule to look for. If the paper mentions multiple dates, there would be no mark-up tag to determine which is the correct date of when the meeting was held. This means that often the attributes for the documents that AW is interested in have to be extracted from the website itself. Reasonably reliable data can be gathered to later determine what committee type the document is or what date the meeting was held.

4.3 Known Problems with Data Analysis

To build a good quality Natural Language Processing (NLP) search engine is difficult. To achieve good results a large, clean, noise free and labelled training corpus is needed. When the data is being extracted from documents, it will be messy and noisy. Under the current circumstances, the data also is unlabelled. Cleaning the scraped data and reconstructing paragraphs is a challenge within itself that remains an open research area.

Scalability and quick response time also have to be considered. There will be more meetings held and more data added continuously for the foreseeable future. This means that our IR engine has to be indexed correctly or use an appropriate solution that will stay fast despite the increase in data.

On top of that, the semantic meaning of what the user really is looking for must be considered despite none of the search words nor their synonyms appearing in the search query. For example, when looking for *'quality governance'*, the expected output should cover clinical audit, staff and patient complaints and incidents, patient experience and outcomes. However, none of the two search words appear in any of the subtopics of interest.

Another known challenge is different spelling and variations of the same word. Different people might spell certain words in a different way, but it does not change the meaning of the word and therefore should not be treated as a different search term. To give a few examples

British English	American English
analyse	analyze
neighbour	neighbor
travelled	traveled

Table 1: Different Spelling in British and American English

of how British spelling differs from American spelling, consider words ending with 'yse'. They are spelled with 'yze' in the American English; British English words ending with 'our' are spelled with just 'or' ; verbs ending with a vowel and 'l' are doubled in the past tense whilst in American English it is not [7], see Table 1.

Understanding has to be formed of how the user intends to interact with the tool before implementing the final solution depending on whether the user is more likely to search for exact word matches or phrases. For example, if the user is likely to look for terms, it is beneficial to remove stop words that hold no valuable meaning in the returned results. Stop words are extremely common words that hold very little value. Such words are 'a', 'the', 'but', 'is' etc. However, if they are more likely to search for phrases, stop words can increase accuracy and return more meaningful documents. A good example is given by Manning et al. (2008) [5]. They considered the search phrase 'President of the United States'. If stop words are removed and search results matching 'President' and 'United States' are returned, the results will have higher recall but low precision.

Furthermore, a decision has to be made on how to pre-process the scraped documents. When training machine learning models, all punctuation is often removed as it usually improves the performance. However, in IR systems the punctuation might add meaningful value. For example, IP addresses, email addresses, website URLs and specific terms, such as programming languages (C#, C++) would lose their meaning if all punctuation were removed. Particularly relevant to our case is hyphenation. If phrases 'Co-located health and social care teams' and 'The funding is a loan and non-recurring money' are considered, depending how we decide to pre-process the hyphenated word, different tokens are extracted: 'co-located', 'colocated'

or ‘co located’, ‘non-recurring’, ‘nonrecurring’ or ‘non recurring’ which then can lead to good or bad matches depending on which approach is chosen. Here, it seems that leaving the hyphen or joining the hyphenated words together is necessary but if a hyphenated word saying is encountered such as ‘follow-the-leader practice’, it seems that the hyphens should be disregard.

Finally, how duplicate documents are being ranked has to be considered. Often, very similar if not identical documents will be uploaded. When a user sees the information for the first time, it holds high value, but none if they have to view the same document again. It is tricky as duplicate documents can have different end points making it seem like unique documents to the machine. Further, in the context of meetings, if both final and draft minutes have been uploaded, they are likely to have very similar content. However, the documents of interest would only be the final minutes in such a case.

4.4 Benefits

There are a number of benefits that a successful project would hold. Firstly, all the documents from multiple bodies would be kept in one, easily accessible and searchable space. This would greatly reduce the time when the document content would need to be retrieved for metadata aggregation or query searches. It would free up the auditors time to do other useful things and improve their productivity. Secondly, it prevents data loss. Having a local copy ensures that users do not have to rely on third party websites to keep the information public and available. No more time will be spent on searching for documents that once existed but are no longer available. Finally, an IR system that is suited to the auditors needs can be tailored. They have full control over it which allows further research and improvements. The data can be used for other purposes to retrieve statistics on metadata, such as ‘How many meetings a year has a specific committee held?’; ‘What is the longest amount of time between the date of the meeting and last modified for a specific committee?’

5 Approach

This section explains the intended approach to addressing the problem and contrasts different candidate approaches.

5.1 Approaches considered

5.1.1 Programming Language

To develop the project, Python was chosen as a programming language as it is easy to use and it has many already built-in libraries and good documentation.

5.1.2 Web-scraping Frameworks

In Python, there are 3 well-known scraping libraries available: Scarpy, Beautiful Soup and Selenium. Scrapy is asynchronous, has built-in XML and CSS extraction tools, supports all OS, is easily extensible via pipelines and middlewares and has good community support. The main advantage of Beautiful Soup is that it is easy to learn, but it is slower than Scrapy and needs external libraries to work. Selenium is primarily designed to test web applications and therefore, can load JavaScript and Ajax produced HTML which Scrapy on its own cannot. The main disadvantage is that Selenium is slow. The websites that have to be scraped for this project do not contain any links loaded by JavaScript, thus Scrapy framework was chosen to implement the web crawler.

5.1.3 PDF Extraction

To extract PDFs, Apache Tika was chosen. It provides Tika REST Server that can be easily integrated into the crawling project and allows to extract text from a byte stream. Alternatives would be `PyMuPDF` and `PyPDF2`. All 3 text extractors produced output that had 99.95% similarity to text that would be copy-pasted from the original document. However, the other two libraries could only read in a local document and could not be integrated into

the crawlers pipeline. The only disadvantage is that for Tika to run, Java 7+ has to be installed on a machine.

5.1.4 DOCX Extraction

To extract DOCX documents `Zipfile` and `XML` libraries were used. Similarly to PDF extraction, the already existing libraries such as `python-docx`, `docx2txt` and `docx2python` could not be integrated within a byte stream. As Word documents are XML formats, thus this library was able to extract the text.

5.1.5 Committee and Item Type Labelling Approaches

When crawling the web, to determine committee type and item type, different HTML fields that potentially contain the desired information have to be recorded. These fields are often fuzzy due to inconsistencies through websites. Sometimes, the committees are named differently across different websites too. An example can be seen in Figure 2. To solve this problem I tested 3 approaches: hard coding, fuzzy set matching and using word embeddings. Hard coding all possible mentions and spellings of committee types is very tedious and would require continuous maintenance. Fuzzy set matching using token_set_ratio from FuzzyWuzzy package calculates standard Levenshtein distance similarity ratio between two sequences [8]. Given the phrase, 'mental health and learning disabilities committee', the package will correctly label the string as 'mental health act committee' with 92% confidence. It tokenises, pre-processes and then alphabetically sorts tokens and measures pairwise similarity between the intersection of both strings and intersection + first string and intersection + second string. The higher the intersection, the higher the score. In this way, there is no need to identify all possible committee spellings as keywords usually stay the same across all body websites. However, it falls short when synonym words have been used. For example, the string 'approved minutes' are equally likely to be assigned labels 'final minutes' and 'draft minutes' as seen in Figure 3a. Using word embeddings can solve this issue. Similar meaning words will have similar vectors and therefore, can measure word vector similarity by using spaCy and

Labels	Aneurin Bevan	Betsi Cadwaladr	Cardiff and Vale	Cwm Taf Morgannwg	Hywel Dda	Powys	Swansea Bay
Audit Committees	Audit Committee	Audit Committee	Audit Committee	Audit and Risk Committee	Audit and Risk Assurance Committee	Audit Risk and Assurance Committee	Audit Committee Papers
Digital and Information Management Committees	Information Governance Committee	Digital and Information Governance Committee	Digital Health Intelligence Committee	Digital & Data Committee		Information Management, Technology and Governance Committee	

Figure 2: Different Namings of the Same Committee Across Bodies

```
>>> fuzz.token_set_ratio('final minutes', 'approved minutes')
70
>>> fuzz.token_set_ratio('draft minutes', 'approved minutes')
70
```

(a) FuzzyWuzzy Package

```
>>> nlp('final minutes').similarity(nlp('approved minutes'))
0.8210549793958809
>>> nlp('draft minutes').similarity(nlp('approved minutes'))
0.7931806897074939
```

(b) FastText Wikipedia Word Embeddings

FastText pre-trained word vectors, achieving the results shown in Figure 3b. It shows that "approved minutes" are more similar to "final minutes" than "draft minutes".

5.1.6 Date Extraction

For date extraction, a regex expression was written as it gives full control of the formats that are expected to be extracted. An already existing fuzzy date extractor from `dateutil` package exists that could not perform well in our context. For strings, such as '> 2015/16 Workforce & Organisational Development Committee Agenda - 30 June 2015', where more than one potential date format appears, the package throws an error shown in Appendix 22. In some cases it also does not recognise the correct date format (as shown in Figure 23 in the Appendix).

5.1.7 Data Storage

Elastic Search (ES) was chosen for our data storage for the IR System. Elastic Search is built on Lucene. It is faster than MySQL database and other relational databases when the corpus is very large as ES queries can operate in-memory [9]. It is designed for time-sensitive full-text search and can offer near real-time response [10]. It is distributed allowing to handle petabytes of data easily. On top of that, it has built-in a large amount of functionality to support data analyses. The main reasons for choosing ES is that as a NoSQL database, it has a dynamic schema that can handle unstructured data, it is resilient, fast and scalable as well as supported by AWS. A SQL database would not have been a good choice for our use

case mainly due to fixed schema, lack of support for unstructured data and scalability since the space requirements are unknown for this task. It also does not have a built-in ranked querying system that ES has.

For the Search Engine algorithm, 3 approaches were considered:

1. Okapi BM25
2. Okapi BM25 with added synonym queries
3. Universal Sentence Encoder (USE)

BM25 is a ranking algorithm that considers TF-IDF and document length when scoring document relevance. It is the default search algorithm in ES. Whilst this algorithm is known to perform well on search queries, it only matches the exact words and it is not considering synonym words. To improve the algorithm, query expansion with synonym words was introduced. This approach is still not resilient to spelling errors nor it is looking for the semantic meaning of search query. Rather than matching each individual term, USE is proposing a different way of measuring similarity by looking at the semantic meaning of the entire sentence. USE capabilities can be demonstrated by looking at the most well-known example, the sentence 'How old are you?'. If we look for similar sentences, the highest-scoring sentence after an exact match, will be the sentence 'What is your age?'. None of the exact words appear in both sentences, yet they have very high similarity.

5.2 Evaluation

As there is no labelled data to help evaluate the approaches, a Performance Auditor was invited to label the returned documents on some sample queries as relevant or irrelevant. They were given an Excel spreadsheet that contained the URLs to the top 10 documents returned for 3 different approaches run on the 3 given queries along with the extracted relevant paragraph of that document. The sample queries were 'Fraud', 'Quality Governance' and 'Patient Incident' and they were run on a corpus of little over 3200 documents that were obtained from a sample scrape from the 7 selected websites. As the queries are looking

for the most relevant paragraphs, it sometimes would return multiple paragraphs for the same URL. The auditor then visited the given URLs to determine if the documents are relevant to the queries and labelled them as relevant or irrelevant. The fully labelled data can be viewed in the Appendix 28, 29 and 30. As the corpus also was unlabelled, recall could not be measured. Recall is the fraction of relevant documents retrieved. As it is not known how many documents of the corpus are relevant to the queries, a score cannot be calculated. Similarly, accuracy also cannot be calculated as it is not known which documents were relevant and were not retrieved and which documents were not relevant and were not retrieved. Instead, Table 3 shows the unique URL count. More unique URLs indicate a larger variety of documents. The chosen approach was based on precision and the expert's opinion.

The auditor was also provided with a set of paragraphs for each approach run on queries of their choice, such as 'Staff and Patient Complaints' run on 2 specific documents. Then they would read through the paragraphs returned and would form an opinion of which paragraph set matches the closest to their expectations. This process is entirely subjective and cannot be measured by a score of success but it helped them to choose an approach as discussed later.

To measure precision, the duplicated URLs of the now labelled data were removed and the following formula was used

$$Precision = \frac{relevant\_unique\_docs}{all\_unique\_docs}$$

The results can be seen in Table 2 where blue highlights the highest precision score for the query. Precision is the fraction of documents that are relevant. The maximum obtainable score is 1 which would indicate that all documents that were returned are relevant. The highest precision on average of the 3 queries was 0.8 which was obtained for USE. The highest average unique document count of 11 was returned for BM25. However, the data shows that the query has returned more results than originally requested, which was fixed in the final implementation.

Despite the result suggesting that USE approach is the most accurate, the expert decided that the BM25 with added synonym queries was the most suited for their needs. The data provided to the expert also included the relevant paragraphs which suggest that despite document relevance, the paragraphs returned for this approach were more informative than the other approaches. One noticeable thing for results of USE, was the sentence length. When using USE approach statistically more relevant documents were returned, but the returned paragraphs were often no longer than a word and too short to be informative. Hence, the BM25 approach with added synonym queries was chosen.

Query	**BM25**	**BM25 + Synonyms**	**USE**
Fraud	0.5	0.7	1
Quality Governance	0.636	0.636	0.9
Patient Incident	0.9	0.5	0.5

Table 2: Accuracy of Queries

Query	**BM25**	**BM25 + Synonyms**	**USE**
Fraud	12	10	6
Quality Governance	11	11	10
Patient Incident	10	10	6

Table 3: Unique Document Count

6 Application of the chosen approach

In this section the requirements and high-level design choices will be discussed. The implementation will be discussed later in Section 7.1.

6.1 Scrapy Crawler for Automated Data Acquisition

To implement a good web crawler the main functional requirements must be determined. The requirements are as follows:

1. Compatibility with AWS. Currently, AW is extensively using Amazon Web Services, thus the proposed implantation will have to be compatible and easy to deploy on AWS.
2. Expansible. To keep the project small, the software must be able to successfully crawl

the 7 given websites. Additionally, the software has to be easy to expand to crawl many more websites.

