



Assessing Trust in the Web

CM2303 ONE SEMESTER INDIVIDUAL PROJECT – FINAL
REPORT – (40 CREDITS)

AUTHOR: OWAIN CARPANINI (C1315645)

SUPERVISOR: DR FEDERICO CERUTTI

MODERATOR: PROF DAVID WALKER

Abstract

This project has created a system that uses topic modelling techniques to compare news articles to find similarities. Similar articles can be placed into an ontology to answer queries, and share differing levels of trust according to a variety of factors. This can allow users, whether professional analysts or amateurs, to build their own picture of an event/events based on the sources available and the trust held in those sources.

The project was started from a rudimentary knowledge of ontologies, and very little knowledge of machine learning, and through the research and implementation of this project has improved hugely. At the end of this project, a system is achieved that mixes a topic modelling approach with a human assisted segment, to produce a system that allows a user to compare articles and decide whether they should be introduced to the ontology.

Acknowledgements

I would like to thank my supervisor, Dr Federico Cerutti, who has provided fantastic help and support throughout the project. I would have struggled immensely if I did not have any advice for some of the topics in this project.

Additionally, I thank my family and friends for their support whilst writing this dissertation. For helping me get away for much needed breaks throughout the course of this semester, without which I would have been sat at a computer for far too long.

1 Contents

1	Introduction	6
1.1	The Aims/goals of the project	6
1.2	Intended Audience	6
2	Background	7
2.1	Intelligence	7
2.2	Trust	7
2.3	Ontologies	8
2.3.1	What is an Ontology?	8
2.3.2	How are Ontologies created?	9
2.4	Machine Learning	9
2.4.1	Topic Modelling	9
3	Selection of Approach	11
3.1	Sources	11
3.1.1	Broadcasters	11
3.1.2	Print Media	11
3.1.3	Social Media	12
3.1.4	Sources to be used in the Project	12
3.2	Case Studies	13
3.2.1	1 – UK politics – Politics news since June 2016	13
3.2.2	2 – Sport – The Premier League in the 2016/17 Season	14
3.3	Ontology Design	15
3.3.1	Competency Questions	16
3.4	System Design	18
3.4.1	Python	18
3.4.2	Ontology Manipulation	18
3.4.3	Topic Modelling for Comparing Articles	19
4	Implementation	21
4.1	Ontology	21
4.1.1	Implementing Classes	21
4.1.2	Implementing Properties	22
4.1.3	Populating the Ontology	23
4.2	System	23

4.2.1	Installing Python Libraries.....	23
4.2.2	Implementing the System.....	24
4.2.3	Manipulating the Ontology.....	24
4.2.4	Topic Modelling using Gensim.....	29
5	Results and Evaluation.....	33
5.1	Testing the Ontology.....	33
5.2	Testing the System	36
5.2.1	Test 1: getContext(inputurl)	36
5.2.2	Test 2: getAllUrlsFromOntology()	36
5.2.3	Test 3: addToOntology(newurl, filename, oldurl="").....	37
5.2.4	Test 4: getText(url).....	37
5.2.5	Test 5: singleInputTokenize(text)	38
5.2.6	Test 6: createModel(tokeisedtexts, numoftopics)	39
5.2.7	Test 7: userCompare(urltocompare)	39
5.2.8	Test 8: LDACompare(urllist, currentTokens, newurl)	40
5.2.9	Test 9: simpleArticleComparison(urllist, currentTokens, newurl).....	41
5.3	Results	42
5.3.1	The Experiment.....	42
5.3.2	Inputs	42
5.3.3	Output.....	43
5.4	Evaluation.....	45
6	Future Work.....	46
6.1	Further Troubleshooting of LDA querying	46
6.2	Dynamic Creation of RDF Triples.....	46
6.3	Implement the Second Case Study	46
6.4	Introduce Social Media to the Ontology	46
6.5	Use Web Scraping to Populate the Ontology	46
6.6	Integrate with NewsAPI	47
7	Conclusion	48
8	Reflection.....	49
9	Bibliography.....	50
10	Appendix.....	53
10.1	Appendix A: Output from new, similar link.....	53

10.2	Appendix B: Output from new, identical link.....	56
10.3	Appendix C: Output from dissimilar link	59

Table of Figures

Figure 1: Representing Trust – Table taken from Table 2.3 [1]	7
Figure 2: Class Hierarchy of Pizza Ontology [5]	8
Figure 3: Object Property Hierarchy of Pizza Ontology	8
Figure 4: Example of a BBC News article [14]	11
Figure 5: Example of a Guardian News article [15]	12
Figure 6: Example of a Social Media Post [16]	12
Figure 7: Table of Sources	13
Figure 8: Table of Sources - Secondary case study	15
Figure 9: Initial diagram of ontology	15
Figure 10: Final diagram of ontology	16
Figure 11: A set of example triples detailing two individuals of class 'Class'	18
Figure 12: Pseudocode for SPARQL to get links from ontology	18
Figure 13: Pseudocode to show process for simple article comparison	20
Figure 14: The final class hierarchy of the ontology	21
Figure 15: The final data property hierarchy of the ontology	22
Figure 16: The final object property hierarchy of the ontology	22
Figure 17: SPARQL query to extract URLs from ontology	25
Figure 18: The AnswerBrexItImpactWalesBBC individual in Turtle format	26
Figure 19: The BBC:BrexItWalesImpact individual in Turtle format	27
Figure 20: The correct top level classes are present	33
Figure 21: The correct subclasses of query are present	33
Figure 22: The correct individuals for queries are present	33
Figure 23: The correct subclasses of Sources are present	33
Figure 24: The correct subclasses of social media are present	34
Figure 25: The correct subclasses are present in Websites	34
Figure 26: The correct individuals are present in the Trust class	35
Figure 27: An example of an answer individual linking to queries and sources	35
Figure 28: The correct subclasses are present in the Answers class	35
Figure 29: Result of Test 1 – getContext	36
Figure 30: Result of Test 2 - getAllUrlsFromOntology	36
Figure 31: Result of Test 3 - addUrlToOntology	37
Figure 32: Result of Test 4 - getText	37
Figure 33: Result of Test 5 - singleInputTokenize	38
Figure 34: Result of Test 6 – createModel	39
Figure 35: Result of Test 7 – userCompare	40
Figure 36: Result of Test 8 – LDACompare	41
Figure 37: Result of Test 9 – simpleArticleComparison	42
Figure 38: Eleventh link in large model	43
Figure 39: Model of similar link being compared	43
Figure 40: Model of identical link being compared	44
Figure 41: Model of dissimilar link being compared	44

1 Introduction

1.1 The Aims/goals of the project

The internet is a massive and rich source of information. By searching the internet, a user can find information on any subject for almost any level of knowledge – from beginner to expert. This information can be used to answer the user's question or simply provide background knowledge and context. An enduring problem for users is how to estimate the amount of trust that can be placed on information gained from the internet. Due to the ease with which information can be published on the internet, this information could be inaccurate or completely false. The amount of trust placed in a source of information can vary for different reasons:

- The reputation of a source.
- The language with which a source is written. For example, factual language will be more trustworthy than purely emotive language.
- The audience a source is targeted at. For example, a source targeted at experts in a field may be more trustworthy than a source targeted at the general public.

Over the course of this project, a method to provide trustworthy information back to a user by providing an ontology of trust across sources will be investigated. Additionally, the project will investigate the use of machine learning techniques to identify similarity between articles. This will enable a system to find which article is a better fit to answer a particular query. This will be compared against a simple (no machine learning) similarity check to assess the effectiveness of such a method.

The aim of this project is to provide a foundation for investigations into specific case studies, such as analysing the results and effects caused by political change, such as the recent vote for the United Kingdom to leave the European Union. Depending on time constraints, the project will also aim to look into a slightly smaller case study – looking at some simpler information such as the results in a sports league over a short period of time. Case studies like these will show the feasibility of far more complex case studies

These will show the feasibility of such future studies by the end of this project, which could allow for a system that aggregates results on controversial topics. For example, the impact humans have on the Earth through global warming, or the efficacy of new vaccinations.

1.2 Intended Audience

The intended audience of this project is not casual users, but users who need to ensure the information they receive can be trusted. A good example of this could be an intelligence analyst, trying to pull together a variety of sources to gain an accurate picture of current events. Alternatively, it could be a campaign director for a political party, who could be trying to assess the effect their campaign is having on the general public. Therefore, the audience for this project is anyone who requires a method of querying and aggregating results for reliable intelligence gathering.

2 Background

2.1 Intelligence

Before carrying out an investigation into how trust can be linked to information, we must first understand intelligence and its uses, and how it can be related to trust and information. Prunckun defines intelligence as knowledge, or insight, that has been acquired through actions and processes. He says that intelligence can be equated to the ability to 'reduce uncertainty' in an area. Intelligence helps inform decisions that need to be made when some parameters are unknown [1, p. 3]. We are interested in open source intelligence. As stated in the book 'Automating Open Source Intelligence', open source intelligence is intelligence created through the use of publicly available data [2].

This project will research the process of acquiring open source intelligence from news articles. In this case, the action or process behind acquiring a piece of intelligence is downloading an article, and comparing it to others in a database to answer a query. Intelligence is gained where the article found can provide new insight into a topic. A query is a statement that aims to find out some intelligence about a subject. It could be a simple 'yes' or 'no' question, or a more complex statement that could return multiple pieces of intelligence covering different viewpoints on a topic.

2.2 Trust

In Chapter 2 of the book Handbook of Scientific Methods of Inquiry for Intelligence Analysis [1, p. 30], the author talks about applying information reliability codes to intelligence that has been gathered. Depending on the past reliability of a source, it is given a code that represents its reliability. In this project, we will apply the same principle to our sources of information. The same codes will be used that can be found in the book (Table 2.3) and are shown below as Figure 1:

Code	Description	Estimated Probability of Trust
A	Completely Reliable	100%
B	Usually Reliable	80%
C	Fairly Reliable	60%
D	Not Usually Reliable	40%
E	Unreliable	20%
F	Cannot be Judged	50%

Figure 1: Representing Trust – Table taken from Table 2.3 [1]

Therefore, we can say that for this project, the trust in a source is the level of reliability that source commands. A trustworthy source will have a high estimated probability of trust, whereas a source that cannot be trusted (and hence is unreliable) will have a low estimated probability of trust. In this project, we will arbitrarily define the trust for each source when implementing the trust in an ontology.

2.3 Ontologies

2.3.1 What is an Ontology?

A computational ontology, as defined by Guarino, Oberle and Staab, is a means to model the structure of a system [3]. Essentially, this means that an ontology describes concepts, using two main structures: Classes, and Properties. Classes describe the concepts themselves, and can provide a hierarchical class structure that represent a whole set of related concepts. Classes are populated with Individuals, which are members of a class or subclass. Properties represent the attributes of the individuals described by concepts, and can be either object properties or data properties. An object property describes attributes between two or more individuals, and a data property describes values attributed to an individual. The University of Manchester have an excellent tutorial to introduce ontologies, representing pizzas and related ingredients as a set of classes and properties [4]:

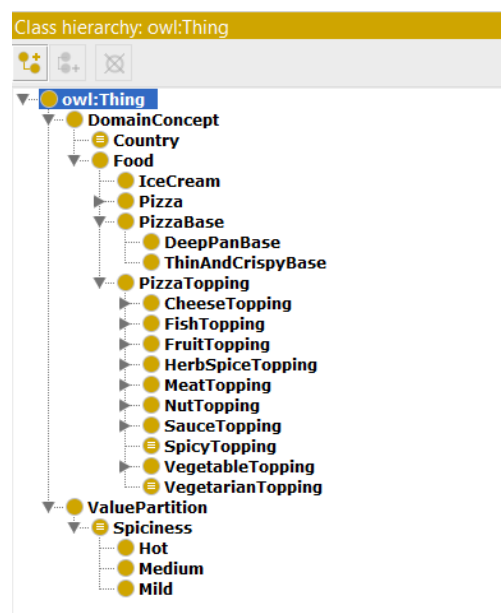


Figure 2: Class Hierarchy of Pizza Ontology [5]

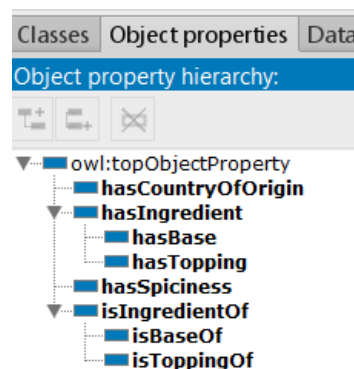


Figure 3: Object Property Hierarchy of Pizza Ontology

2.3.2 How are Ontologies created?

Ontologies can be created through a number of methods. For example, they can be written using an XML syntax such as OWL (Web Ontology Language), as 'triples' using RDF (Resource Description Framework), or through an Ontology editor such as Protégé [6].

RDF is an important part of the Semantic Web, the W3C standard for linked data on the internet [7]. It is a model for representing data, which it does using the concept of triples. A triple is a set of three pieces of information that represent a node in the model. A triple is made up of a subject, a predicate and an object:

- The subject is the resource, or individual, being described by the triple.
- The predicate is the relationship between the subject and the object. In an ontology, this is the data or object property that is to be represented.
- The object is the resource, or individual, being related to the subject. In an ontology, depending on whether the predicate is a data or object property, the object could be either a resource or a data-typed value.

In a RDF model, every resource is represented by a URI, or Uniform Resource Identifier. These look identical to URLs, but do not always point to webpages, as they are purely a unique identifier for the resource. Each subject, predicate, and object defined in a set of triples must have a URI assigned to it. [7]

A RDF model can be queried using SPARQL. SPARQL (SPARQL Protocol and RDF Query Language) is a language used to query and manipulate RDF triples stored in a RDF model [8]. When using SPARQL to query for data, the main components of a query are 'Select' clause and a 'where' clause.

2.4 Machine Learning

Machine Learning is the use of computers and statistics to solve a problem. A machine learning algorithm completes tasks by using previous experience of completing that task to improve its performance. As such, machine learning algorithms will typically run a number of iterations on a dataset to calculate the best possible result. A machine learning algorithm is able to learn from data [9].

2.4.1 Topic Modelling

Topic Modelling is a form of machine learning in which a statistical model is created to learn about topics that are present in a series of articles or documents [10]. A form of topic modelling is Latent Dirichlet Allocation, or LDA. LDA is a model of the distributions of topics in a set of model, as a random mixture of words [11]. The algorithm takes in a set of documents, and returns a trained model, containing a number of topics. These topics correspond to the document taken in, so each document will correspond to a single topic.

When passing text into a LDA model, it first needs to be split into individual words. A good algorithm, will also remove 'stop words' from the text, and then stem the words. Stop words are simply very common words that do not add value to text. An excellent list of stop words can be found at ranks.nl [12]. When these words are removed from the text, the remaining words should be far more focused on the meaning and context of the text. Once

removed, the remaining words should have a stemming algorithm applied. The process of stemming is to 'stem' the suffixes of words in order to have an identical base form for related words [13]. For example, the words 'surprise' and 'surprising' when stemmed would both become 'surpris'. Whilst this makes no sense to a human reader, to a computer this would enable both 'surprise' and 'surprising' to be represented as a single weighting in a topic model.