3. Metadata extraction. The crawler must be able to extract and record predetermined useful attributes which should also be easy to expand upon change of requirements. The required attributes upon completion of this project include public bodies website name, document url, document content, date of the meeting, date the document was last modified, committee type and document type.

4. PDF and DOCX extraction. The crawler has to be able to extract text from .pdf and .docx extension documents.

5. Saving documents. The software must be able to save documents locally or in an external storage unit (such as a database or S3 bucket). It should have a data structure easy to change to suit any desired format for saving.

6. Persistence. The software has to have an option to keep track of websites crawled to rule out redundant crawling.

Currently the auditors can spend up to 30% of their time on looking for the desired documents according to the Performance Auditor. As determined earlier in section 4.4, crawling the web regularly and storing all desired results in one searchable space would hold a number of benefits and would increase their productivity.

It is known that the auditors have a private website to access more documents, but this part will be left for future work due to access limitations. The proposed solution can be used to authenticate using Scrapy framework directly or by incorporating Selenium, thus allowing to crawl private websites too.

Now, that the need and benefits for a web crawling system are specified, a design of such a system can be specified.

6.1.1 Writing Spiders

Firstly, the focus will be on writing spiders. A spider is "a program or automated script which browses the World Wide Web in a methodical, automated manner" [11]. As each website has a different architecture, a separate spider for each website will be written.

The websites of the 7 local health boards in Wales will be crawled, namely, Aneurin Bevan, Betsi Cadwaladr, Cardiff and Vale, Cwm Taf Morgannwg, Hywel Dda, Powys and Swansea Bay.

The general structure of a simple spider can be seen in Figure 4. A Start URL is provided which will be the top-level starting point of a spider. Then, based on XPaths, the spider is allowed (or denied) to follow the URLs encountered on that page. A callback function is specified which can be either called recursively or will call another function, depending on the complexity of the crawl. Similarly, in the callback function XPaths that the spider is allowed to follow until the goal document is reached are specified. The intermediate functions are needed to extract useful information from the HTML of pages encountered before the document. In most websites, the type of committee, item type or date, will be mentioned in one of the previous pages. With these functions, the according fields can be extracted and the item with extra attributes can be yielded. After the final document has been reached, the document object is yielded to then be further processed in the pipelines.

The depth refers to how many sub-requests the spider is allowed to make before terminating a thread. This is an attribute that can be specified in the settings file to prevent spiders from wandering off from the relevant documents.

6.1.2 Writing pipelines

An item pipeline in this context is a "Python class that implements a simple method". It receives the item and processes it as well as decides whether the item should be dropped and no longer processed [12].

Some spider specific pipelines have been written, to extract helper attributes that help in

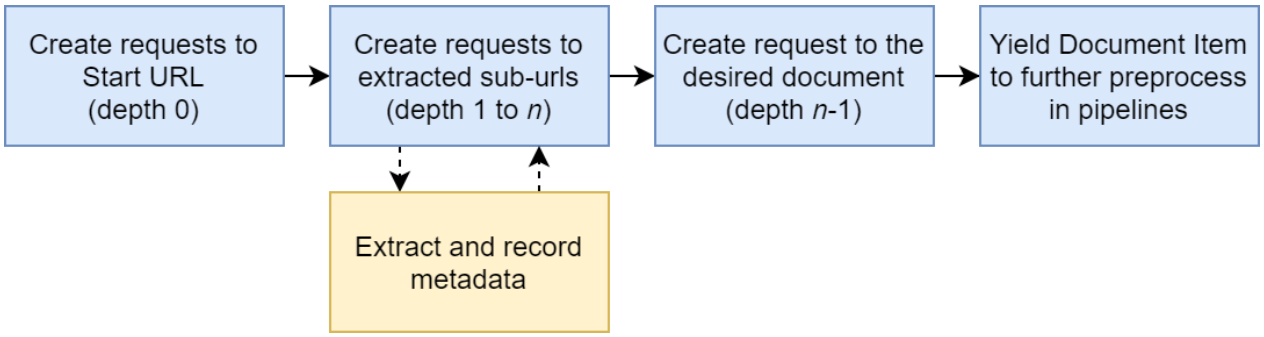


Figure 4: General Spider Design

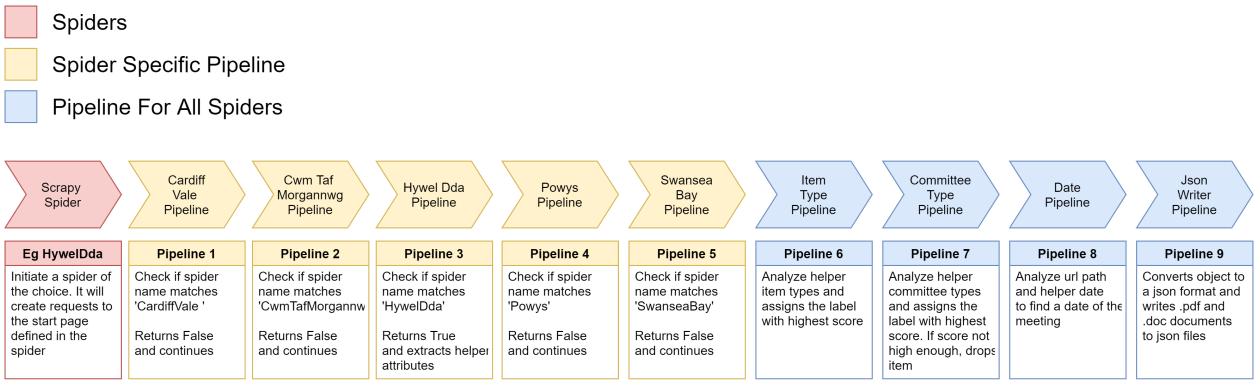


Figure 5: Pipeline of the Web Crawler

the later pipelines to extract attributes that are mandatory. Other pipelines apply to all spiders, namely the Item Type pipeline, Committee Type pipeline, Date pipeline and Write pipeline.

The entire crawler project architecture with a follow through example can be seen in Figure 5. Upon invocation of a selected spider, first, it will go through the process shown in Figure 4, then it will go through each defined pipeline shown in Figure 5. The pipelines that are highlighted in yellow, only process the specific site spider. The pipelines highlighted in blue, affect all spiders as they are extracting and recording required metadata.

6.1.3 Storing data

As a proof of concept, it is enough to store data locally. To consider a centralised solution compatible with AWS two packages were found. To store data in an Amazon S3 bucket or

directly in ES, the user can install `scrapy-s3pipeline` or `ScrapyElasticSearch` packages and configure the settings in the settings file. Before inserting the data in ES, data needs to be cleaned and pre-processed. This will be done in the second part of the project, thus leaving the document insertion in the ES for later.

6.2 ElasticSearch as a Search Engine

The functional requirements for an IR system are as follows:

1. Compatibility with AWS. As mentioned earlier, the solution has to be able to run with AWS.
2. Speed. The response time has to be fast.
3. Expansibility. The search queries have to cover synonym words.
4. Filter attributes. The query can be modified to be filtered for date ranges, specific committee and item types.
5. Order. The search engine must order the returned top n results by most recent first.
6. Relevance. The returned results must only contain the relevant paragraphs to the query of the document.

To keep the project short, front-end implementation will be left for future work. The main focus is on returning a JSON object with all the required attributes and in the correct order.

The process of implementing IR System is broken down into three parts. Firstly, the data acquired from crawls is pre-processed separately and independently from the actual Search Engine. This means, after each crawl, the documents have to be inserted in ES. This is a step that could be implemented within the Scrapy pipeline, but to keep the project loosely coupled, this stage will be carried out independently at the time being.

Secondly, a vocabulary of n-grams has to be built. The vocabulary will act as a synonym

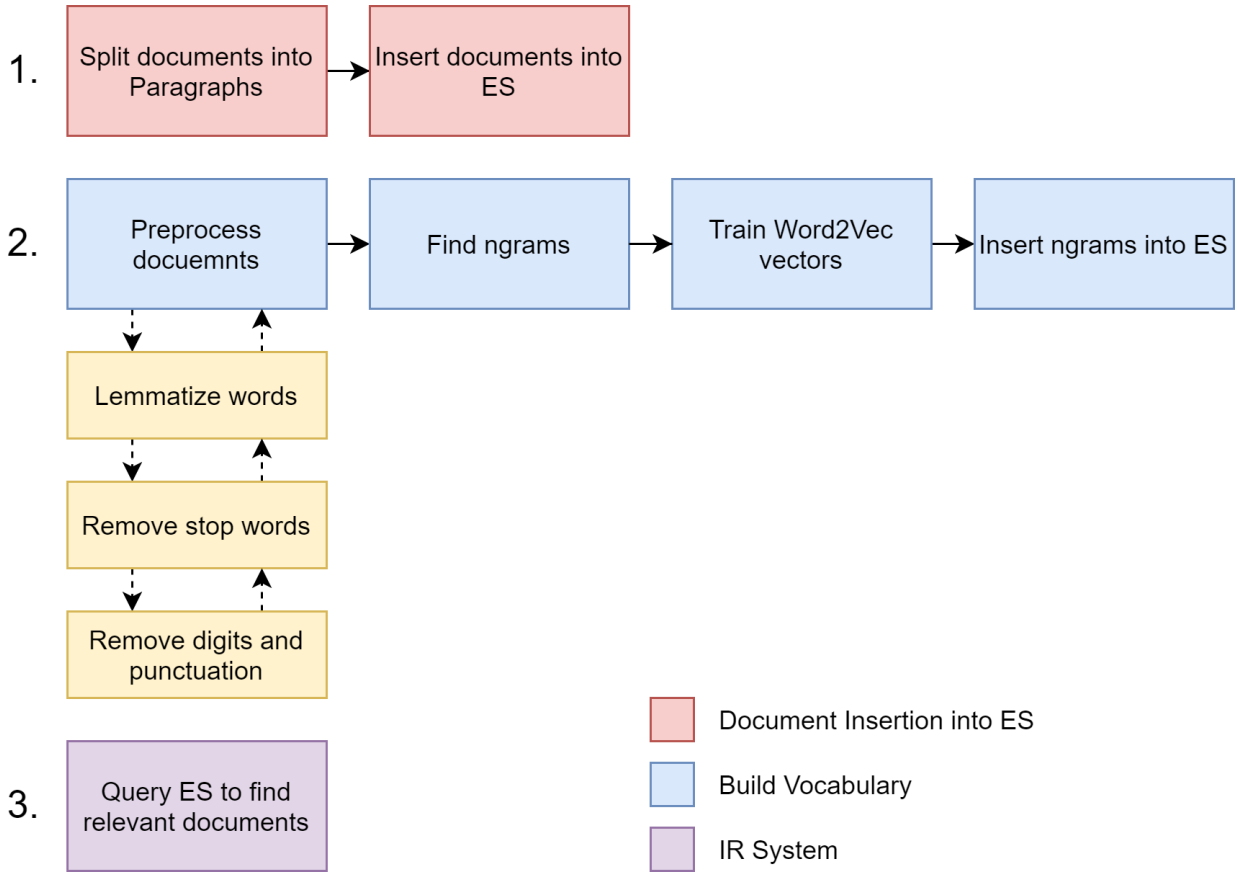


Figure 6: Three Stages of Building Information Retrieval System

dictionary for search queries. To measure the similarity between words, Word2Vec vectors will be used. Word vectors group similar words close to each other in a vector space, making it easy to measure the similarity between words. To create the dictionary, a large corpus of data is needed to train phrases (to find bigrams and trigrams) and to train Word2Vec representations of these grams. This step is time-consuming and not necessary to be carried out every time a new document is crawled. However, it would be beneficial to update the dictionary once in a while to ensure the newest terms are being added.

Finally, the implementation of the actual Search Engine is built. This will be a query to search for the documents stored in ES. A general design diagram of this process can be seen in Figure 6.

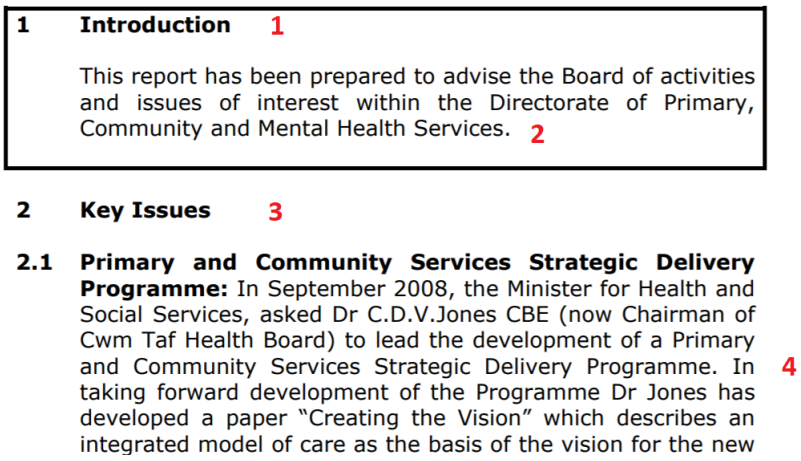


Figure 7: Sections Recognised as Paragraphs

6.2.1 Document insertion into ES

The main challenge in this step is to split paragraphs correctly. There is no package nor clear rules on how to split noisy text with many trailing blank lines and white spaces into paragraphs. Usually, the paragraphs are split on '\n\n'. This works reasonably well but is not always accurate. It may not give informative paragraphs. For example, when a title is encountered, it is treated as a paragraph but it is not long enough to hold valuable information. Figure 7 shows a fragment of a document and count each part that will be counted as a paragraph.

To reduce overhead documents are inserted into ES in bulk. The python Elasticsearch package has a built-in function `streaming_bulk` that is utilised.

6.2.2 Building Vocabulary

To ensure query expansion, a way how to determine synonyms has to be introduced. One such way is to measure the cosine similarity of word vectors. It would be possible to use `py_thesaurus` package but upon testing it, it did not have synonyms for basic words such as 'health' which could be because that it was stopped maintaining since 2018. The creators suggest using Oxford Dictionaries API instead, however, it is not open source. Building our vocabulary was the chosen approach.

To reduce the size of vocabulary, pre-processing is done to lowercase and lemmatize words, remove punctuation and words with numbers. Words with numbers were removed as meaningless strings were added into the vocabulary as shown in Table 4.