3 Selection of Approach

In this section, we will decide on an approach for carrying out this project. The project will consist of four main parts. These parts are:

- Defining potential sources to be used in case studies.
- Describing some case studies.
- An ontology of queries, sources, and trust.
- Some scripts to manipulate the ontology and carry out comparisons between articles.

In this step, we will define the sources and case studies to be used in the project. We will also design an ontology of trust, which will form the backbone of this project. Finally, we will look at the methods to be used to create a prototype system that uses our ontology.

3.1 Sources

In the project, a source is anywhere which holds information that can be used to answer queries in the case studies that will be defined. Sources can usually be divided into different categories, depending on the organisation that owns the source. As mentioned above, sources may or may not be trustworthy – the trust of a source can depend on many factors, such as where it was published or who published it. For the project, we will look at sources in several different categories:

3.1.1 Broadcasters

Broadcasters are organisations that broadcast their information on at least one television channel. They usually create online articles that link to stories that have been or will be broadcast. Additionally, larger broadcasting organisations also write standalone articles such that may not be broadcasted on a television channel.

An example of this category of source is a news article from the BBC:



Figure 4: Example of a BBC News article [14]

3.1.2 Print Media

The Print Media are organisations that publish newspapers and magazines. They post articles from their published copies onto their website. Newspapers are typically either local or cover news from the whole of the UK. Specialist magazines may be focused on a single form of news, such as technology news over a regular period.

An example of this category of source is a news article from the Guardian:



Figure 5: Example of a Guardian News article [15]

3.1.3 Social Media

Social Media allows anyone to post about a topic in various formats. Users of social media can be from the general public or 'official' representatives. For example, a political party will have their own social media pages to share information about their party, and a member of the public may have a page on which they share their opinions on a topic raised by that party.

An example of this category of source is a Tweet by the Express:



Figure 6: Example of a Social Media Post [16]

3.1.4 Sources to be used in the Project

There are several sources that could be useful for the project. The following are sources found in the book 'Open Source Intelligence Techniques' [17]:

- Facebook
- Twitter
- LinkedIn
- Reddit
- Instagram

There are also plenty of other sources that can be used in the project, other than the five just mentioned. These are primarily in the categories of Broadcasters and Print Media, rather than Social Media which is represented in the above list.

All the potential sources have been collated in the table below:

Source	URL	Category
BBC	http://www.bbc.co.uk/	Broadcasters
ITV	http://www.itv.com/news/	Broadcasters
Channel 4	https://www.channel4.com/news/	Broadcasters
The Economist	http://www.economist.com/	Print Media
The Financial Times	https://www.ft.com/	Print Media
The Guardian	https://www.theguardian.com/uk	Print Media
The Independent	http://www.independent.co.uk/	Print Media
The Telegraph	http://www.telegraph.co.uk/	Print Media
The Times	https://www.thetimes.co.uk/	Print Media
The Daily Mail	http://www.dailymail.co.uk/home/index.html	Print Media
The Mirror	http://www.mirror.co.uk/	Print Media
The Daily Express	http://www.express.co.uk/	Print Media
Facebook	https://www.facebook.com/	Social Media
Reddit	https://www.reddit.com/	Social Media
Twitter	https://twitter.com/	Social Media
LinkedIn	https://www.linkedin.com/	Social Media
Instagram	https://www.instagram.com/?hl=en	Social Media

Figure 7: Table of Sources

3.2 Case Studies

As mentioned above, this project will be based on one main case study. A second case study will also be described as a means of demonstrating a slightly different path the project could take. These will cover completely different topics to ensure that the ontology can be tested under different scenarios.

3.2.1 1 – UK politics – Politics news since June 2016

3.2.1.1 Domain

This Case study will be the main case study for this project, covering a much broader area. It will look at news in politics since June 2016. This date is important as on the 23rd June 2016, the people of the United Kingdom took part in a vote on the UK's membership of the European Union. This was referred to as the Brexit vote. As a decision that will affect every aspect of life in the UK, the vote and the subsequent result and following events have received substantial coverage in the media.

The case study could cover a wide range of events and areas, with the main ones being the following:

- The vote itself – both the build-up and the result.

- The effect the vote will have on different regions and people in the UK
- The effect the vote will have on the UK itself.
- The Prime Minister stepping down after the result and being replaced.
- The changes in the political landscape over the course of the period described.

As mentioned above, these areas have all received significant coverage in the media, across a variety of sources. In this case study, we will need to define the sources from which we will gather information so that we can then use these in our ontology.

This case study is useful to us as an example of a ‘real-world’ use of a system such as the one researched in this project. An analyst may want to query a system for information about the vote, and this could be for a variety of reasons. The analyst may want to simply find out news or information about the vote, such as who is leading the vote or the results of the vote. On the other hand, the analyst may wish to do a more in depth analysis, by querying the system for underlying information behind the vote, such as the effects the vote could have on voters.

3.2.1.2 Sources of Information Considered

As the main case study for the project, all the sources named in the Sources section could be relevant. As such, see Figure 7 for a list of the sources that will be included in this case study.

3.2.1.3 Queries

Below are the queries that will be implemented for this case study:

1. What is the impact of Brexit on EU funding in Wales?
2. What was the final result of last year’s Brexit Referendum?
3. How did Wales vote in the Brexit Referendum?
4. Who replaced David Cameron as the Prime Minister of the United Kingdom?

3.2.2 2 – Sport – The Premier League in the 2016/17 Season

3.2.2.1 Domain

This case study is the secondary case study for the project. This case study will be a smaller case study, smaller in scope than the above case study. This will allow for the evaluation of

This case study will look at the fixtures, results and table from the 2016/17 season of the premier league. The Premier League is the top level of football in England, at the top of a pyramid containing all professional football teams in the country, including four teams from Wales. The league consists of twenty teams, who play each other both home and away. At the end of the season, the top four teams are rewarded with qualification to the European Champions League, and the bottom three teams are relegated to the next tier of the league system. This case study will focus on the teams within the league, as a means of determining their position in the table at a certain point, or assessing which teams could finish where. The case study could be useful to anyone interested in following the league casually, or even to those who make use of this kind of data in a professional application.

3.2.2.2 Sources of Information Considered

As this is a smaller, secondary case study, the number of sources considered will be reduced. As such, the list of sources can be seen in the following table:

Source	URL	Category
BBC	http://www.bbc.co.uk/	Broadcasters
The Guardian	https://www.theguardian.com/uk	Print Media
The Independent	http://www.independent.co.uk/	Print Media
The Telegraph	http://www.telegraph.co.uk/	Print Media
The Daily Mail	http://www.dailymail.co.uk/home/index.html	Print Media
The Mirror	http://www.mirror.co.uk/	Print Media
Facebook	https://www.facebook.com/	Social Media
Twitter	https://twitter.com/	Social Media

Figure 8: Table of Sources - Secondary case study

3.2.2.3 Queries

These are the queries that sources in this case study will try and answer. As this is a secondary case study, the queries have been kept simple. This will ensure that the ontology can be kept simple when dealing with this case study – no changes will need to be made from the implementation for the main case study.

1. Which team was at the top of the Premier League at Christmas?
2. Which team has scored the most goals so far in the season?
3. Which teams are most likely to finish in the top 4 places of the league at the end of the current season?

3.3 Ontology Design

As described in the Background, the ontology will be developed using Protégé, a tool developed at Stanford University for the creation and updating of ontologies [6]. It allows for the definitions of classes, to represent individuals, and properties. Properties can be either object properties or data properties. Object properties represent relationships between individuals in an ontology, and data properties tie individuals to data as 'Literals'. A Literal is a pair containing an item of data and the datatype of the data.

The ontology to be created in this project will be an ontology of sources, trust and queries. The ontology will be used alongside the case studies shown above to populate it with sources and queries.

See the below figure of the initial conceptual diagram on which this ontology is based:

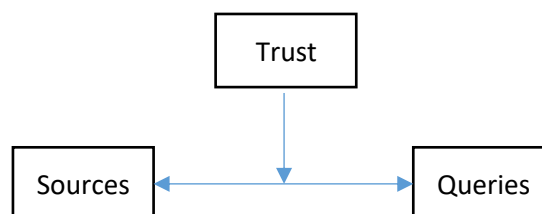


Figure 9: Initial diagram of ontology

This diagram shows the three classes that will be needed for the ontology. The sources and queries classes will have a many to many relationship, where more than one source could be related to a single query and vice versa. The trust to be shown in the individuals of the ontology will be represented by a property linking trust to an instance between the other two classes.

However, this initial model contains no way of answering any queries. To remedy this, an extra class Answers will be implemented in the ontology. This extra class will bring the other three classes together, so that trust can be varied on answers independently of the source it is linked to. The figure below shows the final conceptual model for the ontology:

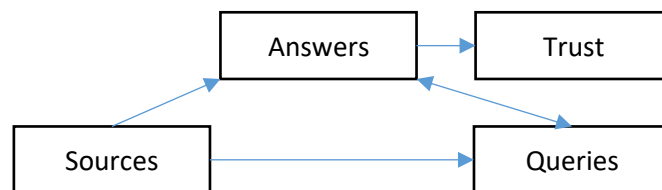


Figure 10: Final diagram of ontology

3.3.1 Competency Questions

With this final model, the next step is to define a set of competency questions. Competency questions are conditions that state what the ontology will and will not be capable of [18]. Similar to the initial phase of application or system design, they are essentially the requirements the ontology must meet. In this case, in addition to being the requirements, they are also the design of the ontology. This is because these competency questions cover every use of the ontology, and through the use of Protégé the ontology can be created by using the questions as definitions for the classes and properties. In the Testing section of this report, we will evaluate our final ontology against these competency questions to ensure that it has met the requirements of the project.

1. There are classes that represent the following:
 - a. Queries
 - b. Sources
 - c. Trust
 - d. Answers to queries
2. The class Queries contains subclasses for each case study we wish to represent:
 - a. A Politics case study for queries about politics.
 - b. A Sports case study for queries about sports.
3. Each subclass of Queries will contain the following individuals representing a query: (For now we are populating only the politics subclass)
 - a. What was the final result of last year's Brexit Referendum?
 - b. What is the impact of Brexit on EU funding in Wales?
 - c. How did Wales vote in the Brexit Referendum?
 - d. Who replaced David Cameron as the Prime Minister of the United Kingdom?
4. The class Sources contains a subclass for every type of source we wish to represent:

- a. Social Media
 - b. Websites
- 5. The subclass SocialMedia contains subclasses to represent social media sources
 - a. Facebook
 - b. Twitter
 - c. Reddit
- 6. The subclass Websites contains subclasses for websites we wish to represent:
 - a. There is a Broadcasters subclass for websites of television broadcasters
 - b. There is Print Media subclass for websites of newspapers and magazines
- 7. The Broadcasters subclass contains subclasses to represent the following broadcasters:
 - a. BBC
 - b. ITV
 - c. Channel 4
- 8. The Print Media subclass contains subclasses to represent the following newspapers/magazines:
 - a. The Guardian
 - b. The Telegraph
 - c. The Times
 - d. The Economist
 - e. The Financial Times
- 9. Every individual stored within the Source class or one of its subclasses represents one of the following:
 - a. Article
 - b. Post
 - c. Page
 - d. (Social Media) Account
- 10. The Trust class contains only individuals to represent the level of trust in a source. This consists of six individuals (see Figure 1 or [1, p. 30]):
 - a. Completely Reliable
 - b. Usually Reliable
 - c. Fairly Reliable
 - d. Not Usually Reliable
 - e. Unreliable
 - f. Cannot Be Judged
- 11. The Answers class contains the answers to instances of queries contained in the Queries class.
- 12. The individuals in the Answers class link individuals from the Sources, Queries and Trust classes to create answers to a query.
 - a. Each answer is linked to a single source, so each query can have multiple answers from different sources.
- 13. The Answers class contains the following subclasses:
 - a. A subclass containing all complete answers.
 - b. A subclass containing templates to represent generic answers.

3.4 System Design

The final part of this section is to decide how to approach the problems of manipulating our above ontology, and how to use topic modelling to compare articles. Therefore, this section covers the approach that will be taken to develop this system, with an overview of the main libraries that will be required to implement it.

3.4.1 Python

Python will be used to implement this system. This is because of Python's relatively simple syntax and the wide array of available modules, both built-in and from third parties. This ease of use will allow for rapid prototyping of a system, which is just that – a prototype as a proof of concept for using an ontology to assess trust.

3.4.2 Ontology Manipulation

To implement functions for manipulating our ontology, there is one library that seems to cover all the areas required: RdfLib [19]. These areas are to be able to load an ontology into python, query the ontology for data, and add new individuals to the ontology. The RdfLib functionality covers these in great detail. It uses a custom graph object as the main focus for the library, which represents the ontology as a series of RDF triples. Once a graph has been loaded, there are many operations which can be executed to manipulate an ontology. These include:

3.4.2.1 Graph()

This is the main function of RdfLib. The function is used to instantiate a new graph object, which can then be populated with an existing ontology or with new triples. A graph can have an ontology loaded into it with the 'parse()' function, which takes either an Ontology URI, or a filename as input. This will then populate the graph object with the contents of that ontology, in the form of triples.

3.4.2.2 Query()

This function of RdfLib uses SPARQL queries to carry out a wide range of tasks. The two main tasks we may need for the project are selecting data, used in the same manner as the select clause in SQL, and inserting data into the ontology. Inserting data using SPARQL may not be necessary, due to the 'add()' function available in RdfLib, but using the select clause will be very useful for collecting the links from the ontology. To collect links from the ontology, a SPARQL query similar to the below will be required:

```
(Individual1, hasLink, Link1)
(Individual2, hasLink, Link2)
(Individual1, hasClass, Class)
(Individual2, hasClass, Class)
```

Figure 11: A set of example triples detailing two individuals of class 'Class'

```
Prefix: URI <Our ontology URI>
SELECT link
WHERE <An individual> URI: hasLink link
```

Figure 12: Pseudocode for SPARQL to get links from ontology

This query would return a link for every individual in the ontology that has a property named 'hasLink'. The subject 'An Individual' is another variable, that is not returned. As 'hasLink' is the only part of the triple concretely named, the query will find every triple that contains the predicate 'hasLink', and then return the link for those triples. In this case, 'Link1' and 'Link2' would both be returned by the query.

3.4.2.3 *Add()*

This function takes in a triple and inserts it into the ontology. As mentioned in Background, a triple consists of a subject, a predicate, and an object. Each of part of the triple will need to be defined as either part of the ontology (as a URIRef or Literal), or use a built-in namespace, such as RDF or FOAF (Friend of a Friend). The built-in namespaces represent World Wide Web Consortium standard schemas for the semantic web. An example of this is 'RDF.type', which is a built-in property to RDF, which states the class of which an individual is an instance.