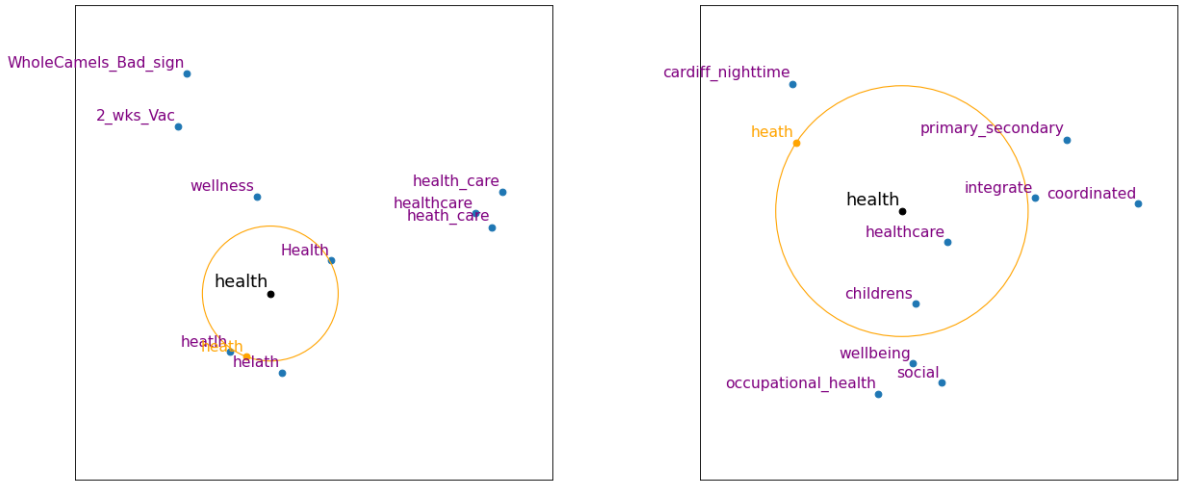
Numbers and Letters	**Letters Only**	**After Adding New Docs**
health	health	health
health_wellbeing	heath	heath
harr	primary_secondary	healthcare
cd7s2w	wellbeing	primary_secondary
irrance	coordinated	coordinated
health_service	acting_collaboration	childrens
nighttime	integrating	wellbeing
serve	rural	occupational_health
irrancadavies	social	integrate
exploitation	seamless	social

Table 4: Top 10 Most Similar Results for 'health' in Different Vocabularies

As the corpus is small, to see if more relevant synonyms would be returned, a scrape from a 'health and safety committee' was carried out to acquire 82 new documents. As Table 4 shows, more relevant words are returned. This means that the vocabulary should be trained on a sufficiently large corpus to ensure good results.

As the training corpus is small, we considered using pretrained `word2vec-GoogleNews-vectors`. These vectors are trained on news articles and contain 3 million words. However, they are not specific enough to our case. If we look for the top 20 bigrams in our entire corpus in the pretrained vectors, only 2 out of 20 words are found in the vocabulary. Figure 24 in the Appendix shows that if we train our own model, 14 out of 20 can be found in the vocabulary.

An easier to understand visualisation can be seen in Figure 8. If Principal Component



(a) Google-300 Word2Vec model

(b) Own Word2Vec model

Figure 8: Top 10 Similar Words Visualised Using PCA Method

Analysis (PCA) method to reduce the dimensionality of the word vectors from 512 dimensions to just 2 is used, such visualisation can be created. As the Figure shows, our own model is smaller but offers more diverse n-grams. If the model was trained on more data, the quality of n-grams would be only improved.

To find bigrams, Gensim Phrases model is used. Bigrams are found from the pre-processed text. Then, to find vector representations of grams, leammatized sentences where bigrams have been replaced with the new phrase representation ('health board' -> 'health_board') are used.

Finally, the vocabulary is inserted in ES. Before inserting the vocabulary, a new index has to be created. While all fields in ES are dynamically configured, if dense vectors are inserted, their dimensions have to be specified in the _mapping beforehand. The reason behind using dense vectors is to use ES built-in `script` field that can calculate the cosine similarity of inserted vectors. An important thing to note is that the ES `script` field currently does not support negative integers, thus the Word2Vec vectors have to be normalised before insertion.

Then 1.0 can be added to all numbers when calculating similarity without breaking ES.

6.2.3 Querying ES

This is the final part of building the search engine. A nested query has to be written to return the relevant paragraphs. The paragraphs have to be specified of type nested as then they can be queried independently of each other. The functionality of filtering date range, committee and item types is also inserted here.

Before executing the query, the vocabulary index is queried to find similar words to the search phrase. Currently, the top 5 most similar words are returned to expand the query. Finally, the query is executed and results are returned.

7 Products

7.1 Implementation

In this section, a detailed account of implementation will be described of the most critical parts.

7.1.1 Scrapy Crawler

In the previous section, it was determined that the starting point of the web crawler implementation was a spider. It has different callback functions and XPath rules to extract relevant fields. To understand the implementation better, further explanations will be provided for the Betsi Cadwaladr spider shown in Figure 9.

After defining the spider name and domain that the spider is allowed to crawl, the `start_requests` function is defined. In this case, there are two starting URLs; one for the board meetings and the other for the committees. For each, a different callback function is defined. This is because when crawling the committees URL, a `helper_committee_type` has to be recorded. `helper_committee_type` is an attribute that records an HTML field where the committee

```python
class BetsiCadwaladrSpider(scrapy.Spider):
    name = 'BetsiCadwaladr'
    allowed_domains = ['bcuhb.nhs.wales']

    def start_requests(self):
        # for board meetings
        yield Request('https://bcuhb.nhs.wales/about-us/health-board-meetings-and-members/health-board-meetings/',
                      self.parse_html, cb_kwargs=dict(helper_committee_type='board meeting'))
        # for committees
        yield Request('https://bcuhb.nhs.wales/about-us/committees-and-advisory-groups/', self.parse)

    def parse(self, response):
        committee_types = response.xpath('//table//strong/text()').getall()
        # some cells has very poor html as some invisible url duplicates appear
        # use library as converting to sets will loose order of elements
        urls = list(unique_everseen(response.xpath('//table//a/@href').getall()))
        for i in range(len(urls)):
            yield Request(
                url=response.urljoin(urls[i]),
                callback=self.parse_html,
                cb_kwargs=dict(helper_committee_type=committee_types[i])
            )

    def parse_html(self, response, helper_committee_type):
        # create manual requests to preserve committee type
        # item type sometimes mentioned in the <a> tag. If not, the item is an agenda
        helper_item_type = response.xpath('//table//a//text()').getall()
        urls = response.xpath('//table//a/@href').getall()
        for i in range(len(urls)):
            yield Request(
                url=response.urljoin(urls[i]),
                callback=self.parse_item,
                cb_kwargs=dict(helper_item_type=helper_item_type[i],
                               helper_committee_type=helper_committee_type)
            )

    def parse_item(self, response, helper_item_type, helper_committee_type):
        item = Item()
        item['name'] = self.name
        item['url'] = response.url
        item['response'] = response.body
        item['content_type'] = response.headers['Content-Type'].decode()
        item['last_modified'] = response.headers['Last-Modified'].decode()
        item['helper_item_type'] = helper_item_type
        item['helper_committee_type'] = helper_committee_type
        yield item
```

Figure 9: Betsi Cadwaladr Spider

type is mentioned. It is not a mandatory attribute to have as not all websites will have the committee type mentioned beforehand, in some cases, it can be found in the document URL. In this case, all committees have been listed in the start URL in a table. Recording the names of these fields will be more accurate as it is known that these are the committee types for this website.

The most generic rule that still extracts these fields has been written: `response.xpath('//table//strong/text()').getall()`. The double forward slashes mean that any element can be before and after the <table> tag as long as there is a tag as its descendant. In a similar manner the URLs are extracted. On this website however, not always one link is given per committee. Due to inconsistencies of HTML, sometimes there are 2 of the same URL. To solve the issue, duplicate URLs are filtered. Then, the extracted URL is re-yielded with an additional `helper_committee_type`.

The `parse_html` function extracts the `helper_item_type` from the HTML in the same manner as previously by using XPath. If this function is called directly as a callback from the `start_requests`, it is known that the committee type will be 'board meeting'. Otherwise this function is being called after `parse` function. The newly extracted URLs are re-yielded with both with `helper_committee_type` and `helper_item_type`.

Finally, the `parse_item` function is called. This function creates an Item object which is then further modified in the pipelines. All the required attributes are recorded.

Now, the item is passed through every pipeline shown in Figure 4. The order of the pipelines the item is passed through is defined in the `settings.py` file. The item travels through spider specific pipelines until it encounters the Betsi Cadwaladr Pipeline defined as shown in Figure 10. Here, it uses the `referer_html` attribute to extract `helper_date`. This tag will be used in the date extraction pipeline.

Next, the item is processed in the `item_type_pipeline`. This pipeline uses the extracted `helper_item_type` to assign a label to the item. The 4 available labels are defined. The `helper_item_type` is usually an extraction from HTML tag. It can be one or many words.

Betsi Cadwaladr website sometimes has `helper_item_type` as an empty string as <a> tag where it is extracted from sometimes does not have `/text()`. At the time being, this is an issue only when the item type is an agenda. To make the solution less specific than hard coding, the item URL is used as the item type is usually mentioned there too. Upon testing, it got determined that in cases where the `helper_item_type` is long, the confidence score would be very low. As the longest label is a bigram, splitting `helper_item_type` into unigrams and bigrams and then measuring similarity between the n-grams and labels improved the confidence scores. To measure the similarity between the pre-defined labels and `helper_item_type` n-gram word vectors, fastText pre-trained word vectors are loaded and then a Pandas data frame is created. Using Pandas data frame built-in function `df.apply()` is a lot faster than using a for loop to measure the similarity between each label and each n-gram. Then the label with the highest score is selected. If the confidence score is less than 0.65, then a generic label of 'papers' is assigned. The cut-off score was determined by recording a sample of `helper_item_types`, predicted labels and confidence scores and then manually labelling whether the prediction was correct.

The next pipeline the item goes through is `committee_type_pipeline`. Using a similar method as before, the committee type is determined. There are two differences. One difference is that there is no way of measuring similarity between the label 'other' for papers of interest that do not fall under the other specific labels. These committees had to be hard-coded as the cut-off score, in this case, is used to determine which committee papers AW is not interested in. If the score is lower than 0.65, the item is dropped and not processed further. The other difference is that the `helper_committee_type` is not split into n-grams of maximum length of a label. Sets of labels are used instead as a few websites mention the committee type in the breadcrumbs menu. However, due to inconsistencies, the level of mention will change as shown in Figures 11 and 12. Thus, a range of breadcrumb levels is recorded and a similarity score is found between each recorded level and a label. The snippet of the code can be seen in Figure 13.

Now, the item is processed in the `date` pipeline. First, the date extraction regex function has

```python
class BetsiCadwaladrPipeline(object):

    def process_item(self, item, spider):
        """
        The date for BetsiCadwaladr Spider is mentioned in the referer's pages table in the same row as the url was
        found in. Extract the content for that row and provide it as a better path for the date_pipeline to pick up
        from there
        """

        if spider.name == 'BetsiCadwaladr':
            tree = etree.fromstring(item['referer_html'], etree.HTMLParser())
            date = tree.xpath('//td/p//text()')
            item['helper_date'] = ' '.join(date)

        return item
```

Figure 10: Betsi Cadwaladr Pipeline

Home > Your Health Board > Statutory Committees of the Board > Audit and Risk Assurance Committee (ARAC) > ARAC Meetings 2020

Figure 11: Committee Type Mentioned in Level 4

Home > Integrated Governance Committee Meeting 27 January 2015

Figure 12: Committee Type Mentioned in Level 2

been defined as shown in Figure 14. This function matches all date formats of the following patterns: dd_mm_yy, dd_mm_yyyy, yy_mm_dd, yyyy_mm_dd, mm_dd_yyyy. The {0} syntax is left to format for all possible date separators which are ., -, / and a blank space. It is ensured that the same separator is used to avoid matching the wrong date in some cases. For example, if string 'QPSC Board Book V3 04.04.19.pdf' is processed, to avoid matching '3 04.04' first, the . separator has to be specified.

The `helper_date` is used to extract the date. After the date is extracted, the date format is normalised to yyyy/mm/dd. The full code can be seen in Figure 15.

Finally, the item goes through the `write_pipeline`. In this pipeline, content type is determined. Depending on whether the content is a PDF or DOCX, different methods are used to extract the text content of the response object. Then, the item object is converted into a JSON format and saved locally where the file name is a hash of the URL.

```
labels = ['board meeting',
          'audit committee',
          'charitable funds committee',
          'digital and information management committee',
          'quality and safety committee',
          'finance and performance committee',
          'remuneration committee',
          'local partnership forum',
          'mental health act committee',
          'stakeholder reference group',
          'health professionals forum',
          'strategy committee',
          'health and safety committee',
          'workforce committee']

nlp = spacy.load('tmp/')

class CommitteeTypePipeline(object):

    def process_item(self, item, spider):

        if spider.name == 'AneurinBevan':
            # newer resources are split by item type tables whilst old resources are public board meetings
            label = {item['helper_committee_type']} if item['helper_committee_type'] else {'public board meetings'}
        elif spider.name in ['BetsiCadwaladr', 'Cavuhb']:
            label = {item['helper_committee_type']}
        elif spider.name in ['CardiffVale', 'CwmTafMorgannwg', 'HywelDda', 'Powys', 'SwanseaBay']:
            label = set(item['helper_committee_type'])
        # label = {data['referer_html']} if data['referer_html'] else {'public board meetings'}
        label = {' '.join(re.findall(r'[a-z]+', i.lower())) for i in label}
        committee_type = None
        if label.intersection(OTHER):
            committee_type = 'other'
        else:
            # sometimes some strings have no letters, make tuples only from non empty items
            data = [list(itertools.product([i], labels)) for i in label if i]
            # 'SpaCy approach with FastText'
            df = pd.DataFrame(data, columns=labels)
            df = df.applymap(lambda x: nlp(x[0]).similarity(nlp(x[1])))
            # if committee type is less than 65%, then drop the item as it should be a committee type
            # that we are not interested to keep
            score = df.max().max()
            if score < 0.65:
                raise DropItem('Committee type score too low {}'.format(score))
            df['max'] = df.max().idxmax(axis=1)
            committee_type = df['max'].values[0]
            print('Committee type: {}, {}'.format(committee_type, score))

        item['committee_type'] = committee_type

        return item
```

Figure 13: Committee Type Pipeline

```
to avoid matching 3.04.04 in string 'QPSC Board Book V3 04.04.19.pdf'
we have to ensure that the same separator is used consistently
considers formats dd_mm_yy, dd_mm_yyyy, yy_mm_dd, yyyy_mm_dd, mm_dd_yyyy
= '((0?[1-9]|0[1-9]|1[0-9]|2[0-9]|3[0-1])(st|nd|rd|th)?'\
  '{0}(0?[1-9]|1[0-2]|(January|February|March|April|May|June|July|August|September|October|November|December)?|(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Sept|Oct|Nov|Dec)?)'
  '{0}((20)?([0-1]\d|2[0-1])))'\
  '|(((20)?([0-1]\d|2[0-1]))'\
  '{0}(0?[1-9]|1[0-2])'\
  '{0}(0[1-9]|1[0-9]|2[0-9]|3[0-1]))'\
  '|((January|February|March|April|May|June|July|August|September|October|November|December)'\
  '{0}(0?[1-9]|1[0-9]|2[0-9]|3[0-1])'\
  '{0}((20)?([0-1]\d|2[0-1])))'

updated = ''
parators = ['\.', '-', '\s', '/', ' ']
```

Figure 14: Regex Expression for Date Extraction

```
class DatePipeline(object):

    def process_item(self, item, spider):
        # drop html items
        if 'text/html' in item['content_type']:
            raise DropItem('Drop HTML item')

        item['date'] = None
        path = unquote(urlparse(item['url']).path)
        # as urls are of format
        # '/Docs/Board_Papers/Legacy_2015-2016/16-03_March_2016/AI_5.1__Appendix_2_Chairs_report_FP&W_for_UHB_2_March_16.pdf'
        # and regex would match '03_March_2016' first
        if spider.name == 'CwmTafMorgannwg':
            path = path.split('/')[-1]
        elif spider.name in ['AneurinBevan', 'CardiffVale']:
            path = item['helper_date']
        date = re.search(r_updated, path, flags=re.IGNORECASE)

        # for powys spider, if there is no date found in url,
        # sometimes the meeting date can be found in the referer's url
        if not date and spider.name in ['BetsiCadwaladr', 'CwmTafMorgannwg', 'HywelDda', 'Powys', 'SwanseaBay']:
            date = re.search(r_updated, item['helper_date'], flags=re.IGNORECASE)
        if date:
            item['date'] = dateparser.parse(date.group(), date_formats=['%d.%m.%Y', '%d-%m-%Y', '%Y-%m-%d'],
                                            settings={'DATE_ORDER': 'DMY'}).strftime('%Y/%m/%d')
            print('Date: ' + str(item['date']))

        return item
```

Figure 15: Date Extraction Pipeline

7.1.2 Search Engine

The first step is to create indexes in the ES. This can be done via Kibana - a front-end application providing search and data visualisation capabilities for data indexed in Elasticsearch [13], a PUT request through Postman or an ElasticSearch client for most programming languages including Java, Python, c#, PHP and others. It is a one-time process, similar to that of creating a relational database. Two indexes have to be created, one to store the vocabulary and the second, to store and query the documents. The vocabulary index, as mentioned in section 6.2.1, has to be predefined as `dense_vector` of size 300. For the documents index, the type nested has to be predefined. By default, ES will assume that the files are not nested which will result in returning the entire document. To prevent it, the type nested is predefined. The shards execute one thread at a time. Thus, to improve the speed, multiple shards are defined and the amount desired is specified in the mapping. To create the indexes the mapping is shown in Figure 16.

```
PUT vocabulary
{
  "mappings": {
    "properties": {
      "vector": {
        "type": "dense_vector",
        "dims": 300
      },
      "word" : {
        "type" : "keyword"
      }
    }
  }
}
```

(a) Vocabulary Index

```
PUT documents
{ "settings" : {
      "index" : {
        "number_of_shards" : 5,
        "number_of_replicas" : 2
      }
    },
  "mappings": {
    "properties": {
      "paragraphs": {
        "type": "nested"
      }
    }
  }
}
```

(b) Documents Index

Figure 16: Indexes for IR System

Once this is done, the vocabulary can be created. This is done by looping through all the crawled documents and preprocessing them by removing punctuation, numbers, stop-

words and lemmatizing and lower-casing words. Then the `gensim` Phraser is used to build bigrams. After, a Word2vec model is trained on the phrases model. This returns a vocabulary consisting of word and vector pairs. The vectors are 300 dimensions long and by default not normalised. Before inserting the vocabulary, the vectors have to be normalised. In the code, some lines save the intermediate models into files. Doing this ensures that when new data comes, the previous model can be loaded and the training can be restarted without the need to retrain the vocabulary on the previous data. The vocabulary is inserted in bulk via ES Python client. The full source code can be seen in Figure 25 in the Appendix.

The next part is to write search functions. The first search function is used to return the top n most similar words of the search query. This is done by returning the word vectors for each queried word and then finding the mean vector between all the searched words. After, the vocabulary index is queried to find the most similar words based on cosine similarity between the averaged word vector and all other vocabulary vectors. A sample query executed in the Kibana can be seen in Figure 17. Here a word vector for the word 'health' is provided and the returned results are the top 10 most similar words to the word 'health'.

Our own script has to be specified as follows:

$$cosineSimilarity(params.queryVector, `vector') + 1.0$$

As ES does not support negative numbers when measuring cosine similarity, 1 has to be added to all numbers as Word2Vec vectors have negative floats. Now, the maximum similarity score instead of 1.0 will be 2.0 indicating an exact match. As seen in Figure 17 before, as the vocabulary contains the word 'health' as well, it is returned as the most similar word with a similarity score of 2.0 as it is an exact match. The next most similar word is 'heath' with a similarity score of 1.476.

The other search function is more advanced. First, the search query is expanded with the words returned from the previous query. The most important part is to use 'more_like_this' query with nested fields to only return the relevant paragraphs of the document instead of all paragraphs if at least one is relevant. Then, optional parameters to filter by date range,

Figure 17: Query and Results for Word 'health'

committee and item types has to be defined. The fields that we want to be returned have to be specified unless we want to return all fields. To improve the speed, the fields of interest are specified: 'date', 'url', 'committee_type' and 'item_type'. Helper attributes and full responses are not needed. The results are ordered in descending date order. It will query for the top 10 most similar documents and then rearrange the order depending on the date.

The search queries in Python are defined the same as in Kibana in a JSON like format. The source code for this function can be seen in Figure 18. The full code of the ElasticSearch functions can be found in the zip folder in the SearchEngine sub-folder submitted with this report.

7.2 Project management

The project was managed in agile like manner. There were no sprints as such but there were weekly meetings with university and AW supervisors to determine what was done in the previous week, what will be done the next week and whether any issues were encountered. This approach worked well in terms of managing time and determining what will be feasible to achieve by the end of the project.

```
def search_bm25_extra_grams(search_phrase, committee_type=None, item_type=None, date_range=None):
    res_vocab = search_similar_terms(search_phrase)
    similar_grams = [hit['_source']['word'] for hit in res_vocab['hits']['hits']]
    print(similar_grams)
    must_match = []
    if item_type:
        must_match.append({"match": {"item_type": item_type}},)
    if committee_type:
        must_match.append({"match": {"committee_type": committee_type}})
    if date_range:
        must_match.append({"range": { "date": { "gte": date_range[0],
                                                "lte": date_range[1]}}})
    must_match.append(
        {
            "nested": {  # to treat each paragraph as a separate document, specify nested path
                "path": "paragraphs",
                "query": {
                    "more_like_this": {
                        "fields": ["paragraphs.paragraph"],
                        "like": similar_grams,  # a list of search terms
                        "min_term_freq": 1,  # min amount of times for term to appear
                        "max_query_terms": 10  # range for terms to appear
                    }
                },
                "inner_hits": {
                    "size": 10,  # max relevant paragraphs
                    "highlight": {
                        "fields": {
                            "paragraphs": {}  # to return relevant paragraphs use highlight
                        }
                    }}}}
    )
    query = {
      "_source": ["date", "url", "committee_type", "item_type"], # json fields to return. Delete the line to return al
      "size" : 10, # amount of docs to return
      "sort" : [
        { "date": {"order" : "desc"}} # sort results by most recent first
      ],
      "query": {
        "bool": {
          "must": must_match}}}

    res = es.search(index="documents_test", body=query) #documents_test
    return res
```

Figure 18: ElasticSearch Querying Function

```
2020-09-03 08:06:51 [scrapy.core.engine] INFO: Spider opened
2020-09-03 08:06:51 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:51 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6024
2020-09-03 08:06:52 [scrapy.extensions.logstats] INFO: Crawled 98 pages (at 5880 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:53 [scrapy.extensions.logstats] INFO: Crawled 172 pages (at 4440 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:54 [scrapy.extensions.logstats] INFO: Crawled 253 pages (at 4860 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:55 [scrapy.extensions.logstats] INFO: Crawled 367 pages (at 6840 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:56 [scrapy.extensions.logstats] INFO: Crawled 471 pages (at 6240 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:57 [scrapy.extensions.logstats] INFO: Crawled 577 pages (at 6360 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:58 [scrapy.extensions.logstats] INFO: Crawled 658 pages (at 4860 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:06:59 [scrapy.extensions.logstats] INFO: Crawled 764 pages (at 6360 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:07:00 [scrapy.extensions.logstats] INFO: Crawled 855 pages (at 5460 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:07:01 [scrapy.extensions.logstats] INFO: Crawled 952 pages (at 5820 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 08:07:01 [scrapy.core.engine] INFO: Closing spider (closespider_timeout)
```

Figure 19: Benchmarking Crawl Output

```
2020-09-03 09:06:29 [scrapy.extensions.logstats] INFO: Crawled 2 pages (at 2 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 09:07:29 [scrapy.extensions.logstats] INFO: Crawled 2 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 09:08:29 [scrapy.extensions.logstats] INFO: Crawled 2 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 09:09:29 [scrapy.extensions.logstats] INFO: Crawled 2 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-09-03 09:10:29 [scrapy.extensions.logstats] INFO: Crawled 6 pages (at 4 pages/min), scraped 4 items (at 4 items/min)
2020-09-03 09:11:29 [scrapy.extensions.logstats] INFO: Crawled 10 pages (at 4 pages/min), scraped 8 items (at 4 items/min)
2020-09-03 09:19:35 [scrapy.extensions.logstats] INFO: Crawled 13 pages (at 3 pages/min), scraped 10 items (at 2 items/min)
2020-09-03 09:20:29 [scrapy.extensions.logstats] INFO: Crawled 16 pages (at 3 pages/min), scraped 14 items (at 4 items/min)
2020-09-03 09:21:29 [scrapy.extensions.logstats] INFO: Crawled 19 pages (at 3 pages/min), scraped 15 items (at 1 items/min)
2020-09-03 09:22:29 [scrapy.extensions.logstats] INFO: Crawled 19 pages (at 0 pages/min), scraped 15 items (at 0 items/min)
```

Figure 20: Benchmarking Throttled Crawl Output

8 Analysis

Scrapy framework has its benchmarking suite. It creates its local HTTP server and crawls it at the maximum speed [14]. These stats can give an idea of how Scrapy would perform on the given hardware. The output can be seen in Figure 19. If the results are averaged, it shows that Scrapy can crawl about 5712 pages per minute on a machine with 6 cores and 16GB of RAM. However, due to most websites requesting the spiders to be throttled to 2 requests per minute, the crawl speed in the run-time will be approximately that as shown in Figure 20. Many 0 pages/min are caused by redirection requests. It is possible to tell Scrapy to disobey robots.txt - a file that tells crawlers how to crawl their website. However, disobeying the rules may result in the bot getting banned as many requests can appear as a DDoS attack. It can be avoided by using Proxy rotators but it is not entirely clear if this is ethical.

To test the performance of the committee and item type labelling, a random sample of 100

documents was selected. The committee and item types were labelled by the pipelines and then manually checked. It was determined that 86 out of 100 documents had the correct committee type label giving 86% accuracy. Upon further inspection, the incorrectly labelled committee types are originating from the `helper_committee_type` where the true label falls under the category 'other'. To give an example, when `helper_committee_type` is 'Integrated Governance Committee Meeting 28 April 2015' we know that the committee type should be 'Integrated Governance Committee' which should be labelled as 'other'. However, because all the labels of type 'other' are hard-coded, and the string does not match exactly due to the added date, it will label the committee incorrectly. The `item_type` pipeline gave accuracy of 98%. Figure 31 in the Appendix shows the results. The only incorrectly labelled results were when the keywords are very wide apart. For example, when the `helper_item_type` is 'Approved Corporate Safeguarding Annual Report' it will get labelled as papers. Due to splitting the string into bigrams, the score of matching it to 'final minutes' will be too low, thus labelling the report as papers.

Evaluating date extraction manually would be labour intensive. To give a rough idea, the only performance metric acquired to measure this was how many documents had a date extracted. Out of 100 sample documents, 87 of them had a date. In cases when the date is not extracted, the last modified date could be used. However, inspecting the dates extracted and last modified often would differ by years. For example, the extracted date from a board meeting was '21/10/2009' which was confirmed to be the correct meeting date. The last modified date for the same document was '14/11/2015'. The dates differ by 6 years which means that assigning the last modified date would be very inaccurate. As the date extraction has very high importance, it should be improved in the future. It should also be noted that there will always be an error. As most documents are human-made, upon developing the project, many misspelt words were encountered. This can cause an error when extracting date as the regex expression will not match the string '17th Novmeber, 2019' due to the misspelled word 'November'.