3.4.3 Topic Modelling for Comparing Articles

To implement article comparison in python, there are a couple of libraries that will be appropriate. These are Gensim, a very comprehensive topic modelling library [20], and the Natural Language Toolkit (NLTK), a library for working with natural language input [21].

In the Gensim library, the most appropriate functions for this project will be the LDA model function. This takes in a corpus of words, and returns a model of all the topics in the corpus.

Gensim allows for querying a model to return related topics. This is useful for the project as it will be comparing a new article against other links in a model. By using this query function, the system will be able to look up an article in the model, extract it, and compare it against the new link.

From Gensim, the main modules required are the corpora and models modules. These provide functions for creating a corpus of documents, and creating models of the created corpus, respectively.

3.4.3.1 *Corpora*

This module provides functions for creating a corpus of input documents, as well as a custom dictionary implementation that stores individual words alongside a unique integer ID [22]. Functions that may be useful are 'doc2bow', and 'add_documents'.

'Doc2bow' converts an input document to Gensim's bag of words format, which is the format required for use as a corpus for a model. The document must already be pre-processed – as discussed in the Background, this consists of tokenising the document, removing stop words and then stemming the remaining words in the document.

'Add_documents' adds new documents to a dictionary object. This will be useful when adding creating a new dictionary, in order to import articles downloaded from the ontology into the dictionary.

3.4.3.2 Models

This module provides functions for creating a new topic model from an input corpus [23]. The important functions here are 'LdaModel', 'print_topics', and 'update'. 'LdaModel' is a constructor that creates a model based on an initial starting corpus. As a constructor, it returns an object that contains a new LDA model. This object can then be manipulated by other functions defined in Gensim, and by any other implemented Python code.

'Print_topics' simply outputs a list of all topics in the model, and the words contained within each topic. This will be useful when implementing a method to allow for human analysis of a model. Finally, the 'update' function can be used to update a model to include new documents in the corpus. This will be useful when updating the corpus with new links from the ontology.

3.4.3.3 Pre-processing documents

Before inserting documents into a corpus, they must first be pre-processed. As described in the background, this involves three steps:

- Tokenizing the document
- Removing stop words from the document;
- Stemming each token from the document.

To implement tokenizing and stemming, the Natural Language Toolkit will be used, as stated above. This will require two of its modules: 'tokenize' [24] and 'stem' [25]. These each implement some functions that together allow for the efficient pre-processing of the documents. The 'stem' module implements a stemming algorithm called 'Porter Stemming' [26]. This was published by Martin Porter in 1980, and is still applicable today. The algorithm looks at suffixes of words, and removes them, so that only common 'stems' of words remain.

3.4.3.4 Algorithm for Simple Article Comparison

This project will also implement a simple algorithm to compare the words in two documents. The algorithm will take each word from an article, and check for its presence in the article against which it is being compared. This will be used to compare output against the output of our final system.

For the simple comparison, some pseudocode is required:

```
Remove duplicate words from new article
Remove duplicate words from original article

For word in new article
    If word in original article
        Increment matches
Percentage similarity = ( matches / length of new article )

If similarity > similarity limit
    New article is similar to old article
```

Figure 13: Pseudocode to show process for simple article comparison

4 Implementation

In this step, the implementation of the project will be discussed. Any problems that arise during the development of the ontology and system will be discussed here. This discussion will consist of a description of the problem found and a description of how the problem was repaired.

4.1 Ontology

There were three main steps to implementing the ontology: First was to implement the classes required, to represent the different concepts in the ontology. Next was to implement the data and object properties that act as relationships for individuals and classes in the ontology. The final step was to populate the ontology with individuals that would provide the knowledge represented in the structure of the ontology.

4.1.1 Implementing Classes

This was a straightforward process. As these had been defined in the competency questions, this was a relatively quick process. However, I realised during the implementation that the ontology would require the use of templates and generic classes to represent individuals where not all information was known. For example, if a new article is entered into the ontology, there would have been no way to set a default level of trust in case there was no obvious level at which this should be set. To avoid this problem, two subclasses were added to the Answers class (represented as 'AnswersToQueries') in the ontology. These were 'AnswersToQueriesImplemented' and 'AnswersToQueriesTemplate'. 'AnswersToQueriesImplemented' is a class representing every answer to a query that has been successfully implemented. 'AnswersToQueriesTemplate' is a class representing a set of individuals that are defined as templates for answers using common sources. These allow the user to assign a template to an answer, which in turn represents a set of 'default' information in the new individual. This can be overridden when appropriate by assigning new properties in the new individual for known information.

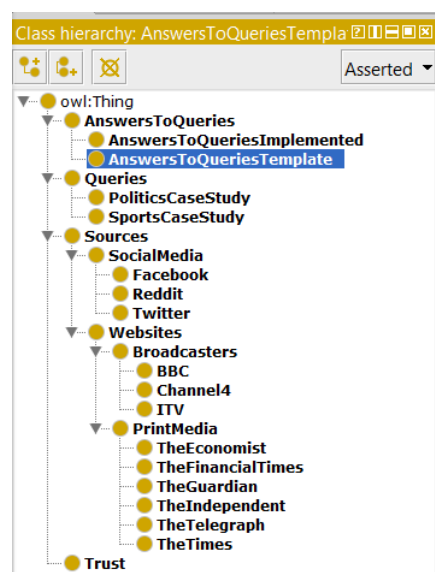


Figure 14: The final class hierarchy of the ontology

4.1.2 Implementing Properties

Once the classes of the ontology had been created, it was time to implement the data and object properties required for the ontology.

4.1.2.1 Data Properties

The data properties in this ontology are to represent the data attributed to each individual stored in the ontology. In this ontology, four data properties have been implemented in the data property hierarchy. Each of these properties represents data of a specific datatype. They are:

- 'autoAdded' – A property representing a Boolean value, used to state whether an individual was inserted from the system or manually. True is the value when an individual has been added automatically.
- 'context' – A property representing a String value, used to describe the concise context given to an individual. For example, if an individual was related to a query about the impact of the Brexit vote on Wales, the context value may read 'BrexitImpactWales'.
- 'description' – A property representing a String value, used to give a description of an individual. For example, if an individual was representing a query described with the context 'BrexitImpactWales', the description may read 'What is the impact the Brexit vote will have on Wales?'.
- 'url' – A property representing a URL value, used to give the URL of an individual representing a Source subclass.

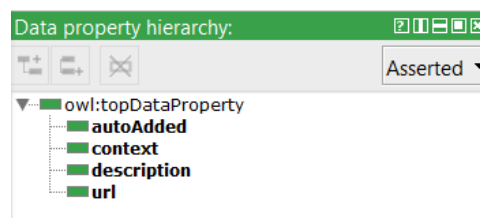


Figure 15: The final data property hierarchy of the ontology

4.1.2.2 Object Properties

The object properties in this ontology are to represent the relationships between individuals stored in the ontology. There are seven object properties implemented in the object property hierarchy:

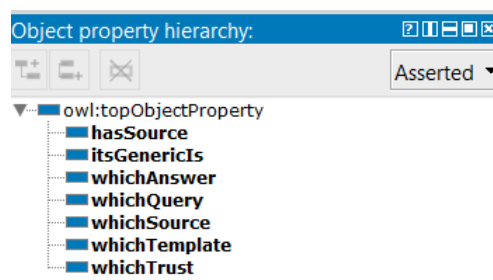


Figure 16: The final object property hierarchy of the ontology

These are designed to be self-describing. However, the property 'itsGenerics' should be explained. This property is used to link an individual of the Source class to a generic individual of the same subclass. This is able to provide common information on a source to the new individual, without requiring re-entry each time.

4.1.3 Populating the Ontology

With the classes and properties of the ontology defined, the final step was to populate the ontology with individuals. Individuals were inserted into the following classes:

- Trust – These individuals were the six entries to the table in Figure 1.
- Queries – These individuals were inserted into the subclass 'PoliticsCaseStudy', representing the four questions posed in the case study definition.
- Sources – These individuals represented sources answering the queries defined in the main case study. Some individuals represented different sources answering the same query, with up to three sources being used for each query. The sources used were all from the table in Figure 1.
- Answers – These individuals represented the answers to each query. They also brought together the query and the source, with a level of trust applied to each answer individual.

4.2 System

4.2.1 Installing Python Libraries

The first step to implementing the system was to install all the Python libraries required. The following libraries were installed:

- RdfLib [27]
- Pprint – A built in library for printing data in a readable manner. [28]
- Urllib.parse – A built in library used for handling URLs. [29]
- Nltk.tokenize – Part of the third party Natural Language Toolkit, for transforming text into tokens. [24]
- Nltk.stem – Part of the third party Natural Language Toolkit, for applying stemming algorithms to tokens. [25]
- Stop_words – A third party library that contains a list of common English stop words. [30]
- Gensim [20]
- Newspaper – A third party library that contains functions for scraping text from HTML webpages. [31]
- NumPy – A large third party library used for scientific computing; This was required to install the gensim library. [32]

To install these libraries, I used the Python install tool 'Pip' where possible. This enables installation from the command line with the following syntax:

pip install <package name>

Where available, pip downloads a '.whl' file from the library repository, and then unpacks and installs that file on the target PC. However, this caused a problem for me when installing NumPy, as Pip was trying to whl file that was not packaged for windows distributions. This was because NumPy is by default packaged for Linux distributions. When attempting to remedy this, I found a list of pre-packaged files prepared by Christoph Gohlke. [33] These were specifically packaged for correct versions of windows and by downloading the correct file – 'numpy-1.12.1+mkl-cp36-cp36m-win32.whl', I was able to install the library with the following command (Executed from within the folder where the file was located):

```
Pip install 'numpy-1.12.1+mkl-cp36-cp36m-win32.whl'
```

4.2.2 Implementing the System

Whilst implementing the project, I made several decisions regarding code in different areas of the created scripts. In addition, I ran into a number of challenges, again in several different areas. The final implementation produced three files, each containing functions for specific tasks. This section will discuss these decisions, and how I made any necessary decisions, as well as highlighting important areas of the code.

4.2.3 Manipulating the Ontology

4.2.3.1 Loading the Ontology into Python

The first step in the development of the system was to be able to load the ontology into python, so that it could be queried and modified. A new RDF object is created using the 'Graph()' function. For this implementation, I first created an empty graph, and then used the 'Parse()' function to load the ontology into the graph. Initially, I was attempting to load the ontology with the parameter 'format="rdf"', as below:

```
g = rdflib.Graph()
g.parse("OntologyOfTrustRDF.owl", format="rdf")
```

I was first under the impression that using this format would recognise that the syntax in the file 'OntologyOfTrustRDF.owl' was not in the OWL/XML format, but in RDF format as a series of triple declarations. However, when running the above code, I found this was not the case. I fixed this issue by testing the parameters 'format="owl"' and 'format="xml"'. When xml was used, the file was correctly loaded – the final correct line can be seen below:

```
g.parse("OntologyOfTrustRDF.owl", format="xml")
```

4.2.3.2 Querying the Ontology

Once the system could load the ontology, the next step was to implement the querying functionality. As stated in the design, to do this I would use the 'Query()' functionality from RdfLib. This function compiles and executes a statement in SPARQL, described in the background section to this project.

The function I created was to get a list of all the URLs stored in the ontology. The first part of this was to write a SPARQL query, that would return the URLs for every individual in the ontology that has the property 'url'. The query I used can be seen below:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX a: <http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#>

SELECT ?url
WHERE {
  ?entity a:url ?url
}
```

Figure 17: SPARQL query to extract URLs from ontology

It is important to note that a SPARQL statement needs to reference the URI of the ontology which it is querying as a prefix. This allows the query to locate the resources it requires to execute the query. In this case, the prefix 'a' is this URI. Without this prefix, the URI would have to be repeated for every concrete resource in the query. In the case above, this would only be for the 'url' property, such that the Where clause would instead read as follows:

```
?entity http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#url ?url
```

In the query, '?entity' and '?url' are variables that change with each line returned by the query (with only url being returned as a result), and so do not need to be referred to through a URI Prefix. I was able to test SPARQL queries in Protégé, using the SPARQL plugin, and this allowed me to ensure the queries worked before setting them up to be executed by RdfLib in Python.

Upon getting the result from the queries, I found that they were returned by rdflib as a custom object. I was able to access this object with the following code:

```
result = g.query(URLQUERY)
outputurls = []
for row in result:
    url = row['url']
    outputurls.append(url)
return outputurls
```

This code first executes the query (Figure 17) and then passes the result to the 'result' variable. As each row in the result can contain multiple 'columns', each corresponding to a value named in the Select clause of the query. In this case, the Select clause reads 'SELECT ?url', where 'url' is the returned value. Therefore, the syntax 'row['url']' or 'row.url' can be used to extract the value from the row. Unfortunately, this caused a problem for me, as the value was returned as a Literal – a custom type in RdfLib. This meant that the value was fine

when printed to the terminal, but could not be manipulated when passed to another function. For example, when trying to pass the output URLs as a list (see 'outputurls' in the above code), each URL was returned in the format 'Literal(url, datatype=XSD.AnyURI)'. After researching the RDFLIB documentation [27], I found there was a function 'toPython()' that would convert Literal objects to a value with the type of their closest Python equivalent (according to the datatype specified by the Literal).

Once I had fixed this, the only part of the code to change from above was line four:

```
url = row['url'].toPython()
```

This code was used in the function 'getAllUrlsFromOntology()' and the function 'getSomeLinks()' which is an extra function used to get links from just a single source.

4.2.3.3 Adding New Links to the Ontology

After creating functions for querying the ontology, the next step of the implementation was to create the required functions to add new links to the ontology. Whilst adding a new individual is very simple in Protégé, it is far more complicated to do programmatically. To do this, I had a choice of using two different RdfLib methods for adding triples to an RDF Graph. The first of these is the 'add()' function, which takes in a Subject, Predicate and Object as a triple. As discussed in the background, these are the three components of a triple in RDF. The other option was to use the 'INSERT' clause of SPARQL. This uses the same query functionality used to get links out of the ontology, but with an 'INSERT' clause followed by a set of triples, rather than a 'SELECT ? WHERE { }' clause. I chose to use the more readable 'add()' method, and this also allowed me to more easily pass in variables to the triples.

Firstly, I assessed which individuals in my ontology I would need to replicate to insert a new link into the ontology. I opened my ontology with turtle syntax, a human readable method of displaying an ontology, and copied out the two individuals I identified:

- AnswerBrexitImpactWalesBBC – An individual of type answer, which links together a source individual to a query and level of trust.
- BBC:BrexitWalesImpact – An individual of type Source (subclass: BBC), which represents the link from the Answer.

The code below show these as methods of

```
### http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#AnswerBrexitImpactWalesBBC
:AnswerBrexitImpactWalesBBC rdf:type owl:NamedIndividual ,
                                :AnswersToQueriesImplemented ;
                                :whichQuery :BrexitImpactEUWalesFunding ;
                                :whichSource :BBC:BrexitWalesImpact ;
                                :whichTemplate :PoliticsCaseStudyAnswersTemplateBBC .
```

Figure 18: The AnswerBrexitImpactWalesBBC individual in Turtle format

```
### http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#BBC:BrexitWalesImpact
:BBC:BrexitWalesImpact rdf:type owl:NamedIndividual ,
                        :BBC ;
                        :itsGenericIs :GenericBBC ;
                        :url "http://www.bbc.co.uk/news/uk-wales-36619404"^^xsd:anyURI .
```

Figure 19: The BBC:BrexitWalesImpact individual in Turtle format

By looking at these two definitions, I could see that I needed to implement a function that added two individuals that mirrored these. The created individuals would be similar, but take in variables from another function – a URL to be added to the ontology, and a string that would act as ‘context’. The context in this ontology is a unique name for an individual that concisely describes the query it answers. For example, in the above individuals, the context would be ‘BrexitWalesImpact’, showing that they are related to the impact Brexit will have on Wales.

In order to use the add function in my system, I first had to define all of the variables I wanted to use as references attached to a Uniform Resource Identifier (URI). This is a string that looks identical to URL, that defines every single item in the ontology. To complete this for the numerous references I would need for the triples, I used a variable named ‘onamespace’ to define a prefix for all of the variables.

```
onamespace = 'http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#'
```

The hashtag (#) represents the start of the value that is being defined. The most difficult of these variables to define was the context of the link. This was because this needed to be user defined, so that it could accurately reflect the meaning of the link. I did this by using Python’s user input function, which pauses the program when called, to wait for input. This was done with the following line of code:

```
context=input('What is the context for this link? (example: BrexitImpactWales):')
```

The string parameter is a prompt that is shown to the user whilst waiting for input. This context variable was then passed to my URI definition, as shown here:

```
ContextQuery = rdflib.URIRef(onamespace + 'Query:' + context)
```

‘ContextQuery’ is a variable to represent the query related to the answer in which the link is referenced. As previously mentioned, the rdflib.URIRef function takes a string and creates a URI object, which can be passed to the add function. In the above example, the string is constructed using the namespace, a prefix, and the user entered context variable.

Any Literals required for the new triples, as objects, also needed to be defined. Similarly to the above code, this was done using the rdflib.Literal function. To create a new Literal object was done with the following code:

```
autoaddtrue = rdflib.Literal('true', datatype=XSD.boolean)
```

This code defines a Boolean Literal, by stating the value as 'true' and then defining the Boolean datatype as part of the XSD namespace (XML Schema Definition) [34].

With all of the required subjects, predicates and objects defined, I was able to move on to creating each triple required to build the two individuals required. Using the add function of RdfLib, this required multiple triples for each individual. The subject was the same for each triple – a reference to the individual being created. The predicate and object was different for each triple, as these showed a different relationship of the individual.

```
g.add((AnswerContextSource, RDF.type, OWL.NamedIndividual))
```

This code is defining a triple that states that AnswerContextSource is of type individual. To complete this individual required five more triples, each added in the same manner. As an example, below is the code used to write the first individual, representing an individual of the 'AnswersToQueriesImplemented' subclass:

```
AnswerContextSource = rdflib.URIRef(ontnamespace + 'Answer:' + context +
sourceonly) #sourceonly represents the source of the article
AnswerType = rdflib.URIRef(ontnamespace + 'AnswersToQueriesImplemented')
whichQuery = rdflib.URIRef(ontnamespace + 'whichQuery')
whichSource = rdflib.URIRef(ontnamespace + 'whichSource')
whichTemplate = rdflib.URIRef(ontnamespace + 'whichTemplate')
ContextQuery = rdflib.URIRef(ontnamespace + 'Query:' + context)
SourceContext = rdflib.URIRef(ontnamespace + context + ':' + sourceonly)
CSTemplate = rdflib.URIRef(ontnamespace + 'PoliticsCaseStudyAnswersTemplate'
+
sourceonly)
autoAdded = rdflib.URIRef(ontnamespace + 'autoAdded')

g.add((AnswerContextSource, RDF.type, OWL.NamedIndividual))
g.add((AnswerContextSource, RDF.type, AnswerType))
g.add((AnswerContextSource, whichQuery, ContextQuery))
g.add((AnswerContextSource, whichSource, SourceContext))
g.add((AnswerContextSource, whichTemplate, CSTemplate))
g.add((AnswerContextSource, autoAdded, autoaddtrue))
```

The above code includes the variable definitions used in the inserted triples. Each variable represents a different resource in the ontology.

The final step was to save the updated ontology as a file. This was done using the ‘serialize’ function in RdfLib. This was the simplest part of implementing code for the ontology, as it was a simple one line statement that sent output to a specified file and format:

```
g.serialize(filename, format="xml")
```

4.2.4 Topic Modelling using Gensim

Upon the completion of the ontology manipulation functions, the next step was to create a set of functions that used topic modelling to compare an article from a new URL against the other articles in the ontology. The approach followed here was to implement functions for scraping articles, and manipulating text first, and then to implement the functions that use topic modelling.

4.2.4.1 Scraping Text from Articles

The initial step was to implement a function to download an article and then return the body (the text) of that article. This proved to be a challenge at first, as it is relatively easy to download the HTML contents of a webpage, but significantly more difficult to parse that HTML to get the body of an article. After some searching, I found a library called Newspaper, that provides simple yet powerful extraction and curation of news articles in multiple languages [31]. The library works by downloading and parsing a html file from a given URL into a custom ‘Article’ object, and then allows for a number of operations to be executed. The most important of these operations required for this project is ‘article.text’ which returns the body of the parsed article as an attribute of an article object. There are also a number of other operations that could be useful in future work for this project, such as ‘.keywords’ which returns key words of the text, and ‘.summary’ which returns a summary of the article text. The final code for this function works as follows:

```
def getText(url):  
    article = Article(url, language='en')  
    article.download()  
    article.parse()  
    return article.text
```

4.2.4.2 Manipulating the Text

Once downloaded, the article text needed to be turned into tokens – where each individual word in the string is turned into a string on its own. This results in an array of strings, with the same length as the number of words in the article. Once tokenized, stop words need to be removed from the string. As described in the background, stop words are a set of commonly used words that will add no value to our model. To remove these, I used the python library Stop-words, which provides a list of stop words that can be imported into the code. This was later replaced by a new, longer list written by Ranks NL [12]. This list contains over 650 stop words – a far more comprehensive amount.

Another problem found when removing stop words was that some words contained in the stop words were used in the articles used in the case study as important parts of the text. For example, stop word lists removed the word 'may'. This was an issue as Theresa May is the Prime Minister of the UK, and a prominent name in many of the articles used as part of the main case study. This was easy to fix by manually removing the word 'may' from the list in the stop words file.

The following code was used to remove the stop words from the list of tokens created from the article:

```
def removeStopWords(intokens):
    stopwords = stopwords_test.getStopWords()
    stoppedtokens = [i for i in intokens if not i in stopwords]
    return stoppedtokens
```

The most important line here is the third line – this is an efficient piece of code that creates a new list of tokens, minus the stop words, all in one line. This is a good demonstration for why Python is an excellent language to use for prototyping in projects such as these. As well as being efficient, it is easy to understand and quick to write.

After Tokenising the article, and removing stop words, the next step was to apply a stemming algorithm to the text. A stemming algorithm removes common suffixes from the ends of words, so as to normalise words. This will allow for more accurate modelling of the words found in each article. As previously mentioned, to do this a Porter Stemmer algorithm will be applied to the text [26]. A good implementation of the Porter Stemmer algorithm is found in the NLTK (Natural Language Toolkit) library for Python. This creates a 'PorterStemmer' object with the 'PorterStemmer()' function, before the '.stem(word)' function is applied, to stem the word. The code for this was executed as follows:

```
def stemTokens(intokens):
    '''stem tokens by using porter stemmer algorithm'''
    stemmer = PorterStemmer()
    stemmedtokens = [stemmer.stem(i) for i in intokens]
    return stemmedtokens
```

Here, the stemmed words are added to a list in the third line. Again, this is an efficient and easy to read method of implementing these functions.

4.2.4.3 *Creating a Model*

The next step was to create an LDA model of the topics from all the articles in the ontology. To do this, the python library Gensim was used. The function here uses the Gensim function 'corpora.Dictionary()' to create a dictionary of words, where the words are the previously tokenised and stemmed words from a group of articles. A corpus was then created, a list where the tokens in the dictionary are converted to the Gensim 'Bag of Words' format:

```
corpus = [dictionary.doc2bow(article) for article in tokenisedtexts]
```

Here ‘article’ is the tokens relating to a single article, and ‘tokenisedtexts’ is a list containing lists of tokens for every article in the ontology. This corpus is now in the correct format to be passed into the Gensim modelling functions – where the LDA model to be used in the rest of the system will be created.

```
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=numoftopics,
id2word=dictionary, passes=15)
```

This generates a new LDA model and passes it into the variable ‘ldamodel’. The ‘num_topics’ parameter is the number of topics which the model needs to generate – in every case this is the number of articles that are being passed into the model. The ‘id2word’ parameter takes in the ‘dictionary’ article and creates a set of vector ID mappings that relate to each word in the dictionary. Finally, the ‘passes’ parameter is the number of times the algorithm should run over the corpus to train the model.

4.2.4.4 LDA Compare

This was the hardest part of the implementation. This function was to take in the tokens generated from the articles in the ontology, and generate a LDA model containing these articles and a new article to be compared against the others. To do this, I planned to use Gensim’s vector ID functionality, where a vector can be generated that represents the position of a document in the model. However, this returned an error. Despite looking all over the Gensim website and tutorials, and the wider internet, I was unable to find a fix. Unfortunately, despite a long period troubleshooting, I was forced to leave this function unfixed. See Testing section in Results and Evaluation; Test 8 for more detail about the error. The below code is the current state of this function – trying to print the vector location of a query in the LDA model space:

```
def LDACompare(urllist, currentTokens, newurl):
    dictionary = corpora.Dictionary(currentTokens)
    corpus = [dictionary.doc2bow(article) for article in currentTokens]

    model = gensim.models.ldamodel.LdaModel(corpus, id2word=dictionary, passes=15)
    query = "This is a test document to be used as a query"
    vec_query = dictionary.doc2bow(query.lower().split())
    vec_lda = model[query] #convert query to the LDA space
    print(vec_lda)
```


4.2.4.5 Simple Article Comparison

The final part of the implementation was to implement the simple comparison of articles. This was done following the pseudocode in Figure 13, however some changes were made during the implementation:

```
def simpleArticleComparison(urllist, currentTokens, newurl):
    '''Return the url of the most similar article to the new url'''
    newtokens = list(set(singleInputTokenize(getText(newurl))))
    #set() creates an ordered list, removing duplicates, list() returns the set to list
    type
    print(str(len(newtokens)))
    # Calculates the % of words that match in the tokens
    count = 0
    nummatcheslist = []
    for tokenlist in currentTokens:
        nummatches = 0
        uniquetokenlist = list(set(tokenlist)) #remove duplicates
        for token in newtokens:
            if token in uniquetokenlist:
                #increment if the same token is found in each list
                nummatches += 1
        print(str(count) + ': ' + str(nummatches))
        nummatcheslist.append(nummatches)
        count += 1
    print('Matches List: ' + str(nummatcheslist))
    similarities = []
    for i in nummatcheslist:
        #create a list of each article compared and the similarity to new doc
        similarity = (i / len(newtokens))
        similarities.append(similarity)
        print(str(similarity))
    print('Similarity' + str(similarities))
```

This code takes in a list of URLs, the set of articles to be compared against (as tokens), and the URL of the article to compare against the current articles. It removes duplicate words from each set of article tokens using the set() function, and then compares the tokens in the articles to find the number of matching tokens. It then calculates this as a percentage, and prints out the values to be appraised by the user.

5 Results and Evaluation

5.1 Testing the Ontology

To test the ontology, we will evaluate the competency questions against our ontology, to ensure that each one has been met as described.

1. There are classes that represent the following:
 - a. Queries
 - b. Sources
 - c. Trust
 - d. Answers to queries

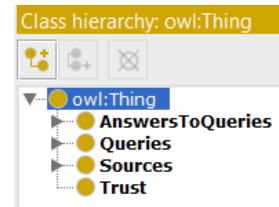


Figure 20: The correct top level classes are present

This is complete, these classes are present:

2. The class Queries contains subclasses for each case study we wish to represent:
 - a. A Politics case study for queries about politics.
 - b. A Sports case study for queries about sports.



Figure 21: The correct subclasses of query are present

This is complete, these subclasses are present:

3. Each subclass of Queries will contain the following individuals representing a query: (For now we are populating only the politics subclass)
 - a. What was the final result of last year's Brexit Referendum?
 - b. What is the impact of Brexit on EU funding in Wales?
 - c. How did Wales vote in the Brexit Referendum?
 - d. Who replaced David Cameron as the Prime Minister of the United Kingdom?

This is complete, the correct individuals are present:

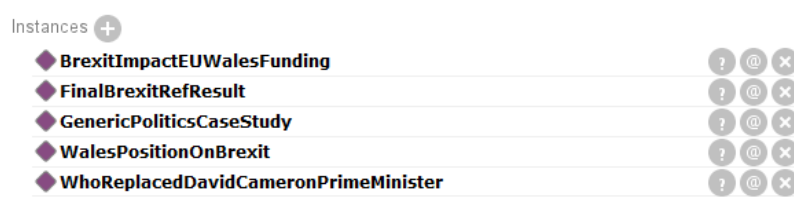


Figure 22: The correct individuals for queries are present

4. The class Sources contains a subclass for every type of source we wish to represent:
 - a. Social Media
 - b. Websites



Figure 23: The correct subclasses of Sources are present

This is complete, these subclasses are present:

5. The subclass SocialMedia contains subclasses to represent social media sources
 - a. Facebook
 - b. Twitter
 - c. Reddit

This is complete, these subclasses are present:

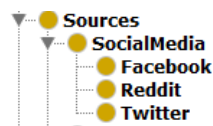


Figure 24: The correct subclasses of social media are present

6. The subclass Websites contains subclasses for websites we wish to represent:
 - a. There is a Broadcasters subclass for websites of television broadcasters
 - b. There is Print Media subclass for websites of newspapers and magazines
7. The Broadcasters subclass contains subclasses to represent the following broadcasters:
 - a. BBC
 - b. ITV
 - c. Channel 4
8. The Print Media subclass contains subclasses to represent the following newspapers/magazines:
 - a. The Guardian
 - b. The Telegraph
 - c. The Times
 - d. The Economist
 - e. The Financial Times

The requirements 6, 7, and 8 are complete, these subclasses are all present:

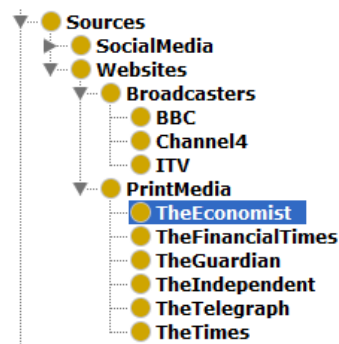


Figure 25: The correct subclasses are present in Websites

9. Every individual stored within the Source class or one of its subclasses represents one of the following:
 - a. Article
 - b. Post
 - c. Page
 - d. (Social Media) Account

This is complete, all individuals in this class represent the above types of source. The exception is social media accounts/posts, which have not been populated in this ontology below the class level.

10. The Trust class contains only individuals to represent the level of trust in a source.

This consists of six individuals (see Figure 1 or [1, p. 30]):

- a. Completely Reliable
- b. Usually Reliable
- c. Fairly Reliable
- d. Not Usually Reliable
- e. Unreliable
- f. Cannot Be Judged

This is complete, the above individuals are present:

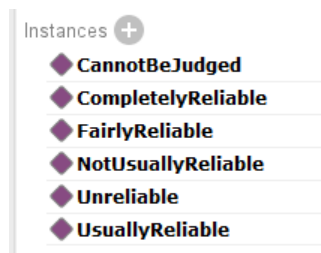


Figure 26: The correct individuals are present in the Trust class

11. The individuals in the Answers class link individuals from the Sources, Queries and Trust classes to create answers to a query.

- a. Each answer is linked to a single source, so each query can have multiple answers from different sources.

This is complete, below is an example in the form of 'AnswerBrexitImpactWalesBBC', which answers the query 'What is the impact of Brexit on Wales?' and links the two together with a source:

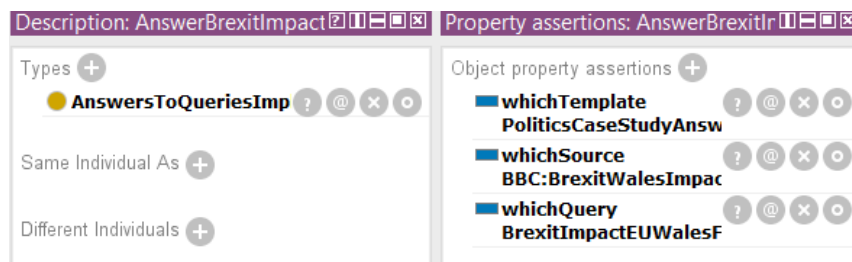


Figure 27: An example of an answer individual linking to queries and sources

12. The Answers class contains the following subclasses:

- a. A subclass containing all complete answers.
- b. A subclass containing templates to represent generic answers.

This is complete, the subclasses are present:

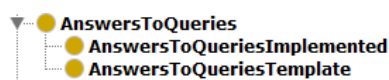


Figure 28: The correct subclasses are present in the Answers class

5.2 Testing the System

We will need to ensure that the system works as expected and that there are no problems. Any problems found will be noted and, where possible, a fix described alongside. This testing will be in the form of unit-testing of the functions used in the final build of the project. For each test, the function will be set up to run with its input in the main method of the Python file and then executed using the Windows terminal.

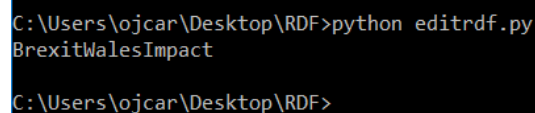
5.2.1 Test 1: getContext(inputurl)

This function takes in an input URL, and returns the context property related to the source individual representing that URL in the ontology.

Input: <http://www.bbc.co.uk/news/uk-wales-36619404> (Present in ontology)

Expected Output: 'BrexitWalesImpact'

Output:



```
C:\Users\ojcar\Desktop\RDF>python editrdf.py
BrexitWalesImpact

C:\Users\ojcar\Desktop\RDF>
```

Figure 29: Result of Test 1 – getContext

This test was a success.

5.2.2 Test 2: getAllUrlsFromOntology()

This function extracts every link represented by an individual in the ontology (through the 'url' property). The function takes no input.

Expected Output: A list of 13 URLs.

Output:



```
C:\Users\ojcar\Desktop\RDF>python editrdf.py
Length of List: 13
['https://www.theguardian.com/small-business-network/2016/nov/11/what-brexit-mean-business-funding-wales', 'http://www.bbc.co.uk/news/politics/eu_referendum/results', 'https://www.theguardian.com/politics/2016/jul/11/cameron-announces-he-will-step-down-after-pmq-s-on-wednesday', 'http://www.telegraph.co.uk/news/0/theresa-may-who-is-the-woman-bidding-to-be-the-next-tory-leader/', 'http://www.telegraph.co.uk/news/2016/06/24/what-can-we-learn-from-the-eu-referendum-results/', 'http://www.bbc.co.uk/news/uk-politics-eu-referendum-36612308', 'http://www.independent.co.uk/news/uk/politics/brexit-wales-vote-leave-eu-referendum-result-brussels-funding-economy-a7136196.html', 'http://www.bbc.co.uk/news/uk-politics-36788782', 'http://www.bbc.co.uk/news/uk-wales-36619404', 'http://www.bbc.co.uk/news/uk-politics-38996179', 'http://www.telegraph.co.uk/news/2016/06/24/how-britain-backed-a-brexit-the-key-moments-of-the-night/', 'https://www.theguardian.com/commentisfree/2016/dec/08/wales-brexit-welsh-national-assembly-uk-constitution', 'https://www.theguardian.com/politics/2016/jun/24/britain-votes-for-brexit-eu-referendum-david-cameron']

C:\Users\ojcar\Desktop\RDF>
```

Figure 30: Result of Test 2 - getAllUrlsFromOntology

This test was a success.

5.2.3 Test 3: addToOntology(newurl, filename, oldurl="")

This function takes in a new URL to add to the ontology, a filename for the new ontology file, and an optional input for the oldurl.

Input: newurl = <http://www.bbc.co.uk/news/uk-wales-politics-36867963> ; filename = test.rdf ; Context will be 'TestContext'.

Expected Output: two new individuals named 'TestContext'.

Output:

```
<rdf:Description rdf:about="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#TestContext:BBC">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <untitled-ontology-11:itsGenericIs rdf:resource="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#GenericBBC"/>
  <untitled-ontology-11:url rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.bbc.co.uk/news/uk-wales-politics-36867963
    </untitled-ontology-11:url>
  <untitled-ontology-11:autoAdded rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</untitled-ontology-11:autoAdded>
  <rdf:type rdf:resource="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#BBC"/>
</rdf:Description>

<rdf:Description rdf:about="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#Answer:TestContextBBC">
  <untitled-ontology-11:whichSource rdf:resource="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#TestContext:BBC"/>
  <untitled-ontology-11:whichQuery rdf:resource="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#Query:TestContext"/>
  <rdf:type rdf:resource="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#AnswersToQueriesImplemented"/>
  <untitled-ontology-11:whichTemplate rdf:resource="http://cicero.cs.cf.ac.uk/onto/trustosint/ontology#PoliticsCaseStudyAnswersTemplateBBC"/>
  <untitled-ontology-11:autoAdded rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</untitled-ontology-11:autoAdded>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
</rdf:Description>
```

Figure 31: Result of Test 3 - addUrlToOntology

This test was a success, the output RDF shows two new individuals using the new information.

5.2.4 Test 4: getText(url)

This function takes in a URL and returns the text (contents) of the article located at the URL.

Input: <http://www.bbc.co.uk/news/uk-wales-36619404>

Expected Output: Full text of the above article.

Output:

```
C:\Users\ojcar\Desktop\RDF>python comparearticles.py
C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\utils.py:855: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
First 500 characters:
#####
Image copyright Christopher Furlong/Getty Images Image caption Areas in west Wales have qualified for the highest amount
of EU support since 2000

It is clearly about more than money.

Wales gets the largest amount of money from the European Union and it also returned a vote to leave.

Wales' trade with EU member countries and investment from them is very significant but so too are funds from the EU itse
lf.

Since 2000, Wales has been awarded the highest levels of economic support because we hav

C:\Users\ojcar\Desktop\RDF>
```

Figure 32: Result of Test 4 - getText

This test was a success, the first 500 characters of the downloaded article match those in the link (as of 05/05/2017).

5.2.5 Test 5: singleInputTokenize(text)

This function takes in an article text, and carries out the following operations: splits the text into tokens, removes stop words, and then stems the remaining tokens.

Input: Text from <http://www.bbc.co.uk/news/uk-wales-36619404>

Expected Output: A list of tokens that have been correctly tokenised.

Output:

```
C:\Users\ojcar\Desktop\RDF>python comparearticles.py
C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\utils.py:855: UserWarning: detected
Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
Output:
#####
['imag', 'copyright', 'christoph', 'furlong', 'getti', 'imag', 'imag', 'caption', 'area', 'west', 'wale', 'qualifi', '
highest', 'amount', 'eu', 'support', '2000', 'clearli', 'money', 'wale', 'largest', 'amount', 'money', 'european', 'un
ion', 'return', 'vote', 'leav', 'wale', 'trade', 'eu', 'member', 'countri', 'invest', 'fund', 'eu', '2000', 'wale', 'a
ward', 'highest', 'level', 'econom', 'support', 'consist', 'poorest', 'part', 'eu', 'despit', 'member', 'state', 'east
ern', 'europ', 'media', 'playback', 'unsupport', 'devic', 'media', 'caption', 'wale', 'poorest', 'econom', 'area', 'bl
aenau', 'gwent', '62', 'peopl', 'vote', 'brexit', 'wale', '4bn', 'call', 'structur', 'fund', 'eu', '2000', 'welsh', 'g
overn', 'set', 'continu', '2020', 'unclear', 'happen', 'money', 'alloc', 'year', 'tranch', 'may', 'continu', 'idea', '
money', 'help', 'welsh', 'econom', 'stronger', 'wealthier', 'imag', 'caption', 'wale', 'divid', 'area', 'qualifi', 'l
evel', 'eu', 'structur', 'fund', 'wale', 'divid', 'area', 'west', 'wale', 'valley', '15', 'local', 'author', 'east', '
wale', 'west', 'wale', 'valley', 'highest', 'level', 'call', 'structur', 'fund', '2000', 'call', 'object', 'call', 'co
nverg', 'fund', 'countri', 'eu', 'gva', 'three', 'quarter', 'eu', 'averag', 'gva', 'money', 'wale', 'alloc', 'current'
, 'round', '2020', 'involv', '1', '89bn', 'structur', 'fund', 'wale', 'pot', 'european', 'social', 'fund', 'esf', 'eur
opean', 'region', 'develop', 'fund', 'erdf', 'money', 'fund', 'initi', 'wale', 'instanc', 'job', 'growth', 'wale', 'up
grad', 'a465', 'head', 'valley', 'road', 'uk', 'vote', 'leav', 'eu', 'wale', 'clearli', 'longer', 'elig', 'fund', 'pro
', 'brexit', 'campaign', 'adam', 'regener', 'fund', 'flow', 'wale', 'uk', 'treasuri', 'uk', 'leav', 'imag', 'caption',
'eu', 'money', 'help', 'fund', 'head', 'valley', 'road', 'improv', 'wale', 'stand', 'higher', 'proport', 'invest', 'f
di', 'averag', 'uk', '1', '100', 'foreign', 'own', 'firm', 'work', 'wale', 'employ', '150', '000', 'peopl', 'european'
, 'singl', 'market', 'meant', 'trade', 'massiv', 'market', 'place', 'trade', 'barrier', 'chang', 'impact', 'exist', 'f
oreign', 'compani', 'wale', 'may', 'hope', 'attract', 'morn', 'chief', 'execut', 'aston', 'martin', 'invest', 'hundr',
'million', 'wale', 'creat', '750', 'job', 'st', 'athan', 'britain', 'seek', 'tariff', 'free', 'european', 'market', '
tata', 'cours', 'own', 'port', 'talbot', 'llanwern', 'shotton', 'trostr', 'orb', 'steel', 'work', 'well', 'jaguar', 'l
and', 'rover', 'welsh', 'firm', 'suppli', 'access', 'market', 'skill', 'workforc', 'remain', 'consider', 'busi', 'brit
ain']

C:\Users\ojcar\Desktop\RDF>
```

Figure 33: Result of Test 5 - singleInputTokenize

This test was a success. The function returns a list of stemmed tokens, with stop words removed.

5.2.6 Test 6: createModel(tokenisedtexts, numoftopics)

This function creates a LDA Model. It takes in the tokenized articles, and the number of articles being entered as input.

Input: tokenisedtexts = ['This is a small input for testing one', 'This is a small input for testing two', 'This is a small input for testing three'], numoftopics = 3

Expected Output: A printed model displaying topics from the three texts entered.

Output:

```
C:\Users\ojcar\Desktop\PDF>python comparearticles.py
C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\g
Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
Output:
#####
[(0, '0.294*"test" + 0.294*"input" + 0.294*"small" + 0.117*"three"'),
 (1, '0.250*"small" + 0.250*"test" + 0.250*"input" + 0.249*"three"'),
 (2, '0.250*"small" + 0.250*"input" + 0.250*"test" + 0.249*"three"')]
C:\Users\ojcar\Desktop\PDF>
```

Figure 34: Result of Test 6 – createModel

This test was a success. The model was printed containing the input words under topics.

5.2.7 Test 7: userCompare(urltocompare)

This function compares a url against the whole ontology, and then gets inserted into the ontology if a similar match is identified by the user. The models are printed out to the terminal for inspection by the user prior to adding the link to the ontology. This test will test up to the end of the comparison; addToOntology has already been proven to work.

Input: <http://www.bbc.co.uk/news/uk-wales-politics-36867963>

Expected Output: Program will print out a model of the links in the ontology, and a model containing the new link. User analyst must compare the two models to see if a match is found, then decide if the new link should be inserted in to the ontology.

Output:

```
'0.001*may" + 0.001*blair" + 0.001*referendum" + 0.001*cameron" + '
'0.001*caption" + 0.001*european" + 0.001*busi"),
(10,
'0.017*may" + 0.011*1" + 0.006*year" + 0.006*member" + 0.006*day" + '
'0.006*own" + 0.006*polit" + 0.006*100" + 0.006*british" + 0.006*born" + '
'+ 0.006*lost" + 0.006*kitten" + 0.006*inject" + 0.006*earli" + '
'0.006*politician" + 0.006*octob" + 0.006*rest" + 0.006*item" + '
'0.006*cookbook" + 0.006*die"),
(11,
'0.055*fund" + 0.033*busi" + 0.032*wale" + 0.029*eu" + 0.015*social" + '
'0.010*project" + 0.010*support" + 0.009*peopl" + 0.009*govern" + '
'0.007*enterpris" + 0.007*small" + 0.007*provid" + 0.007*wallich" + '
'0.007*welsh" + 0.006*money" + 0.006*homeless" + 0.006*servic" + '
'0.006*european" + 0.006*work" + 0.006*invest"),
(12,
'0.001*may" + 0.001*brexit" + 0.001*prime" + 0.001*eu" + 0.001*minist" + '
'+ 0.001*imag" + 0.001*campaign" + 0.001*peopl" + 0.001*secretari" + '
'0.001*cameron" + 0.001*blair" + 0.001*britain" + 0.001*govern" + '
'0.001*vote" + 0.001*media" + 0.001*leav" + 0.001*caption" + '
'0.001*referendum" + 0.001*street" + 0.001*uk')])
#####
Model of link to compare:
#####
[(0,
'0.020*welsh" + 0.018*wale" + 0.013*brexit" + 0.011*time" + 0.011*imag"
'+ 0.011*leav" + 0.009*govern" + 0.009*sens" + 0.009*fund" + '
'0.009*economi" + 0.007*negoti" + 0.007*vote" + 0.007*earli" + '
'0.007*fundament" + 0.007*remain" + 0.007*campaign" + 0.007*month" + '
'0.007*system" + 0.005*eu" + 0.005*spend')]
#####
Would you like to insert this link into the ontology? (y/n):
```

Figure 35: Result of Test 7 – userCompare

This test was a success, both required models were printed, and the user can compare the two to find a similarity. The user is correctly prompted to add the new link into the ontology if they find a similarity.

5.2.8 Test 8: LDACompare(urllist, currentTokens, newurl)

This function takes in the list of URLs from the ontology, those URLs as tokens, and the URL to compare against the current URLs. The function should query the model to return the article being compared, and the article being compared against, and then return a similarity value between the two articles.

Input: urllist = all URLs in the ontology, currentTokens = These URLs through the tokenize function, and newurl = <http://www.bbc.co.uk/news/uk-wales-politics-36867963>

Expected Output: The function will check to see if a similarity is found, and if there are two articles with a high similarity, will add the URL to the ontology.

Output:

```
C:\Users\ojcar\Desktop\PDF>python comparearticles.py
C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\utils.py:855: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
Traceback (most recent call last):
  File "comparearticles.py", line 237, in <module>
    LDACompare(listurls, tokens, 'http://www.bbc.co.uk/news/uk-politics-eu-referendu
m-36612308')
  File "comparearticles.py", line 166, in LDACompare
    vec_lda = model[query] #convert query to the lda space
  File "C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\models\ldamodel.py", line 974, in __getitem__
    return self.get_document_topics(bow, eps, self.minimum_phi_value, self.per_word_topics)
  File "C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\models\ldamodel.py", line 915, in get_document_topics
    gamma,phis = self.inference([bow], collect_sstats=per_word_topics)
  File "C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\models\ldamodel.py", line 430, in inference
    ids = [int(id) for id, _ in doc]
  File "C:\Users\ojcar\AppData\Local\Programs\Python\Python36-32\lib\site-packages\gensim\models\ldamodel.py", line 430, in <listcomp>
    ids = [int(id) for id, _ in doc]
ValueError: not enough values to unpack (expected 2, got 1)
```

Figure 36: Result of Test 8 – LDACompare

This test was a failure. At the query stage the function failed, returning the error shown in Figure 36. After a number of different attempts to troubleshoot and repair this error, no fixes were found. As this was a main function of the system, the focus will now turn to the 'userCompare()' function, as this can still compare articles. This will just require more human assistance instead.

5.2.9 Test 9: simpleArticleComparison(urllist, currentTokens, newurl)

This function takes in the list of URLs from the ontology, those URLs as tokens, and the URL to compare against the current URLs. The function executes a fast comparison of the words in the new URL against the words in each article of the current URLs. The function should return a list of similarity scores to the user, which can be analysed and if there is one (or more) that is similar enough the URL can be added to the ontology.

Input: urllist = all URLs in the ontology, currentTokens = These URLs through the tokenize function, and newurl = <http://www.bbc.co.uk/news/uk-wales-politics-36867963>

Expected Output: The function will return a list of percentage scores that can be used by the user to decide on similarity.

Output:

```
Matches List: [63, 19, 66, 91, 12, 79, 315, 98, 47, 86, 0, 18, 9]
0.2
0.06031746031746032
0.20952380952380953
0.28888888888888886
0.0380952380952381
0.2507936507936508
1.0
0.3111111111111111
0.1492063492063492
0.273015873015873
0.0
0.05714285714285714
0.02857142857142857
Similarity[0.2, 0.06031746031746032, 0.20952380952380953, 0.28888888888888886, 0.0380952380952381, 0.2507936507936508, 1.0, 0.3111111111111111, 0.1492063492063492, 0.273015873015873, 0.0, 0.05714285714285714, 0.02857142857142857]
C:\Users\ojcar\Desktop\RDF>
```

Figure 37: Result of Test 9 – simpleArticleComparison

This test was a success; the function returns a list of similarities for the user to analyse. As can be seen, there is an article with a 100% similarity, so there would be no need to insert this article into the ontology as it already exists.

5.3 Results

After testing, we have found that the system works in almost all areas. The main problem is in the LDACompare function. This function failed the unit testing, and will not be able to be used any further in the project. See the Future Works section for more on the use of this function in the future.

In this section of the report, we will gather more results for the userCompare function of the project in order to evaluate whether it is a viable use of the LDA model. If it is a viable model then we will be able to conclude that the prototype system developed in this project (and by extension the ontology) is a success. The evaluation will also assess how our notion of trust fits into the prototype.

5.3.1 The Experiment

As part of this project, the aim is to understand how we can assess trust in the web. By comparing articles, the project should show that if an article is similar to another, the trust should be maintained from one article to the other. This is because the viewpoint will stay the same.

The experiment to run, is being able to use LDA to create a new model containing the articles in the ontology, and manually comparing the output another model of an article to be compared. This will allow for the project to see if it is possible for a system to leave some control to the user. This is useful to examine, as a user may make different assumptions and conclusions to a programmatic solution, which may result in new insight to whether an article is similar.

5.3.2 Inputs

For producing these results, I will use a set of three links to compare against the database. These will consist of one 'new' link, which is similar to a link already in the ontology, one

‘new’ link relating to a different case study, and one link which is identical to one already in the ontology. Below are the links to be used as input:

- New, similar link: <http://www.bbc.co.uk/news/uk-politics-eu-referendum-36618219>
- Identical link: <https://www.theguardian.com/politics/2016/jun/24/britain-votes-for-brexit-eu-referendum-david-cameron>
- Dissimilar link: <http://www.bbc.co.uk/sport/rugby-union/38951926>

5.3.3 Output

5.3.3.1 New, Similar Link

The results when executing the human-assisted system with a similar link are as follows:

When this link was entered, it returned the following a full model of all of the links in the ontology, as well as a small model representing the new link. By examining the full output, in Appendix A: Output from new, similar link, we see that the eleventh link in the large model is fairly similar to the new link:

```
(11,
'0.001*"leav" + 0.001*"eu" + 0.001*"vote" + 0.001*"brexit" + '
'0.001*"campaign" + 0.001*"peopl" + 0.001*"wale" + 0.001*"labour" + '
'0.001*"fund" + 0.001*"blair" + 0.001*"minist" + 0.001*"parti" + '
'0.001*"leader" + 0.001*"remain" + 0.001*"referendum" + 0.001*"uk" + '
'0.001*"govern" + 0.001*"welsh" + 0.001*"area" + 0.001*"mp"' ),
```

Figure 38: Eleventh link in large model

```
[(0,
'0.028*"wale" + 0.019*"leav" + 0.017*"vote" + 0.016*"welsh" + 0.011*"labour" '
'+ 0.010*"campaign" + 0.009*"voter" + 0.009*"remain" + 0.008*"peopl" + '
'0.008*"leader" + 0.008*"eu" + 0.008*"polit" + 0.008*"bbc" + 0.007*"support" '
'+ 0.007*"jone" + 0.007*"minist" + 0.007*"referendum" + 0.007*"area" + '
'0.006*"surpris" + 0.006*"govern"')]
```

Figure 39: Model of similar link being compared

Between these two figures, it can be seen that there is a similarity between the outputs – both share the words ‘leav’ and ‘vote’ prominently, and there is a lot of cross over into the middle areas of both models. This shows that the link is similar. Now, as this is a human assisted process, I as the user need to give the system a ‘yes’ to say that this link can be added to the ontology.

Once selected, I used the context ‘WalesRejectsEU’. This successfully added the link to the ontology, and created the required two individuals in the ontology.

5.3.3.2 Identical Link

When executing the system using a link that is identical to one already in the ontology, the relevant output is below (Full output included as Appendix B: Output from new, identical link):

```
[(0,
  '0.020*"leav" + 0.019*"vote" + 0.017*"remain" + 0.015*"labour" + '
  '0.011*"peopl" + 0.010*"cameron" + 0.009*"parti" + 0.008*"leader" + '
  '0.008*"brexit" + 0.008*"uk" + 0.008*"victori" + 0.008*"campaign" + '
  '0.008*"referendum" + 0.007*"corbyn" + 0.007*"call" + 0.007*"minist" + '
  '0.007*"london" + 0.007*"eu" + 0.006*"nation" + 0.006*"prime"')]
```

Figure 40: Model of identical link being compared

When comparing the above model, containing the new link, against the model containing the current links, there is an interesting issue. This new link is identical to a link that is already in the ontology, and yet the topic models over these links share very few similarities. This excellently demonstrates a property of LDA Models – that the distribution of words in a topic is partly random. A way to fix this would be to run the model again, editing the code to train the model a far higher number of times. This would decrease the randomness in the word distribution, and enable an analyst to far more easily identify a similar link in the model.

For this link, there is no need to add it to the ontology – it is known that it is identical to an already present link, and the difficulty to assess its similarity also causes a problem.

5.3.3.3 Dissimilar Link

The final link to be run through the system is one which is covering a totally different subject to the subject of the case study. In this case, the new link will be from a sport website. The result from the system should show a topic model containing totally different words to the ones in the current model. Below we can see the output from the model for the new link (Full output in Appendix C: Output from dissimilar link):