ES Rally is a tool used to benchmark ElasticSearch. The machine this project was devel-

oped on has Windows OS. ES Rally, however, is only developed for Unix systems. The benchmarking was run on a server that has Debian OS via SSHing. ES Rally has built-in benchmarking datasets with different challenges. The project uses ES version 7.9.0 thus the testing was done on the said version with a challenge to measure the performance of nested queries. The dataset is called Nested and the challenge nested-search-challenge. This dataset contains questions and answers of Stackoverflow of the totalling size of 3.4GB of data and implements a similar nested structure to the 'documents' index created for this project. In table 5 it is shown that the median throughput of queries is 18 ops/s (which is how many operations have been sent a second but should be interpreted as how many documents per second). If we look at the 50th and 100th percentile latency, it is shown that on average it takes 63.8 milliseconds to return results for a nested query and at most 197ms. If the queries are executed on larger documents, the latency on average is 126ms and at most 378.1ms. The results of all tests are shown in Figure 32 in the Appendix. To get a better performance measure on our specific case, where 2 queries are executed, one to acquire synonyms and second to query for similar documents, a query is executed 10 times of the documents index which contains 3241 documents. Then the average execution time is recorded. The average response time was 0.538 seconds. The goal was to return the top 10 most similar documents in less than 2 seconds which means the goal was achieved.

The most time-consuming process in this project is to build the vocabulary. The random crawled sample is a little over 3000 documents and to extract bigrams from that on a 16GB RAM machine takes 529.7 seconds. Then, to create the Word2vec model takes 503.4 seconds. To speed the process up, a machine with more RAM would be needed. As this process takes so long and is not necessary to be run every time a crawl has been done, it is recommended to create it as a separate service.

Originally, the Search Engine tool was meant to be an automatic text summarization tool. Upon research on automatic text summarization, two approaches were determined: extractive and abstractive. Extractive text approach ranks sentences by importance and then selects the highest-ranked sentences as the summary. The abstractive approach generates new, unseen

Metric	Task	Value
Min Throughput	randomized-nested-queries-with-inner-hits_default	17.97 ops/s
Median Throughput	randomized-nested-queries-with-inner-hits_default	18 ops/s
Max Throughput	randomized-nested-queries-with-inner-hits_default	18.01 ops/s
50th percentile latency	randomized-nested-queries-with-inner-hits_default	63.81577131 ms
100th percentile latency	randomized-nested-queries-with-inner-hits_default	197.0269746 ms
Min Throughput	randomized-nested-queries-with-inner-hits_default_big_size	15.96 ops/s
Median Throughput	randomized-nested-queries-with-inner-hits_default_big_size	15.99 ops/s
Max Throughput	randomized-nested-queries-with-inner-hits_default_big_size	16 ops/s
50th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	126.5732895ms
100th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	378.18228 ms

Table 5: Nested ES Rally Challenge

sentences to better summarize the text. To start with, a simple text rank algorithm was implemented to summarize meeting minutes. However, it was quickly discovered that the tool was intended to be used only on annual reports that is a summary of everything done throughout the year. These reports can be more than 500 pages long and cover 100s of topics. It would be impossible to write a summary of a summary in 10 sentences when the report covers many more topics than the amount of the desired summary sentences. Therefore, the project got changed to build an IR System instead.

The final two products are ready to be deployed as commercial products. The crawling system can reliably crawl all the desired websites, can be integrated with AWS, is easily expansible, extracts the desired metadata, extracts text from PDFs and DOCX documents, saves the documents, and offers the option to crawl documents once (persistence). The search engine system also is compatible with AWS, takes less than a second to return results for a search query, expands search query by adding synonym words, offers the option to filter by date, item, and committee types, orders the results in date descending order and only returns the relevant paragraphs of the document.

9 Conclusions

In conclusion, the project has been a successful prototype of both crawling and IR systems which can be implemented as a commercial tool. The crawling project is fully working and can be integrated with AWS. It is extracting the desired attributes and text as well as has a sufficiently large example of web-scrapers for future expansion to other websites. The search engine is also fully working. It returns the top 10 documents in less than a second and has filtering options available. There are some flaws that will be addressed in the next section.

The project is also a success according to the performance auditor who would be using the tool. It would decrease the workload by automating the process of document retrieval and building an IR system. To give a specific example, the auditor mentioned that recently they went through 5 years worth of documents from many different public body websites to find a document that'd fit their needs. In the end, no such document was found. They mentioned that a tool that is built now, would have reduced the time greatly as all the documents would be collected in one space and would be easy to search.

Throughout the project, there were no major issues encountered. Most problems were resolved quickly via looking up tutorials on TowardsDataScience website [15] and searching Stackoverflow. It is recognised that as the project specification was only describing the desired end product, a lot of things were left up to the implementer on how to arrive at the final project. This includes making decisions on the programming language used or architectures implemented. The code is up to coding standards with PEP8, with the only exception to line lengths.

9.1 Future Work

Due to the lack of time, there are a few things that could improve the project that have not been addressed yet.

Firstly, there is no auto-correct in the IR system. When a user misspells a word, there is no spell check that would make sure that it is an actual English word. Similarly, the words inserted in the vocabulary also can be any letter string that might not be a meaningful word. There exist libraries for Python such as PyEnchant that can check if the word exists and suggest a corrected version of the word. In the future, such a spell check would be beneficial.

Another flaw is that currently the vocabulary is built only from unigrams and bigrams. To build trigrams, the Phraser model would need to be run twice. This means the first time it has to be run to build bigrams and the second time to build trigrams from the bigrams. As this process is time-consuming and the data sample is very small, adding trigrams in the vocabulary would not have added any value. However, if the project is used commercially with a lot more data, the vocabulary should also contain trigrams.

A nice feature to have would be to return not only the relevant paragraphs but also the paragraph before and/or after the relevant paragraph. To do this, the indexes of the paragraphs have to be recorded and then a query request for paragraphs at $index + 1$ or $index - 1$ can be executed to return the paragraph before or after. This feature would only be useful when the front-end is implemented. This brings up the next point.

Whilst the back-end functionality is implemented, there was not enough time to build a user interface (UI). UI is very important for the end-user as it is not easy to understand and interpret the output of ES in the console. A mock-up of the proposed front-end was shown in the Section 2 in Figure 1.

It is recognised that only a limited amount of ranked similarity algorithms was tested. The choices were made based on the amount of supporting literature. However, there are also other algorithms such as Divergence From Randomness (DFR), Divergence From Independence (DFI), Information Based (IB), LM Dirichlet and many other similarity models that could be tested.

For the crawlers project, as mentioned earlier, the date extractor has very high importance,

Whilst it gives a high accuracy of 0.86, it would be advised to improve it as much as possible as the filtering option and date sorting option is relying on this extracted date.

It was mentioned that Scrapy has an option to record the fingerprints of websites to crawl them only once to improve efficiency. However, it was not mentioned what to do in cases, when the documents are very similar but are not identical nor have the same URL. For example, when a draft minute document has been revisited to make it into a final minute document but the changes are insignificant. If both are recorded, then upon querying, the user might find themselves looking at the same results repeated but under two unique document IDs. This is very a dupe filter that would add value to the project. One proposed method could be using ES and their built-in similarity modules in a similar manner the query search is working. However, this is not efficient as a full-text comparison will be executed. Another, more elegant solution would be to use Sim-Hash. Sim-hash is a hashing function that hashes similar text inputs into similar hashes [16]. The more similar texts have been given as input, the smaller the Hamming distance of their hashes. Comparing hashes would be faster than comparing full texts. To reduce the load of ElasticSearch or to use a cheaper alternative another database can be used. For example, Figure 21 shows the price in Europe (Ireland) for the most basic instances for ElasticSearch and Redis. The smallest instance in Redis cost 0.018 $/h whilst ES costs 0.2 $/h at the time of the completion of this project. It might seem like the difference is not significant, but it is worth considering the size of the instance needed and the time they will be used as the price will go up.

General Purpose - Current Generation	vCPU	Memory (GiB)	Instance Storage (GB)	Price Per hour
t2.micro.elasticsearch	1	1	EBS Only	$0.02
t2.small.elasticsearch	1	2	EBS Only	$0.039
t2.medium.elasticsearch	2	4	EBS Only	$0.078

(a) Pricing for ES

Standard Cache Nodes - Current Generation	Price Per Hour
cache.t3.micro	$0.018
cache.t3.small	$0.036
cache.t3.medium	$0.072

(b) Pricing for Redis

Figure 21: Pricing of AWS Services in Europe (Ireland)

Another useful tool that would bring great value to the production code would be using DataDog [17] or similar software. DataDog is a monitoring service for cloud applications. It can be used to monitor data from servers, databases, containers and other services including

Scrapy framework. It can be added as simply as a middleware. Then it can be configured to send desired metrics such as items scraped count and HTTP status codes to monitor the performance of the crawler as well as easily detect if the crawler stops working if web-pages are timing out, redirecting too often or do not exist anymore. All the data is displayed in easy to understand dashboards.

As a good industry standard, both projects need unit tests to detect changes that might break the current design. Unit tests are encapsulated and do not use external resources. This can come in handy when detecting whether there is a bug in the code or with the infrastructure of how the code is getting deployed. In the later stages of a project, this can be used as a part of continuous development where unit tests are used before auto-deploying newly merged changes onto a live system.

10 Reflection/Learning

This project allowed me to develop a set of new skills as well as utilise already acquired skills.

The project was delivered to AW as a zip file of source code. The handover involved a series of demos of the project to the entire DA team and the performance auditor as well as a walkthrough of the code to the selected people of the DA team.

As I had developed a crawling project while working in an industry, I had a prior knowledge of existing technologies and integration methods used when developing web-scraping projects. However, the spiders I had written before were a lot more simple. They usually were using website sitemaps or built-in searches to return the desired results. The websites I was working with during this project required a lot more complex manipulation. I had to learn how to re-yield the HTTP requests whilst keeping the extracted attributes and how to accurately use XPaths.

When deciding on how to reliably and robustly develop an item and committee type detection

pipelines, I did not know which method would be the best. Researching and comparing different methods is a time-consuming process. It requires a lot of patience. However, it has high importance as different methods are better in different scenarios. It is important to critically evaluate possible solutions to choose the best fitting one.

Similarly, the research on document extraction was helpful. In the end, the solutions that could be integrated within the pipeline were chosen. However, other solutions, specifically for DOCX documents could have been improved. As word documents are based on XML, it would have been possible to extract titles and recognise bullet-points. For PDFs, it also would have been possible to extract tables. However, the table extraction libraries were very time consuming (can take 10s of seconds for a single document) as well as not very precise, when dealing with merged cells. I felt like I did not have enough knowledge of document extraction to write something better as most existing libraries were not able to capture a lot of the text structure. Based on research, it would not be easy and would require a lot more knowledge than I had so it was no implemented in the solution.

I have used ES before but I had never built an IR system. Reading the relevant books I learned how to build one. I also did not know that ElasticSearch is supporting vectors and has been expanded to include a built-in Cosine similarity function. This project was refreshing as well as expanding my knowledge of ES.

I have used ES before but I had never built an IR system. Reading the relevant books I learnt how to build one. I also did not know that ElasticSearch is supporting vectors and has been expanded to include built-in Cosine similarity function. This project was refreshing as well expanding my knowledge of ES.

This project proved that I can independently carry out a project. Usually, when working in a professional environment, there are many people with different roles that ensure that big tasks are broken down into small, concise units such as tickets in a Kanban board. There is also an architect who has a clear idea on how everything will work together and often which technologies should be used. However, the planning and choices of the design were left all

to me. I successfully managed to plan my time to complete the project on time. Of course, if more time was given there would be many things to improve. This means that I had to learn how to prioritise the most important parts of the project to make it functional even if not fully finished. I think this experience improved my leadership and problem-solving skills.

The biggest struggle I found during this project was my lack of knowledge of already existing libraries, solutions and frameworks. Whilst I can find solutions that work and can be integrated, often, I find myself questioning if these are the best solutions and what other undiscovered options have I missed. Or whether there are ways of how to improve the speed and performance of the tools that I have made. To try to address this issue, I tried to do thorough research in combination with my prior knowledge of working in industry. I found that the best source of information was asking people who have done something similar. They were able to recommend research papers and books. The second best source of material was the Cardiff University's Library search. It has many good books and papers available to read online. The best way how to deal with bugs or errors within the code was Google searches and Stackoverflow.

The other struggle I often have is poor time management. Usually, I find myself coding the project until a week before the deadline and then rushing the report. This time I tried to allocate half of the given time for coding and the other half for writing to ensure that both parts of the project are completed. The weekly meetings with university and placement supervisors were useful to assure that I am managing my time well. The demos throughout the project with the data analytics team were useful to give feedback on the project to assure it is going in the right direction. In the end, I think I managed my time well.

Having to carry out the project fully remotely definitely improved my communication skills. Due to the lock-down, the supervisor, DA team meetings and the meetings with the end-user were all over Skype or MS Teams. To get the most value out of these meetings, all parties had to know beforehand what the meeting was about and what was the desired outcome of the meetings. When working in an office, it is easy to walk up and clarify small things.

Now, everything had to be thought through a little better and communicated in a clear manner.

In conclusion, I feel like this project was a good project to demonstrate my existing technical skills in combination with learning many new things. I improved my time management, communication and problem-solving skills which will come in handy when working in industry.

References

[1] "Publication scheme — audit wales." https://www.audit.wales/openness-and-transparency/publication-scheme. (Accessed on 09/14/2020).

[2] "World wide web wanderer (complete history)." https://history-computer.com/Internet/Conquering/Wanderer.html. (Accessed on 08/28/2020).

[3] R. Mitchell, *Web Scraping with Python: Collecting More Data from the Modern Web*. Sebastopol: O'Reilly Media, Incorporated, 2018.

[4] M. Heydt, *Python web scraping cookbook: over 90 proven recipes to get you scraping with Python, microservices, Docker, and AWS*. Birmingham: PACKT Publishing, 1st ed. ed., 2018.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[6] "What's so hard about pdf text extraction? ." https://filingdb.com/b/pdf-text-extraction. (Accessed on 07/27/2020).

[7] "The differences in british and american spelling — oxford international english schools." https://www.oxfordinternationalenglish.com/differences-in-british-and-american-spelling/: :text=The(Accessed on 08/18/2020).

[8] seatgeek, "Fuzzywuzzy." https://github.com/seatgeek/fuzzywuzzy, 2011.

[9] "One billion data from mysql imported into elasticsearch, how es performance - elastic stack / elasticsearch - discuss the elastic stack." https://discuss.elastic.co/t/one-billion-data-from-mysql-imported-into-elasticsearch-how-es-performance/22461/2. (Accessed on 08/24/2020).

[10] "What is elasticsearch? — elastic." https://www.elastic.co/what-is/elasticsearch. (Accessed on 08/20/2020).

[11] "Web crawler." https://www.sciencedaily.com/terms/web_crawler.htm. (Accessed on 08/12/2020).

[12] "Item pipeline — scrapy 2.3.0 documentation." https://docs.scrapy.org/en/latest/topics/item-pipeline.html. (Accessed on 08/13/2020).

[13] "What is kibana? — elastic." https://www.elastic.co/what-is/kibana. (Accessed on 09/02/2020).

[14] "Benchmarking — scrapy 2.3.0 documentation." https://docs.scrapy.org/en/latest/topics/benchmarki (Accessed on 09/03/2020).

[15] "Towards data science." https://towardsdatascience.com/. (Accessed on 09/17/2020).

[16] "Sim-hash: Detection of duplicate texts — by saurav omar — medium." https://medium.com/@sauravomar01/sim-hash-detection-of-duplicate-texts-d5dc2ce2538a. (Accessed on 09/08/2020).

[17] "Cloud monitoring as a service — datadog." https://www.datadoghq.com/. (Accessed on 09/14/2020).

A Appendix

```
>>> dparser.parse('2015/16 Workforce  Organisational Development Committee Agenda - 30 June 2015', fuzzy=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\ivonn\.virtualenvs\Dissertation2020-o0RO07WS\lib\site-packages\dateutil\parser\_parser.py", line 1374, in parse
    return DEFAULTPARSER.parse(timestr, **kwargs)
  File "C:\Users\ivonn\.virtualenvs\Dissertation2020-o0RO07WS\lib\site-packages\dateutil\parser\_parser.py", line 649, in parse
    raise ParserError("Unknown string format: %s", timestr)
dateutil.parser._parser.ParserError: Unknown string format: 2015/16 Workforce  Organisational Development Committee Agenda - 30 June 2015
```

Figure 22: Dateparser Returns Errors When String Contains Only Year

```
>>> for date in dates:
...     print('{0:25} {1}'.format(str(dparser.parse(date,fuzzy=True)), date))
...
2010-05-26 00:00:00       Public Board Meeting 26th May 2010
2018-12-06 00:00:00       Thursday 6th December 2018
2010-11-24 00:00:00       Public Board Meeting 24th November 2010
2015-05-26 00:00:00       Audit Committee - 26 May 2015
2015-06-16 00:00:00       Quality Safety and Experience Committee - 16 June 2015
2013-10-15 00:00:00       Quality Safety & Experience Committee - 15 October 2013
2013-03-05 00:00:00       Quality and Safety Committee - 5 March 2013
2013-12-10 00:00:00       Quality Safety and Experience Committee - 10 December 2013
2020-02-26 00:00:00       26 February 2020
2014-02-18 00:00:00       Quality, Safety and Experience Committee - 18 February 2014
2020-05-28 00:00:00       28th May 2020
2019-08-15 00:00:00       15 August 2019
2014-08-12 00:00:00       Audit Committee - 12 August 2014
2004-03-10 20:20:00       Quality, Safety and Risk Committee  /  10 MARCH 04 2020  /
2019-11-28 00:00:00       Charitable Funds Committee  /  28 November 2019  /
2020-01-12 00:00:00       Finance, Performance and Workforce Committee  /  012 January 2020  /
2014-07-12 20:20:00       Quality, Safety and Risk Committee  /  12 JULY 14 2020  /
2004-03-10 20:20:00       Quality, Safety and Risk Committee  /  10 MARCH 04 2020  /
2017-08-08 00:00:00       8 August 2017 UPB Meeting
2015-04-20 00:00:00       Strategy & Planning Committee Meeting 20 April 2015
2014-10-16 00:00:00       Strategy and Planning Committee Meeting 16 October 2014
2015-11-16 00:00:00       16 November 2015 UPB Meeting
2014-10-16 00:00:00       Strategy and Planning Committee Meeting 16 October 2014
2015-01-27 00:00:00       Integrated Governance Committee Meeting 27 January 2015
2015-11-16 00:00:00       16 November 2015 UPB Meeting
2015-04-20 00:00:00       Strategy & Planning Committee Meeting 20 April 2015
```

Figure 23: Incorrectly Determined Date by Dateparser

health_board	False
quality_safety	False
mental_health	True
primary_care	False
cwm_taf	False
welsh_government	False
univeristy_health	False
safety_risk	False
nhs_wales	False
action_plan	False
performance_workforce	False
executive_director	False
internal_audit	False
risk_committee	False
agenda_item	False
health_safety	False
health_care	True
finance_performance	False
patient_safety	False
aneurin_bevan	False

(a) Bigrams in Google-300

health_board	True
quality_safety	True
mental_health	True
primary_care	True
cwm_taf	True
welsh_government	True
univeristy_health	False
safety_risk	False
nhs_wales	False
action_plan	True
performance_workforce	False
executive_director	True
internal_audit	True
risk_committee	False
agenda_item	True
health_safety	True
health_care	True
finance_performance	True
patient_safety	False
aneurin_bevan	False

(b) Bigrams in our own pretrained Word2Vec model

Figure 24: Top 20 bigrams of the corpus

```
lemmatizer = nltk.stem.WordNetLemmatizer()

module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
model = hub.load(module_url)


def remove_punct(tokens):
    no_punct = []
    for token in tokens:
        token = re.sub(r'[^a-z]', '', token)
        if token and len(token) > 1:
            no_punct.append(token)

    return no_punct


def build_phrases(sentences):
    phrases = Phrases(sentences,
                      min_count=5,
                      threshold=3,
                      progress_per=1000)
    return Phraser(phrases)


def get_sentences(text):
    sentences = nltk.tokenize.sent_tokenize(text.lower())
    lemmatized_sentences = []
    for sentence in sentences:
        tokens = sentence.split()
        cleaned_tokens = remove_punct(tokens)
        lemmatized_tokens = [lemmatizer.lemmatize(token) for token in cleaned_tokens if token not in STOP_WORDS]
        # don't add empty lists
        if lemmatized_tokens:
            lemmatized_sentences.append(lemmatized_tokens)
    return lemmatized_sentences


def sentence_to_bi_grams(phrases_model, sentence):
    return ' '.join(phrases_model[sentence])


def embed(input):
  return model(input)


def loop_through_files(folders):
    lemmatized_sentences = []
    for folder in os.listdir(folders):
        path = folders + '/' + folder + '/'
        for file in os.listdir(path):
            with open(path + file, 'r', encoding="utf8") as f:
                data = json.load(f)
                # some docs have response as null
                if data['response']:
                    # replace blank lines with dots as titles usually don't have period to indicate the end of senten
                    lemmas = get_sentences(data['response'].replace('\n', '. '))
                    # discard empty lists
                    if lemmas:
                        lemmatized_sentences += lemmas
        #         break
        # break

    return lemmatized_sentences

# create phrases
lemmatized_sentences = loop_through_files('../out')
phrases = build_phrases(lemmatized_sentences)
# phrases.save('phrases_model.txt')

# build w2v vocab
lemmatized_corpus = phrases[lemmatized_sentences]
w2v_model = Word2Vec(lemmatized_corpus, size=300, window=5, min_count=2)
# w2v_model.save('w2v_vocab.model')
# w2v_model = KeyedVectors.load('w2v_vocab.model')
```

Figure 25: Create Vocabulary Source Code

Risk Assessment 2019-20

WALES AUDIT OFFICE
SWYDDFA ARCHWILIO CYMRU

XXXX Health Board

Risks that the organisation's arrangements do not support good governance or efficient, effective and economical use of resources.

The Code of Audit Practice [CoAP] requires auditors to develop an annual programme of work, based on an assessment of risks of the body not making proper arrangements for securing effective use of resources. Auditor General Guidance XC18 (November 2019): 'Proper Arrangements in the Use of Resources' sets out requirements in further detail.

Auditors should use the risk assessment tool set out in this document to record audit risks identified from work undertaken and included in the 2019 Annual Audit Report. This MUST include the risks identified through Structured Assessment. Auditors should also consider:

- the following[1] audit findings and results (as relevant to arrangements for securing value for money):
    - audit of financial statements, review of the annual governance statement and any correspondence regarding the audited body;
    - other audit work, including recent audit reviews and vfm examinations / studies (where relevant), the progress the audited body has made in responding to recommendations and the use of data matching exercises; and
    - the work of internal audit and of external review bodies where appropriate.
- wider intelligence and information gained through Board / Committee [attendance, document and data review]; and engagement with key officers as part of the risk assessment and local audit planning process.

The risk assessment tool (Appendix 1) is grouped by theme, reflecting areas considered as part of Structured Assessment and as included in the CoAP.

Within each theme, auditors should record:

- the specific risk[s] identified, including:
    - the rationale [and root cause where known]; and
    - implications for economy, efficiency and effectiveness; and
    - implications for WFG and the SD Principle;

[1] Includes the work set out in the Code of Audit Practice to be considered in reaching a value for money conclusion

- the evidence source[2];
- any mitigating actions being taken by the audited body to lessen or address the risk;
- a RAG rating of the [post-mitigation] residual risk, by theme; and
- proposed actions for the Wales Audit Office engagement team to consider as part of audit planning[3].

The RAG ratings for each thematic area should be used to inform an overall assessment of risks and prioritisation of audit work to be considered for 2020. The ratings should reflect the level of audit risk after mitigations by the audited body.

RAG descriptors are:

	No significant risks or concerns.
	The organisation is aware of the risks identified and has well developed and appropriate plans in progress to address issues.
	The organisation recognises the risks identified but plans to address these are not well developed or not progressing with sufficient pace.
	There are significant risks, with little or no awareness and / or no plans to address the issues.

[2] Evidence sources may include structured assessment, other performance/financial audit, vfm studies, NFI, internal audit or other regulatory reviews

[3] Options could include full review, monitoring, follow-up or light-touch review; or raising with internal audit or other inspectorate/review body/WAO team.

Figure 26: Risk Assessment Tool Guidelines

Thematic area
1. Well led, well governed (corporate governance)
• Board and committee effectiveness (quoracy, decision logs, public meetings, terms of reference, Standing Orders) • **Risk management** – risk strategy, **board assurance framework**, **risk register**, **risk appetite**, risk management policies and procedures • Board assurance • Performance management • **Quality governance**, including **clinical audit**, **complaints** and **incidents** (patients, staff or visitors), **patient experience**, **patient outcomes**, **Health & Care Standards**, **patient/staff stories, quality improvement** • Management information • Reporting and scrutiny • Organisational structures
2. Strategic planning
• Organisational strategy and strategic objectives • Alignment with WFG and implementing SD principles • Clinical services plan • Integrated medium-term plan (IMTP) development / approval • Planning capacity • Strategic planning approach • Public / partner engagement • Monitoring and scrutiny of plan delivery • Programme and change management capacity
3. Use of financial resources
• Financial planning • **Financial controls** (**Schemes of Delegation**, Registers of Interest, Standing Financial Instructions) • Preventing fraud and financial loss • Budget setting • Financial costing • Financial efficiencies • Cost improvements • **Procurement** including **Single Tender Action** (STA)/**Single Quotation Action** (SQA) • Contract management • Asset management
4. Performance
• productivity • efficiency • targets • performance trajectory
5. Workforce management
• Workforce planning • Workforce strategy • Workforce productivity e.g. sickness absence rates • Temporary staff – bank, agency, locum • Appraisal and performance and development reviews (PADR) • Medical revalidation • Consultant job planning • Recruitment, retention • Safe staffing levels • Staff engagement and wellbeing • Training and development

Figure 27: Thematic Areas of Risk Assessment Tool

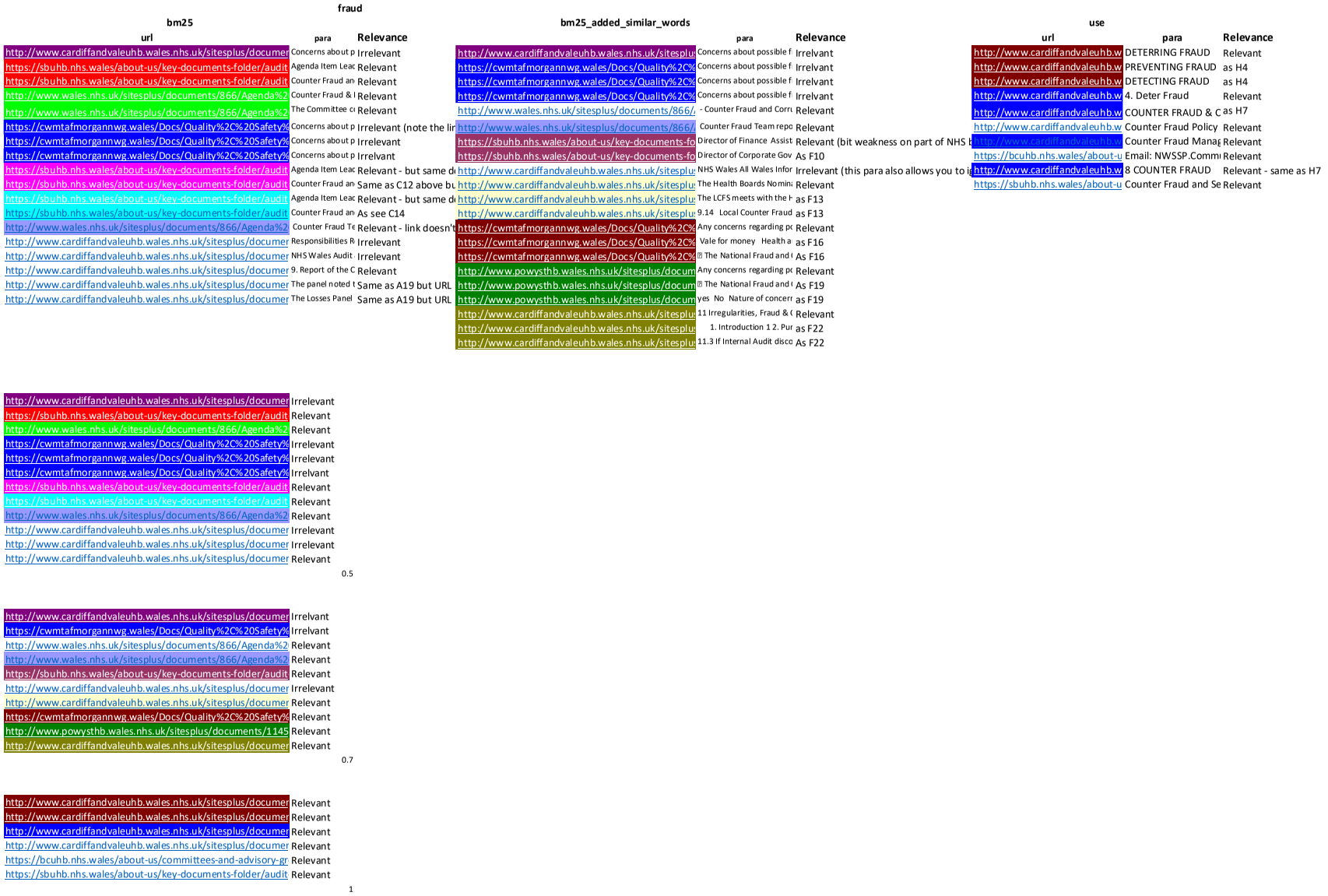


Figure 28: Expert Labelled Data for 'Fraud'