```
[(0,
  '0.036*"wale" + 0.028*"game" + 0.019*"win" + 0.017*"england" + '
  '0.015*"warburton" + 0.015*"februari" + 0.013*"bbc" + 0.013*"25" + '
  '0.013*"saturday" + 0.013*"team" + 0.013*"scotland" + 0.013*"improv" + '
  '0.011*"jone" + 0.011*"march" + 0.011*"ve" + 0.011*"gmt" + 0.011*"nation" + '
  '0.009*"minut" + 0.009*"manag" + 0.009*"cardiff"')]
```

Figure 41: Model of dissimilar link being compared

In this model, we can see that there is no overlap at all between the above model for the new link, and the other model containing the current links. This is a good demonstration of

where human assisted analysis in the system is effective – it can be seen that this model is completely different at a glance.

Again, for this link there is no reason to add it to the ontology, as it is completely unrelated to the case study.

5.4 Evaluation

In summary, most the system works as predicted, as the system passed all but one of the tests set. The failed test was the part of the system based on using LDA to compare articles programmatically, rather than through human assisted comparison, and this is a blow to the system. However, the human assisted comparison of LDA models proved fairly effective, and these results show that the system can potentially be increased in scope in the future to include more complex case studies.

Looking at the results of the human assisted comparison, we saw that a user would be able to adequately compare articles between the models fairly accurately and efficiently. A problem with this is that if the ontology becomes more complex, and more populated, the user's job would become much harder. This is because with a more populated ontology comes a far larger model, and it is at this point that a programmatic solution would become even more useful.

The other half of the system, the half that deals with manipulating the ontology, turned out very well in the project. The query functions worked well, and the function to add new individuals to the ontology worked well. A problem was that the add function was slightly constrained – the individuals had to fit a strict structure of the RDF triples being inserted into the ontology. This is something that can be looked at in the future work section below.

The ontology shows that the system accurately uses the trust in a source. Trust is automatically passed on to links from the same source, and this is perfectly fine for links from reputable and consistent news sources. However, as some sources can potentially offer differing levels of trust (perhaps based on the author of a particular article), it could be interesting to see how a system would automatically recalculate the level of trust based on other factors.

6 Future Work

6.1 Further Troubleshooting of LDA querying

As mentioned above, I had a lot of trouble with querying the LDA model to return a result for a single article. In the future, I would like to spend more time looking at this in closer detail, to try and troubleshoot more successfully where the issue lies. This would enable us to use the full power of an LDA model, as instead of relying on a human analyst to compare the models, we could get the system to do it programmatically.

6.2 Dynamic Creation of RDF Triples

When inserting new individuals into the ontology, the current system uses a very strict triple structure. A possible expansion of the project in the future could look at how to implement the dynamic creation of triples to the ontology, such that the structure of an individual is not limited to a set structure in the code.

6.3 Implement the Second Case Study

This project focused on implementing the first case study defined in Selection of Approach. However, in the future, the second case study could be implemented as a means of testing the ontology structure in a different structure. This could also be appropriate for potential future applications of this ontology, where it could be used in more complex, real world problems such as analysing the effects of global warming or the efficacy of vaccinations around the world.

6.4 Introduce Social Media to the Ontology

In its current form, the final ontology has a class to handle social media links, but does not make use of any. In the future, I would look at ways to introduce social media to the ontology, as social media posts can provide an excellent method of seeing multiple perspectives on a single issue. This is something that would be incredibly useful to an intelligence analyst, and hence is something that would be appropriate for inclusion through a future project. In addition, this would be a fantastic method to evaluate trust in social media, as the posters of social media posts could be official news or simply members of the public sharing an opinion. It would be interesting to see how to assign trust to different social media accounts or posts, and how to represent this in our ontology.

This could be expanded further by using social media APIs, such as the Twitter REST or Streaming APIs [35]. These would allow us to pull real time data from twitter, maybe related to a hashtag for a current event. This data could then be analysed to see if there is a consensus among the posts or if there are two or more sides with differing viewpoints on a single event.

6.5 Use Web Scraping to Populate the Ontology

Another piece of potential future work would be to use a web scraping or crawling platform to automatically populate the ontology with relevant articles. This could possibly be done in python using a library such as Scrapy [36]. Scrapy could be used in conjunction with the link comparison system to find new links, appropriate to a query and then compare them to existing links in the ontology. These could then be added to the ontology either if there

were no links to answer a query or if a link is found to be more suitable than the original link.

6.6 Integrate with NewsAPI

NewsAPI is an API that returns JSON data for headlines displayed on news sources [37]. The data returned also contains the URL to each article, so this could be used to populate the ontology with up to date news information in a case study related to the subject of current affairs. An interesting experiment would be to implement this API alongside the comparison system and see whether it is possible to differentiate between trustworthy and untrustworthy news sources, to gather a broad view of opinion on the subject of the case study.

7 Conclusion

Upon completing the initial plan for this project, the following overall aims and objectives were set:

- Carry out a literature review, and identify some appropriate case studies.
- Define a small set of queries to be used alongside the case studies.
- Define a set of sources upon which queries can be run.
- Develop an ontology of the degree of trust in information and sources of information.
- Investigate the use of machine learning to work alongside the ontology.

The first four aims have been met. A background review was carried out to gain an insight into the information available on the internet, and how this information can be used to assess trust. In particular, open source intelligence was of interest, as this is information available at all times without restrictive use. A pair of case studies were also defined, along with a set of sources to apply to them. The first case study was used in this project, and it provided a broad base of information on which to create the first version of the ontology. The ontology was also developed to a good standard, and this does an excellent job of showing the relationships between source, queries, and trust.

Finally, the use of machine learning was investigated through using topic modelling. This was successful, in that the system could create new models, and output them to the user. The user could manually compare the models and then ask the system to add the new link to the ontology if a similarity was found. However, it was found that this would be time consuming and very inefficient on a large scale. Therefore, it would be better to programmatically compare the models, or at least the articles inside the models to find similarities and decide whether to insert a new link into the ontology. Unfortunately, this functionality was not able to be introduced in a working manner to the system.

To summarise, the project has resulted in the creation of an ontology of queries, sources, and trust, and the creation of a system to compare articles with some help from a human user. The system can successfully help the user find a similar article over a small query set of documents, and to see how trust in a source relates to queries and answers, but will struggle when scaled up to a large data set in its current form. The project as a whole has set a broad basis for further work to be done in this area. The further work section describes several potential directions in which the project could be taken, alongside fixing issues remaining from this project.

8 Reflection

At the start of this project, my knowledge of ontologies was very minimal. I had briefly covered aspects of ontologies in some of my lectures prior to this semester (Knowledge Management; Large Scale Databases), but they were not an area in which I had a huge amount of knowledge. As mentioned multiple times throughout this report, I used a University of Manchester tutorial on OWL and Protégé to learn how to implement my own ontologies, and this was helpful. This was relatively easy however, especially in the environment created by Protégé. What was harder was to learn RDF and SPARQL to a high standard. To learn these, I essentially dived straight in using RdfLib in Python, and learnt how to manipulate existing RDF with the library functions. This was followed by creating my own RDF graph and manipulating that in Python to gain a deeper understanding.

The other aspect of this project was using machine learning to carry out topic modelling. This was an area in which I had no experience at all, and whilst it was a very interesting area, I did struggle to understand it completely. This is especially clear in my code, where I struggled to get a couple of the LDA functions to work correctly. In the end, most of them were fine, apart from one, where I am still unable to understand the issue.

When approaching this project, the approach I took meant I struggled with some timings, especially towards the end of the project. This was caused by misallocating my time when completing the initial plan, and underestimating the amount of time I would need to complete some areas of the project (and overestimating the time required for others). In the future, I will more carefully consider the time required to complete tasks as I plan them in advance.

One area of my approach that was definitely successful was scheduling weekly meetings to review the progress on the project with my supervisor. This really enabled me to get help and advice if I was stuck on a problem each week. I was never stuck on a problem for too long, or at least I was always able to at least get some advice if nothing else.

Overall, I feel like the project went well. There were areas that I could have gone better, for example the problems with some of my code, but I feel despite this I still found good results. I feel that the report itself has gone quite well as I tend to struggle with writing long reports such as this one. Luckily however, although there have been a few blocks along the way, I feel the report has gone well.

9 Bibliography

- [1] H. Prunckhun, *Handbook of Scientific Methods of Inquiry for Intelligence Analysis*, Lanham, MD: Scarecrow Press, 2010.
- [2] R. Layton and P. A. Watters, "Chapter 1 The Automating of Open Source Intelligence," in *Automating Open Source Intelligence*, Waltham, MA, Syngress, 2016, pp. 1-21.
- [3] N. Guarino, D. Oberle and S. Staab, "What is an Ontology?," in *Handbook on ontologies*, Berlin Heidelberg, Springer, 2009, pp. 1-17.
- [4] M. Horridge, "A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.3," 24 Mar 2011. [Online]. Available: http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf. [Accessed 04 May 2017].
- [5] "http://protege.stanford.edu/ontologies/pizza/pizza.owl," 2011. [Online]. Available: <http://protege.stanford.edu/ontologies/pizza/pizza.owl>. [Accessed 04 May 2017].
- [6] Stanford Center for Biomedical Informatics Research, "Protege," 2016. [Online]. Available: <http://protege.stanford.edu/>. [Accessed 04 May 2017].
- [7] S. Jupp, "Resource Description Framework (RDF). Ontogenesis.," 2010. [Online]. Available: <http://ontogenesis.knowledgeblog.org/235>. [Accessed 04 May 2017].
- [8] W3C SPARQL Working Group, "SPARQL 1.1 Overview," 2013. [Online]. Available: <https://www.w3.org/TR/sparql11-overview/>. [Accessed 04 May 2017].
- [9] I. Goodfellow, Y. Bengio and A. Courville, "Machine Learning Basics," in *Deep Learning*, MIT Press, 2016, p. 98.
- [10] T. Luostarinen and O. Kohonen, "Using Topic Models in Content-Based News Recommender Systems," in *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013); May 22-24; 2013; Oslo University; Norway. NEALT Proceedings Series 16 No. 085*, Linköping University Electronic Press, 2013.
- [11] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research* 3, pp. 993-1022, 2003.
- [12] Ranks NL, "Stopword Lists," [Online]. Available: <http://www.ranks.nl/stopwords>. [Accessed 04 May 2017].
- [13] C. D. Manning, P. Raghavan and H. Schütze, "Stemming and Lemmatization," in *An Introduction to Information Retrieval*, Cambridge, Cambridge University Press, 2009, pp. 32-34.

- [14] BBC News, “‘No turning back’ on Brexit as Article 50 triggered,” 30 Mar 2017. [Online]. Available: <http://www.bbc.co.uk/news/uk-politics-39431428>. [Accessed 04 May 2017].
- [15] D. Boffey and L. O'Carroll, “Brexit: EU leaders to demand May respect citizens' residency rights,” 26 Apr 2017. [Online]. Available: <https://www.theguardian.com/politics/2017/apr/26/brexit-eu-leaders-to-demand-may-respect-citizens-residency-rights>. [Accessed 04 May 2017].
- [16] @Daily_Express, “Businesses call on Theresa May to STAY in single market as Article 50 triggered <http://ln.is/www.express.co.uk/fi/CyJXT> ... #brexitday #Brexit,” 29 Mar 2017. [Online]. Available: https://twitter.com/Daily_Express/status/847250535598116866. [Accessed 04 May 2017].
- [17] M. Bazzell, Open Source Intelligence Techniques: Resources for Searching and Analyzing Online Information, 5th ed., Independent, 2016.
- [18] R. Stevens, “Competency questions for ontologies,” 2014. [Online]. Available: <http://studentnet.cs.manchester.ac.uk/pgt/2014/COMP60421/slides/Week2-CQ.pdf>. [Accessed 04 May 2017].
- [19] RDFLib Team, “rdflib 4.2.2 documentation,” 2013. [Online]. Available: <https://rdflib.readthedocs.io/en/stable/>. [Accessed 04 May 2017].
- [20] R. Řehůřek, “gensim: Topic modelling for humans,” 2009. [Online]. Available: <https://radimrehurek.com/gensim/>. [Accessed 04 May 2017].
- [21] NLTK Project, “Natural Language Toolkit - NLTK 3.0 documentation,” 2015. [Online]. Available: <http://www.nltk.org/>. [Accessed 04 May 2017].
- [22] R. Řehůřek, “corpora.dictionary – Construct word<->id mappings,” 2009. [Online]. Available: <https://radimrehurek.com/gensim/corpora/dictionary.html>. [Accessed 04 May 2017].
- [23] R. Řehůřek, “models.ldamodel – Latent Dirichlet Allocation,” 2009. [Online]. Available: <https://radimrehurek.com/gensim/models/ldamodel.html>. [Accessed 04 May 2017].
- [24] NLTK Project, “nltk.tokenize package,” 2015. [Online]. Available: <http://www.nltk.org/api/nltk.tokenize.html>. [Accessed 04 May 2017].
- [25] NLTK Project, “nltk.stem package,” 2015. [Online]. Available: <http://www.nltk.org/api/nltk.stem.html>. [Accessed 04 May 2017].
- [26] M. F. Porter, “An algorithm for suffix stripping,” 1980. [Online]. Available: <https://tartarus.org/martin/PorterStemmer/def.txt>. [Accessed 04 May 2017].

- [27] RDFLib Team, “rdflib Package,” 2013. [Online]. Available: <http://rdflib.readthedocs.io/en/stable/apidocs/rdflib.html#id1>. [Accessed 04 May 2017].
- [28] Python Software Foundation, “8.11. pprint — Data pretty printer,” 2017. [Online]. Available: <https://docs.python.org/3/library/pprint.html>. [Accessed 04 May 2017].
- [29] Python Software Foundation, “21.8. urllib.parse — Parse URLs into components,” 2017. [Online]. Available: <https://docs.python.org/3/library/urllib.parse.html#module-urllib.parse>. [Accessed 04 May 2017].
- [30] A. Savand, “stop-words 2015.2.23.1,” 2015. [Online]. Available: <https://pypi.python.org/pypi/stop-words>. [Accessed 04 May 2017].
- [31] L. Ou-Yang, “Newspaper3k: Article scraping & curation,” 2016. [Online]. Available: <https://github.com/codelucas/newspaper>. [Accessed 04 May 2017].
- [32] NumPy developers, “NumPy,” 2017. [Online]. Available: <http://www.numpy.org/>. [Accessed 04 May 2017].
- [33] C. Gohlke, “Unofficial Windows Binaries for Python Extension Packages,” 2017. [Online]. Available: <http://www.lfd.uci.edu/~gohlke/pythonlibs/>. [Accessed 04 May 2017].
- [34] P. V. Biron and A. Malhotra, “W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes,” 5 Apr 2012. [Online]. Available: <https://www.w3.org/TR/xmlschema11-2/>. [Accessed 04 May 2017].
- [35] Twitter, inc, “API Overview,” 2017. [Online]. Available: <https://dev.twitter.com/overview/api>. [Accessed 04 May 2017].
- [36] Scrapy Developers, “Scrapy | A Fast and Powerful Scraping and Web Crawling Framework,” [Online]. Available: <https://scrapy.org/>. [Accessed 04 May 2017].
- [37] News API, “News API - A JSON API for live news and blog headlines,” 2017. [Online]. Available: <https://newsapi.org/>. [Accessed 04 May 2017].

10 Appendix

10.1 Appendix A: Output from new, similar link

#####

Model of links in Ontology:

#####

[(0,

'0.001*"leav" + 0.001*"eu" + 0.001*"brexit" + 0.001*"wale" + 0.001*"support" ' +
'0.001*"fund" + 0.001*"vote" + 0.001*"campaign" + 0.001*"referendum" + ' +
'0.001*"govern" + 0.001*"blair" + 0.001*"peopl" + 0.001*"minist" + ' +
'0.001*"10" + 0.001*"uk" + 0.001*"cameron" + 0.001*"remain" + 0.001*"leader" ' +
'0.001*"parti" + 0.001*"busi"'),

(1,

'0.041*"eu" + 0.025*"leav" + 0.022*"wale" + 0.020*"vote" + 0.019*"10" + ' +
'0.017*"fund" + 0.014*"support" + 0.014*"referendum" + 0.013*"brexit" + ' +
'0.011*"welsh" + 0.011*"govern" + 0.009*"uk" + 0.009*"receiv" + 0.009*"area" ' +
'0.008*"campaign" + 0.008*"cent" + 0.008*"announc" + 0.008*"cornwal" + ' +
'0.006*"year" + 0.006*"european"'),

(2,

'0.025*"may" + 0.019*"minist" + 0.015*"prime" + 0.013*"cameron" + ' +
'0.010*"work" + 0.010*"theresa" + 0.009*"secretari" + 0.009*"parti" + ' +
'0.009*"campaign" + 0.009*"leadership" + 0.009*"leadsom" + 0.008*"brexit" + ' +
'0.008*"peopl" + 0.008*"imag" + 0.007*"leav" + 0.007*"role" + ' +
'0.007*"countri" + 0.006*"eu" + 0.006*"10" + 0.006*"down"'),

(3,

'0.014*"eu" + 0.014*"referendum" + 0.014*"result" + 0.007*"leav" + ' +
'0.007*"vote" + 0.007*"uk" + 0.007*"campaign" + 0.007*"area" + ' +
'0.007*"countri" + 0.007*"london" + 0.007*"england" + 0.007*"scotland" + ' +
'0.007*"decis" + 0.007*"union" + 0.007*"european" + 0.007*"live" + ' +
'0.007*"pictur" + 0.007*"led" + 0.007*"interest" + 0.007*"full"'),

(4,

'0.026*"leav" + 0.024*"vote" + 0.021*"wale" + 0.017*"remain" + ' +
'0.017*"labour" + 0.012*"peopl" + 0.011*"welsh" + 0.011*"campaign" + ' +

'0.010*"leader" + 0.009*"eu" + 0.009*"cameron" + 0.009*"referendum" + '
'0.009*"voter" + 0.008*"parti" + 0.008*"minist" + 0.008*"uk" + 0.007*"polit" '
' + 0.006*"area" + 0.006*"call" + 0.006*"brexit"''),

(5,

'0.053*"wale" + 0.028*"fund" + 0.024*"eu" + 0.018*"money" + 0.013*"imag" + '
'0.013*"european" + 0.011*"caption" + 0.011*"call" + 0.011*"area" + '
'0.011*"2000" + 0.011*"market" + 0.011*"structur" + 0.011*"valley" + '
'0.010*"uk" + 0.008*"leav" + 0.008*"level" + 0.008*"highest" + 0.008*"west" '
' + 0.008*"trade" + 0.008*"vote"''),

(6,

'0.055*"fund" + 0.034*"busi" + 0.032*"wale" + 0.029*"eu" + 0.016*"social" + '
'0.010*"support" + 0.010*"project" + 0.009*"govern" + 0.009*"peopl" + '
'0.007*"welsh" + 0.007*"provid" + 0.007*"enterpris" + 0.007*"wallich" + '
'0.007*"small" + 0.006*"year" + 0.006*"alloc" + 0.006*"invest" + '
'0.006*"money" + 0.006*"work" + 0.006*"european"''),

(7,

'0.022*"leav" + 0.017*"eu" + 0.017*"vote" + 0.011*"remain" + 0.011*"uk" + '
'0.011*"referendum" + 0.011*"won" + 0.011*"52" + 0.011*"major" + 0.011*"48" '
' + 0.011*"cent" + 0.006*"brexit" + 0.006*"minist" + 0.006*"cameron" + '
'0.006*"wale" + 0.006*"campaign" + 0.006*"prime" + 0.006*"david" + '
'0.006*"street" + 0.006*"farag"''),

(8,

'0.032*"brexit" + 0.027*"blair" + 0.015*"eu" + 0.013*"campaign" + '
'0.013*"peopl" + 0.013*"govern" + 0.011*"leav" + 0.011*"vote" + '
'0.010*"referendum" + 0.010*"media" + 0.010*"toni" + 0.008*"minist" + '
'0.008*"uk" + 0.008*"britain" + 0.008*"mp" + 0.007*"labour" + 0.007*"prime" '
' + 0.007*"caption" + 0.007*"polit" + 0.007*"parti"''),

(9,

'0.017*"may" + 0.011*"1" + 0.006*"polit" + 0.006*"british" + 0.006*"year" + '
'0.006*"day" + 0.006*"thing" + 0.006*"die" + 0.006*"month" + 0.006*"member" '
' + 0.006*"lost" + 0.006*"earli" + 0.006*"don" + 0.006*"father" + '
'0.006*"born" + 0.006*"life" + 0.006*"outfit" + 0.006*"walk" + 0.006*"grew" '
' + 0.006*"politician"''),