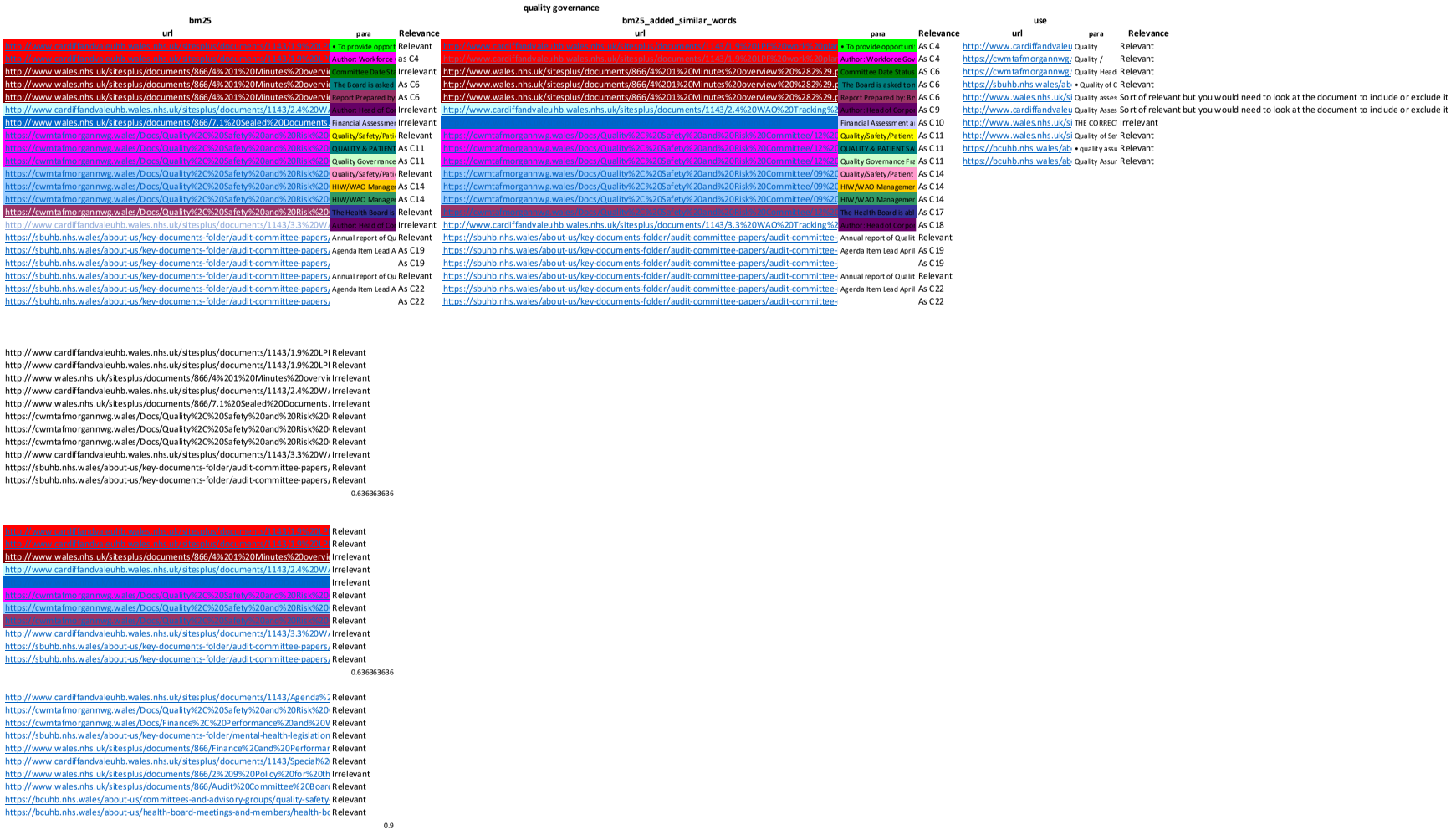


Figure 29: Expert Labelled Data for 'Quality Governance'

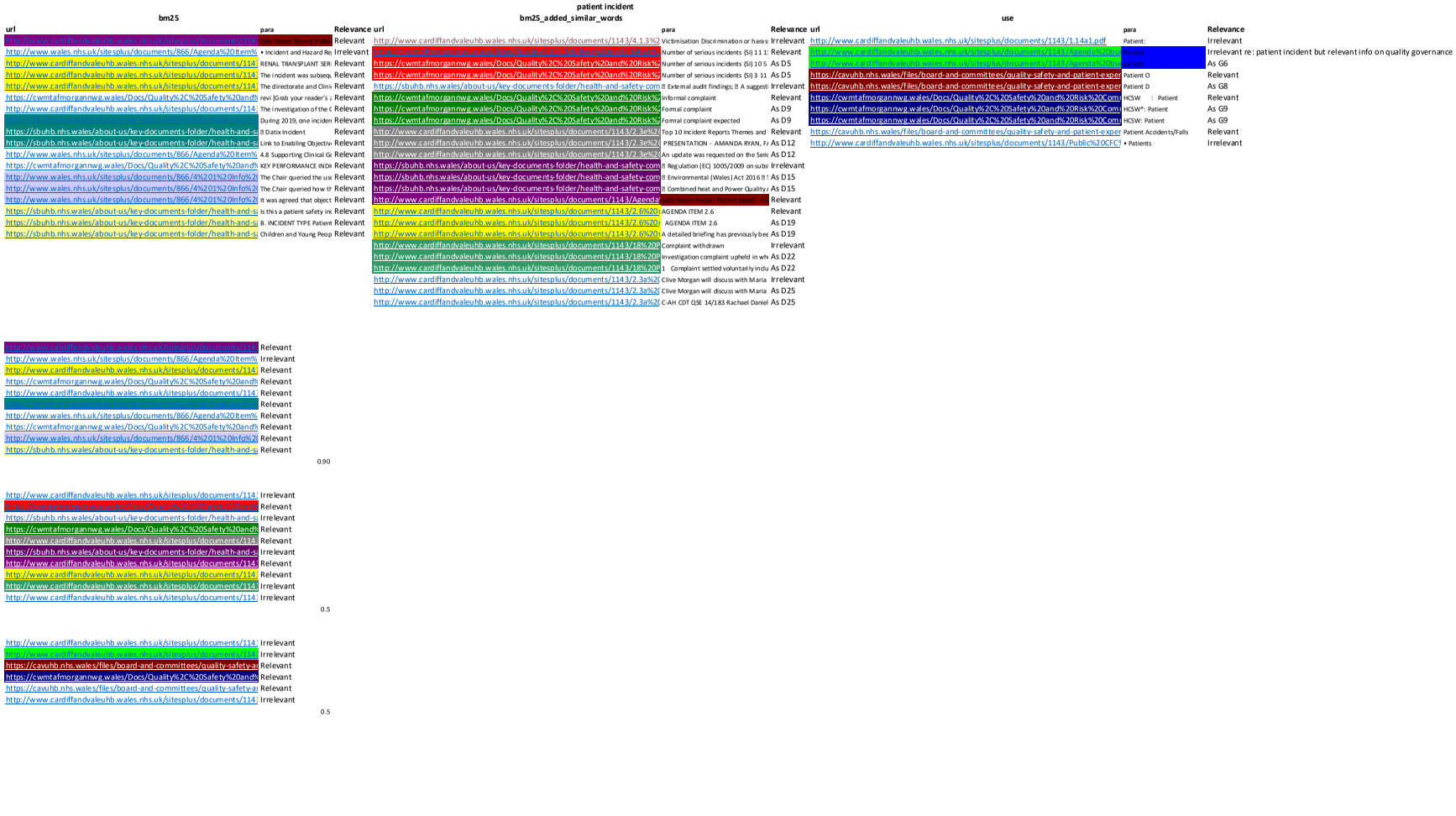


Figure 30: Expert Labelled Data for ‘Patient Incident’

url	committee_type		item_type	
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 3.3 Interi
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 1.6 Ratifi
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	health and safety committee	['Home', ' > Health and Safety Committee']	papers	papers
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 24 February 2015']	papers	papers
http://www.wales.nhs.uk/sitesplus/documents/862/Age	strategy committee	[' > Strategy and Planning Committee Meeting 16 O	agenda	Agenda Special Strate
http://www.wales.nhs.uk/sitesplus/documents/862/Age	finance and performance committee	[' > Integrated Governance Committee Meeting 9 M	agenda	Agenda Integrated Go
http://www.wales.nhs.uk/sitesplus/documents/862/Age	health and safety committee	[' > ', 'Your Health Board']	agenda	Agenda Primary Care /
http://www.wales.nhs.uk/sitesplus/documents/866/Fina	finance and performance committee	Finance and Performance Committee	agenda	Agenda and Papers
https://bcuhb.nhs.wales/about-us/committees-and-advi	health and safety committee	Strategy, Partnerships and Population Health Comn	agenda	agenda-3-9-19
https://sbuhb.nhs.wales/about-us/key-documents-folde	mental health act committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	agenda	0. Agenda - May 2018
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee']	papers	papers
https://bcuhb.nhs.wales/about-us/committees-and-advi	quality and safety committee	Quality, Safety and Experience Committee	final minutes	Approved Minutes
https://bcuhb.nhs.wales/about-us/committees-and-advi	health professionals forum	Healthcare Professionals Forum (HPF)	final minutes	Approved minutes
http://www.wales.nhs.uk/sitesplus/documents/862/Iten	health and safety committee	[' > ', 'Your Health Board']	papers	Item 07a University Pa
http://www.wales.nhs.uk/sitesplus/documents/862/Atta	strategy committee	[' > Strategy and Planning Committee Meeting 16 O	papers	Attachment F Paediat
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 3.7 Discr
http://www.wales.nhs.uk/sitesplus/documents/862/Age	health and safety committee	[' > ', 'Your Health Board']	agenda	Agenda Primary Care /
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 5.1 Finan
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_1.3_Chair'
https://bcuhb.nhs.wales/about-us/committees-and-advi	local partnership forum	Local Partnership Forum (LPF)	final minutes	Approved minutes
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee']	papers	papers
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_2.2_WOD
https://cavuhb.nhs.wales/files/board-and-committees/q	quality and safety committee	quality-safety-and-experience-committee	agenda	https://cavuhb.nhs.wa
https://sbuhb.nhs.wales/about-us/key-documents-folde	mental health act committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	4.4 Appendix 3.pdf (
http://www.wales.nhs.uk/sitesplus/documents/862/Age	finance and performance committee	[' > Integrated Governance Committee Meeting 27 J	agenda	Agenda Integrated Go
http://www.wales.nhs.uk/sitesplus/documents/862/Atta	finance and performance committee	[' > Integrated Governance Committee Meeting 19 J	papers	Attachment 8i - Confir
http://www.wales.nhs.uk/sitesplus/documents/866/PPW	other	Public Partnerships and Wellbeing Committee	agenda	Agenda and Papers
https://sbuhb.nhs.wales/about-us/key-documents-folde	audit committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	2e. Appendix B Extern
https://cavuhb.nhs.wales/files/board-and-committees/q	quality and safety committee	quality-safety-and-experience-committee	papers	https://cavuhb.nhs.wa
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Archived Audit Committee Papers']	papers	Archived Audit Commi
http://www.wales.nhs.uk/sitesplus/documents/862/Iten	health and safety committee	[' > ', 'Your Health Board']	papers	Item 18 Community Pl
http://www.wales.nhs.uk/sitesplus/documents/866/IGC	digital and information management committee	Information Governance Committee	agenda	Agenda and Papers
https://bcuhb.nhs.wales/about-us/committees-and-advi	audit committee	Audit Committee	agenda	agenda-bundle-audit-(
http://www.wales.nhs.uk/sitesplus/documents/866/IGC	digital and information management committee	Information Governance Committee	agenda	Agenda and Papers
http://www.wales.nhs.uk/sitesplus/documents/862/App	health and safety committee	[' > ', 'Your Health Board']	final minutes	Approved PCAC Minut
https://sbuhb.nhs.wales/about-us/key-documents-folde	mental health act committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	agenda	0. Agenda - 8th Febru
https://cavuhb.nhs.wales/files/board-and-committees/q	quality and safety committee	quality-safety-and-experience-committee	papers	https://cavuhb.nhs.wa
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 22 October 2013']	papers	papers
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	health and safety committee	['Home', ' > Health and Safety Committee']	papers	papers
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	1.6 Health and Safety
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_1.5_Comr
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 2.6 Healt
https://bcuhb.nhs.wales/about-us/committees-and-advi	quality and safety committee	Quality, Safety and Experience Committee	final minutes	Approved Minutes
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 26 May 2015']	papers	papers
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_1.5_Comr
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_3.4_2018-
http://www.wales.nhs.uk/sitesplus/documents/862/Iten	health and safety committee	[' > ', 'Your Health Board']	papers	Item 2.5 Remote Pulm
http://www.wales.nhs.uk/sitesplus/documents/862/Iten	health and safety committee	[' > ', 'Your Health Board']	papers	Item 05 Table of Actio
https://sbuhb.nhs.wales/about-us/key-documents-folde	audit committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	agenda	0. May Audit Committ
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	4.2 Singleton Hospital
https://bcuhb.nhs.wales/about-us/committees-and-advi	quality and safety committee	Quality, Safety and Experience Committee	agenda	agenda-bundle-qse-17
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 24 February 2015']	papers	papers
http://www.powysthb.wales.nhs.uk/sitesplus/document	workforce committee	[' > ', "The Board's Committees, Partnership Boards	papers	W&OD_Item_2.4_Org
http://www.wales.nhs.uk/sitesplus/documents/862/Age	strategy committee	[' > Strategy and Planning Committee Meeting 23 Ju	agenda	Agenda Strategy and F
http://www.powysthb.wales.nhs.uk/sitesplus/document	workforce committee	[' > ', "The Board's Committees, Partnership Boards	papers	WF&OD_1.6_Committ
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 3.1 Comr
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 2.11 a - (
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_2.1_Healt
https://bcuhb.nhs.wales/about-us/committees-and-advi	health and safety committee	Strategy, Partnerships and Population Health Comn	final minutes	Approved minutes
http://www.wales.nhs.uk/sitesplus/documents/862/Atta	strategy committee	[' > Strategy and Planning Committee Meeting 16 O	papers	Attachment A Paediat
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 6.3a Prin
https://bcuhb.nhs.wales/about-us/committees-and-advi	health and safety committee	Strategy, Partnerships and Population Health Comn	draft minutes	Draft minutes 5.3.20
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 24 February 2015']	papers	papers
http://www.wales.nhs.uk/sitesplus/documents/862/Age	health and safety committee	[' > ', 'Your Health Board']	agenda	Agenda University Par
https://bcuhb.nhs.wales/about-us/committees-and-advi	health professionals forum	Healthcare Professionals Forum (HPF)	agenda	agenda-for-hpf-13-3-2
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 12 August 2014']	papers	papers
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 2.3 Trans
http://www.wales.nhs.uk/sitesplus/documents/866/MHI	mental health act committee	Mental Health and Learning Disabilities Committee	agenda	Agenda and Papers
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	draft minutes	2.3 Appendix 6 - Draft
https://sbuhb.nhs.wales/about-us/key-documents-folde	audit committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	1e. Action Log.pdf (
http://www.wales.nhs.uk/sitesplus/documents/862/Iten	health and safety committee	[' > ', 'Your Health Board']	final minutes	Item 4 Primary Care A
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	4b. Planning for the A
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	4.1 Manual Handling (
https://sbuhb.nhs.wales/about-us/key-documents-folde	audit committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	agenda	0. Agenda - 19th April
https://bcuhb.nhs.wales/about-us/committees-and-advi	quality and safety committee	Quality, Safety and Experience Committee	papers	Approved Corporate S
https://sbuhb.nhs.wales/about-us/key-documents-folde	mental health act committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	7a. CAMHS Convener
https://cavuhb.nhs.wales/files/board-and-committees/q	quality and safety committee	quality-safety-and-experience-committee	papers	https://cavuhb.nhs.wa
https://bcuhb.nhs.wales/about-us/committees-and-advi	local partnership forum	Local Partnership Forum (LPF)	final minutes	Approved minutes
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - Confirmed Minutes']	final minutes	Audit Committee - Cor
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_2.3_WOD
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 4.4 Fund
http://www.wales.nhs.uk/sitesplus/documents/862/Iten	health and safety committee	[' > ', 'Your Health Board']	papers	Item 17 D Snape Dent
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 22 October 2013']	papers	papers
http://www.wales.nhs.uk/sitesplus/documents/862/Atta	strategy committee	[' > Strategy and Planning Committee Meeting 16 O	papers	Attachment I Paediatr
https://sbuhb.nhs.wales/about-us/key-documents-folde	mental health act committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	2.1 Mental Health Leg
http://www.powysthb.wales.nhs.uk/sitesplus/document	board meeting	[' > ', 'Board Meetings', ' > ', 'Archived Board Papers	agenda	Agenda_Item_2.1b_Ap
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 24 February 2015']	papers	papers
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	2.5 Lockdown Proceed
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 7.1 Struc
http://www.wales.nhs.uk/sitesplus/documents/866/Pub	board meeting	Public Board Meetings 2019	agenda	Agenda and Papers
http://www.wales.nhs.uk/sitesplus/documents/862/Atta	strategy committee	[' > Strategy and Planning Committee Meeting 16 O	papers	Attachment B Paediat
http://www.wales.nhs.uk/sitesplus/documents/862/Atta	finance and performance committee	[' > Integrated Governance Committee Meeting 28 A	papers	Attachment 4 - Table (
http://www.wales.nhs.uk/sitesplus/documents/866/Age	board meeting	None	agenda	Agenda Item 2.4 - Orth
http://www.powysthb.wales.nhs.uk/sitesplus/document	digital and information management committee	[' > ', "The Board's Committees, Partnership Boards	papers	WOD_Item_3.7_Opera
https://cavuhb.nhs.wales/files/board-and-committees/q	quality and safety committee	quality-safety-and-experience-committee	agenda	https://cavuhb.nhs.wa
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 14 April 2015']	papers	papers
https://sbuhb.nhs.wales/about-us/key-documents-folde	mental health act committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	1.4 Mental Capacity A
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	2.11 Health and Safet
https://sbuhb.nhs.wales/about-us/key-documents-folde	health and safety committee	['sbuhb.nhs.wales', 'about-us', 'key-documents-fold	papers	2.11 Health and Safet
http://www.cardiffandvaleuhb.wales.nhs.uk/sitesplus/dc	audit committee	['Home', ' > Audit Committee - 14 April 2015']	papers	papers
	0.86		0.98	

Figure 31: Correctly labelled committee and item types.

Metric	Task	Value	Unit
Cumulative indexing time of primary shards		28.33218	min
Min cumulative indexing time across primary shards		28.33218	min
Median cumulative indexing time across primary shards		28.33218	min
Max cumulative indexing time across primary shards		28.33218	min
Cumulative indexing throttle time of primary shards		0	min
Min cumulative indexing throttle time across primary shards		0	min
Median cumulative indexing throttle time across primary shards		0	min
Max cumulative indexing throttle time across primary shards		0	min
Cumulative merge time of primary shards		9.80015	min
Cumulative merge count of primary shards		21	
Min cumulative merge time across primary shards		9.80015	min
Median cumulative merge time across primary shards		9.80015	min
Max cumulative merge time across primary shards		9.80015	min
Cumulative merge throttle time of primary shards		3.924783	min
Min cumulative merge throttle time across primary shards		3.924783	min
Median cumulative merge throttle time across primary shards		3.924783	min
Max cumulative merge throttle time across primary shards		3.924783	min
Cumulative refresh time of primary shards		0.187167	min
Cumulative refresh count of primary shards		29	
Min cumulative refresh time across primary shards		0.187167	min
Median cumulative refresh time across primary shards		0.187167	min
Max cumulative refresh time across primary shards		0.187167	min
Cumulative flush time of primary shards		0.469233	min
Cumulative flush count of primary shards		9	
Min cumulative flush time across primary shards		0.469233	min
Median cumulative flush time across primary shards		0.469233	min
Max cumulative flush time across primary shards		0.469233	min
Total Young Gen GC time		18.265	s
Total Young Gen GC count		1622	
Total Old Gen GC time		2.81	s
Total Old Gen GC count		46	
Store size		3.372252	GB
Translog size		5.12E-08	GB
Heap used for segments		0.082645	MB
Heap used for doc values		0.008587	MB
Heap used for terms		0.042023	MB
Heap used for norms		0.001648	MB
Heap used for points		0	MB
Heap used for stored fields		0.030388	MB
Segment count		27	
Min Throughput	index-append	22699.82	docs/s
Median Throughput	index-append	23812.61	docs/s
Max Throughput	index-append	23960.77	docs/s
50th percentile latency	index-append	734.3545	ms
90th percentile latency	index-append	836.6826	ms
99th percentile latency	index-append	2675.141	ms
99.9th percentile latency	index-append	3071.174	ms
100th percentile latency	index-append	3211.7	ms
50th percentile service time	index-append	734.3545	ms
90th percentile service time	index-append	836.6826	ms
99th percentile service time	index-append	2675.141	ms
99.9th percentile service time	index-append	3071.174	ms
100th percentile service time	index-append	3211.7	ms
error rate	index-append	0	%
Min Throughput	randomized-nested-queries	19.97	ops/s
Median Throughput	randomized-nested-queries	20	ops/s
Max Throughput	randomized-nested-queries	20.01	ops/s
50th percentile latency	randomized-nested-queries	61.58833	ms
90th percentile latency	randomized-nested-queries	124.2629	ms
99th percentile latency	randomized-nested-queries	161.1374	ms
99.9th percentile latency	randomized-nested-queries	174.9623	ms
100th percentile latency	randomized-nested-queries	186.8907	ms
50th percentile service time	randomized-nested-queries	55.7832	ms
90th percentile service time	randomized-nested-queries	116.8102	ms
99th percentile service time	randomized-nested-queries	134.874	ms
99.9th percentile service time	randomized-nested-queries	143.5715	ms
100th percentile service time	randomized-nested-queries	158.4357	ms
error rate	randomized-nested-queries	0	%
Min Throughput	randomized-term-queries	25.02	ops/s
Median Throughput	randomized-term-queries	25.02	ops/s
Max Throughput	randomized-term-queries	25.02	ops/s
50th percentile latency	randomized-term-queries	4.806757	ms
90th percentile latency	randomized-term-queries	5.271623	ms
99th percentile latency	randomized-term-queries	5.689158	ms
100th percentile latency	randomized-term-queries	13.37233	ms
50th percentile service time	randomized-term-queries	3.760047	ms
90th percentile service time	randomized-term-queries	4.075883	ms
99th percentile service time	randomized-term-queries	4.442302	ms
100th percentile service time	randomized-term-queries	12.08564	ms
error rate	randomized-term-queries	0	%
Min Throughput	randomized-sorted-term-queries	13.88	ops/s
Median Throughput	randomized-sorted-term-queries	13.93	ops/s
Max Throughput	randomized-sorted-term-queries	14.02	ops/s
50th percentile latency	randomized-sorted-term-queries	11065.97	ms
90th percentile latency	randomized-sorted-term-queries	12869.43	ms
99th percentile latency	randomized-sorted-term-queries	13280.64	ms
100th percentile latency	randomized-sorted-term-queries	13364.1	ms
50th percentile service time	randomized-sorted-term-queries	131.7176	ms
90th percentile service time	randomized-sorted-term-queries	238.6923	ms
99th percentile service time	randomized-sorted-term-queries	260.9358	ms
100th percentile service time	randomized-sorted-term-queries	276.2287	ms
error rate	randomized-sorted-term-queries	0	%
Min Throughput	match-all	5	ops/s
Median Throughput	match-all	5	ops/s
Max Throughput	match-all	5	ops/s
50th percentile latency	match-all	4.149556	ms
90th percentile latency	match-all	4.527419	ms
99th percentile latency	match-all	6.349226	ms
100th percentile latency	match-all	18.5085	ms
50th percentile service time	match-all	2.774015	ms
90th percentile service time	match-all	2.930397	ms
99th percentile service time	match-all	4.895581	ms
100th percentile service time	match-all	16.75208	ms
error rate	match-all	0	%
Min Throughput	nested-date-histo	0.72	ops/s
Median Throughput	nested-date-histo	0.73	ops/s
Max Throughput	nested-date-histo	0.73	ops/s
50th percentile latency	nested-date-histo	153036.1	ms
90th percentile latency	nested-date-histo	211797.8	ms
99th percentile latency	nested-date-histo	225124.2	ms
100th percentile latency	nested-date-histo	226579.8	ms
50th percentile service time	nested-date-histo	2725.309	ms
90th percentile service time	nested-date-histo	2811.345	ms
99th percentile service time	nested-date-histo	2868.206	ms
100th percentile service time	nested-date-histo	2953.442	ms
error rate	nested-date-histo	0	%
Min Throughput	randomized-nested-queries-with-inner-hits_default	17.97	ops/s
Median Throughput	randomized-nested-queries-with-inner-hits_default	18	ops/s
Max Throughput	randomized-nested-queries-with-inner-hits_default	18.01	ops/s
50th percentile latency	randomized-nested-queries-with-inner-hits_default	63.81577	ms
90th percentile latency	randomized-nested-queries-with-inner-hits_default	132.5787	ms
99th percentile latency	randomized-nested-queries-with-inner-hits_default	170.5586	ms
99.9th percentile latency	randomized-nested-queries-with-inner-hits_default	193.5492	ms
100th percentile latency	randomized-nested-queries-with-inner-hits_default	197.027	ms
50th percentile service time	randomized-nested-queries-with-inner-hits_default	57.9447	ms
90th percentile service time	randomized-nested-queries-with-inner-hits_default	124.1478	ms
99th percentile service time	randomized-nested-queries-with-inner-hits_default	145.9454	ms
99.9th percentile service time	randomized-nested-queries-with-inner-hits_default	154.1659	ms
100th percentile service time	randomized-nested-queries-with-inner-hits_default	157.7219	ms
error rate	randomized-nested-queries-with-inner-hits_default	0	%
Min Throughput	randomized-nested-queries-with-inner-hits_default_big_size	15.96	ops/s
Median Throughput	randomized-nested-queries-with-inner-hits_default_big_size	15.99	ops/s
Max Throughput	randomized-nested-queries-with-inner-hits_default_big_size	16	ops/s
50th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	126.5733	ms
90th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	227.3218	ms
99th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	314.1484	ms
99.9th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	360.6439	ms
100th percentile latency	randomized-nested-queries-with-inner-hits_default_big_size	378.1823	ms
50th percentile service time	randomized-nested-queries-with-inner-hits_default_big_size	98.03558	ms
90th percentile service time	randomized-nested-queries-with-inner-hits_default_big_size	163.1654	ms
99th percentile service time	randomized-nested-queries-with-inner-hits_default_big_size	190.4598	ms
99.9th percentile service time	randomized-nested-queries-with-inner-hits_default_big_size	201.1779	ms
100th percentile service time	randomized-nested-queries-with-inner-hits_default_big_size	222.7104	ms
error rate	randomized-nested-queries-with-inner-hits_default_big_size	0	%

Figure 32: Es Rally Nested Challenge.