```
(10,
'0.001*"wale" + 0.001*"may" + 0.001*"eu" + 0.001*"brexit" + 0.001*"vote" + '
'0.001*"fund" + 0.001*"leav" + 0.001*"busi" + 0.001*"peopl" + '
'0.001*"campaign" + 0.001*"prime" + 0.001*"minist" + 0.001*"uk" + '
'0.001*"labour" + 0.001*"imag" + 0.001*"remain" + 0.001*"parti" + '
'0.001*"cameron" + 0.001*"leader" + 0.001*"support"'),
(11,
'0.001*"leav" + 0.001*"eu" + 0.001*"vote" + 0.001*"brexit" + '
'0.001*"campaign" + 0.001*"peopl" + 0.001*"wale" + 0.001*"labour" + '
'0.001*"fund" + 0.001*"blair" + 0.001*"minist" + 0.001*"parti" + '
'0.001*"leader" + 0.001*"remain" + 0.001*"referendum" + 0.001*"uk" + '
'0.001*"govern" + 0.001*"welsh" + 0.001*"area" + 0.001*"mp"'),
(12,
'0.001*"leav" + 0.001*"vote" + 0.001*"labour" + 0.001*"minist" + '
'0.001*"campaign" + 0.001*"peopl" + 0.001*"cameron" + 0.001*"wale" + '
'0.001*"remain" + 0.001*"referendum" + 0.001*"eu" + 0.001*"brexit" + '
'0.001*"prime" + 0.001*"parti" + 0.001*"may" + 0.001*"leader" + 0.001*"uk" + '
'0.001*"support" + 0.001*"polit" + 0.001*"secretari"')])
```

#####

Model of link to compare:

#####

```
[(0,
'0.028*"wale" + 0.019*"leav" + 0.017*"vote" + 0.016*"welsh" + 0.011*"labour" '
'+ 0.010*"campaign" + 0.009*"voter" + 0.009*"remain" + 0.008*"peopl" + '
'0.008*"leader" + 0.008*"eu" + 0.008*"polit" + 0.008*"bbc" + 0.007*"support" '
'+ 0.007*"jone" + 0.007*"minist" + 0.007*"referendum" + 0.007*"area" + '
'0.006*"surpris" + 0.006*"govern"')])
```

#####

Would you like to insert this link into the ontology? (y/n):

10.2 Appendix B: Output from new, identical link

#####

Model of links in Ontology:

#####

[(0,

'0.055*"fund" + 0.033*"busi" + 0.032*"wale" + 0.029*"eu" + 0.015*"social" + '
'0.010*"support" + 0.010*"project" + 0.009*"govern" + 0.009*"peopl" + '
'0.007*"welsh" + 0.007*"small" + 0.007*"provid" + 0.007*"enterpris" + '
'0.007*"wallich" + 0.006*"european" + 0.006*"money" + 0.006*"invest" + '
'0.006*"alloc" + 0.006*"financ" + 0.006*"year"')],

(1,

'0.001*"eu" + 0.001*"vote" + 0.001*"fund" + 0.001*"minist" + 0.001*"brexit" + '
' + 0.001*"campaign" + 0.001*"wale" + 0.001*"leav" + 0.001*"may" + '
'0.001*"peopl" + 0.001*"busi" + 0.001*"remain" + 0.001*"parti" + '
'0.001*"prime" + 0.001*"countri" + 0.001*"cameron" + 0.001*"labour" + '
'0.001*"support" + 0.001*"back" + 0.001*"mp"')],

(2,

'0.001*"eu" + 0.001*"fund" + 0.001*"wale" + 0.001*"busi" + 0.001*"leav" + '
'0.001*"minist" + 0.001*"may" + 0.001*"vote" + 0.001*"peopl" + '
'0.001*"campaign" + 0.001*"govern" + 0.001*"social" + 0.001*"prime" + '
'0.001*"cameron" + 0.001*"support" + 0.001*"brexit" + 0.001*"work" + '
'0.001*"parti" + 0.001*"uk" + 0.001*"remain"')],

(3,

'0.017*"may" + 0.011*"1" + 0.006*"year" + 0.006*"polit" + 0.006*"member" + '
'0.006*"lost" + 0.006*"day" + 0.006*"die" + 0.006*"own" + 0.006*"don" + '
'0.006*"earli" + 0.006*"chose" + 0.006*"100" + 0.006*"born" + 0.006*"life" + '
'0.006*"politician" + 0.006*"item" + 0.006*"car" + 0.006*"wife" + '
'0.006*"british"')],

(4,

'0.001*"minist" + 0.001*"leav" + 0.001*"vote" + 0.001*"cameron" + '
'0.001*"prime" + 0.001*"remain" + 0.001*"labour" + 0.001*"eu" + '

'0.001*"campaign" + 0.001*"may" + 0.001*"wale" + 0.001*"brexit" + 0.001*"uk" ' + 0.001*"leader" + 0.001*"peopl" + 0.001*"welsh" + 0.001*"parti" + ' '0.001*"referendum" + 0.001*"secretari" + 0.001*"support"'),

(5,

'0.027*"leav" + 0.027*"eu" + 0.024*"vote" + 0.014*"referendum" + ' '0.013*"wale" + 0.013*"remain" + 0.011*"brexit" + 0.011*"labour" + ' '0.011*"uk" + 0.011*"10" + 0.010*"campaign" + 0.009*"cameron" + ' '0.009*"support" + 0.009*"fund" + 0.008*"peopl" + 0.008*"area" + ' '0.008*"london" + 0.007*"leader" + 0.007*"welsh" + 0.007*"parti"'),

(6,

'0.016*"leav" + 0.014*"brexit" + 0.013*"wale" + 0.013*"campaign" + ' '0.012*"vote" + 0.012*"minist" + 0.010*"peopl" + 0.010*"eu" + 0.009*"parti" ' + 0.009*"labour" + 0.009*"blair" + 0.009*"may" + 0.008*"govern" + ' '0.007*"welsh" + 0.007*"leader" + 0.007*"prime" + 0.006*"referendum" + ' '0.006*"remain" + 0.006*"work" + 0.006*"support"'),

(7,

'0.052*"wale" + 0.029*"eu" + 0.029*"fund" + 0.017*"money" + 0.014*"vote" + ' '0.014*"leav" + 0.014*"uk" + 0.012*"imag" + 0.012*"european" + 0.010*"call" ' + 0.010*"area" + 0.010*"caption" + 0.010*"2000" + 0.010*"market" + ' '0.010*"valley" + 0.010*"structur" + 0.007*"welsh" + 0.007*"west" + ' '0.007*"trade" + 0.007*"level"'),

(8,

'0.001*"fund" + 0.001*"busi" + 0.001*"eu" + 0.001*"wale" + 0.001*"peopl" + ' '0.001*"support" + 0.001*"social" + 0.001*"work" + 0.001*"welsh" + ' '0.001*"provid" + 0.001*"project" + 0.001*"wallich" + 0.001*"govern" + ' '0.001*"may" + 0.001*"enterpris" + 0.001*"minist" + 0.001*"deliv" + ' '0.001*"year" + 0.001*"brexit" + 0.001*"small"'),

(9,

'0.028*"may" + 0.019*"prime" + 0.019*"minist" + 0.019*"imag" + ' '0.014*"cameron" + 0.014*"secretari" + 0.014*"caption" + 0.012*"down" + ' '0.012*"street" + 0.009*"eu" + 0.009*"peopl" + 0.009*"queen" + 0.009*"media" ' +

```
'+ 0.009*"role" + 0.009*"copyright" + 0.009*"theresa" + 0.007*"govern" + '
'0.007*"brexit" + 0.007*"uk" + 0.007*"work"),
(10,
'0.001*"eu" + 0.001*"wale" + 0.001*"brexit" + 0.001*"leav" + 0.001*"vote" + '
'0.001*"peopl" + 0.001*"minist" + 0.001*"govern" + 0.001*"fund" + '
'0.001*"campaign" + 0.001*"support" + 0.001*"remain" + 0.001*"prime" + '
'0.001*"caption" + 0.001*"uk" + 0.001*"blair" + 0.001*"welsh" + '
'0.001*"labour" + 0.001*"referendum" + 0.001*"britain"),
(11,
'0.013*"referendum" + 0.013*"cent" + 0.007*"eu" + 0.007*"leav" + '
'0.007*"brexit" + 0.007*"campaign" + 0.007*"minist" + 0.007*"remain" + '
'0.007*"prime" + 0.007*"cameron" + 0.007*"1" + 0.007*"johnson" + '
'0.007*"david" + 0.007*"street" + 0.007*"bori" + 0.007*"down" + '
'0.007*"farag" + 0.007*"won" + 0.007*"nigel" + 0.007*"agre"),
(12,
'0.001*"brexit" + 0.001*"may" + 0.001*"minist" + 0.001*"countri" + '
'0.001*"parti" + 0.001*"work" + 0.001*"leav" + 0.001*"prime" + 0.001*"peopl" '
'+ 0.001*"cameron" + 0.001*"leadsom" + 0.001*"campaign" + 0.001*"fund" + '
'0.001*"leadership" + 0.001*"labour" + 0.001*"eu" + 0.001*"support" + '
'0.001*"blair" + 0.001*"wale" + 0.001*"theresa"]]
```

#####

Model of link to compare:

#####

```
[(0,
'0.020*"leav" + 0.019*"vote" + 0.017*"remain" + 0.015*"labour" + '
'0.011*"peopl" + 0.010*"cameron" + 0.009*"parti" + 0.008*"leader" + '
'0.008*"brexit" + 0.008*"uk" + 0.008*"victori" + 0.008*"campaign" + '
'0.008*"referendum" + 0.007*"corbyn" + 0.007*"call" + 0.007*"minist" + '
'0.007*"london" + 0.007*"eu" + 0.006*"nation" + 0.006*"prime"]]
```

#####

Would you like to insert this link into the ontology? (y/n):

10.3 Appendix C: Output from dissimilar link

#####

Model of links in Ontology:

#####

[(0,

'0.001*"wale" + 0.001*"eu" + 0.001*"vote" + 0.001*"fund" + 0.001*"brexit" + '
'0.001*"leav" + 0.001*"campaign" + 0.001*"labour" + 0.001*"remain" + '
'0.001*"uk" + 0.001*"peopl" + 0.001*"area" + 0.001*"call" + '
'0.001*"referendum" + 0.001*"parti" + 0.001*"european" + 0.001*"imag" + '
'0.001*"minist" + 0.001*"britain" + 0.001*"money"'),

(1,

'0.001*"eu" + 0.001*"wale" + 0.001*"brexit" + 0.001*"vote" + 0.001*"leav" + '
'0.001*"fund" + 0.001*"govern" + 0.001*"uk" + 0.001*"peopl" + '
'0.001*"support" + 0.001*"campaign" + 0.001*"referendum" + 0.001*"minist" + '
'0.001*"busi" + 0.001*"10" + 0.001*"remain" + 0.001*"labour" + 0.001*"blair" '
'+ 0.001*"parti" + 0.001*"european"'),

(2,

'0.025*"leav" + 0.024*"vote" + 0.021*"remain" + 0.018*"labour" + '
'0.013*"peopl" + 0.012*"cameron" + 0.011*"parti" + 0.009*"campaign" + '
'0.009*"leader" + 0.009*"victori" + 0.009*"brexit" + 0.009*"uk" + '
'0.009*"referendum" + 0.008*"minist" + 0.008*"call" + 0.008*"corbyn" + '
'0.008*"eu" + 0.008*"london" + 0.006*"expect" + 0.006*"close"'),

(3,

'0.028*"may" + 0.019*"imag" + 0.019*"minist" + 0.019*"prime" + '
'0.014*"secretari" + 0.014*"cameron" + 0.014*"caption" + 0.012*"down" + '
'0.012*"street" + 0.009*"peopl" + 0.009*"eu" + 0.009*"media" + '
'0.009*"theresa" + 0.009*"queen" + 0.009*"role" + 0.009*"copyright" + '
'0.007*"brexit" + 0.007*"govern" + 0.007*"uk" + 0.007*"10"'),

(4,

'0.001*"eu" + 0.001*"brexit" + 0.001*"fund" + 0.001*"blair" + 0.001*"may" + '
'0.001*"leav" + 0.001*"govern" + 0.001*"campaign" + 0.001*"minist" + '

'0.001*"vote" + 0.001*"wale" + 0.001*"peopl" + 0.001*"busi" + 0.001*"polit" ' + 0.001*"prime" + 0.001*"work" + 0.001*"parti" + 0.001*"support" + ' 0.001*"uk" + 0.001*"british"),

(5,

'0.015*"referendum" + 0.011*"eu" + 0.011*"may" + 0.011*"1" + 0.008*"leav" + ' 0.008*"campaign" + 0.008*"result" + 0.008*"cent" + 0.004*"vote" + ' 0.004*"thursday" + 0.004*"remain" + 0.004*"brexit" + 0.004*"cameron" + ' 0.004*"uk" + 0.004*"countri" + 0.004*"minist" + 0.004*"prime" + ' 0.004*"live" + 0.004*"midland" + 0.004*"3"),

(6,

'0.001*"eu" + 0.001*"leav" + 0.001*"cameron" + 0.001*"vote" + 0.001*"minist" ' + 0.001*"may" + 0.001*"campaign" + 0.001*"wale" + 0.001*"remain" + ' 0.001*"referendum" + 0.001*"labour" + 0.001*"brexit" + 0.001*"peopl" + ' 0.001*"fund" + 0.001*"support" + 0.001*"10" + 0.001*"govern" + ' 0.001*"prime" + 0.001*"leader" + 0.001*"uk"),

(7,

'0.080*"blair" + 0.031*"toni" + 0.013*"duncan" + 0.013*"urg" + ' 0.011*"promin" + 0.011*"lib" + 0.010*"control" + 0.010*"bulli" + ' 0.009*"bill" + 0.009*"brexit" + 0.008*"debat" + 0.008*"arrog" + ' 0.008*"true" + 0.007*"dem" + 0.007*"rise" + 0.006*"horror" + 0.006*"tactic" ' + 0.006*"knowledg" + 0.006*"undemocrat" + 0.006*"nick"),

(8,

'0.001*"eu" + 0.001*"fund" + 0.001*"wale" + 0.001*"brexit" + 0.001*"may" + ' 0.001*"peopl" + 0.001*"vote" + 0.001*"leav" + 0.001*"uk" + 0.001*"govern" + ' 0.001*"prime" + 0.001*"minist" + 0.001*"european" + 0.001*"caption" + ' 0.001*"cameron" + 0.001*"campaign" + 0.001*"referendum" + 0.001*"secretari" ' + 0.001*"imag" + 0.001*"busi"),

(9,

'0.055*"fund" + 0.033*"busi" + 0.032*"wale" + 0.029*"eu" + 0.015*"social" + ' 0.010*"support" + 0.010*"project" + 0.009*"peopl" + 0.009*"govern" + ' 0.007*"welsh" + 0.007*"provid" + 0.007*"small" + 0.007*"enterpris" + '

```
'0.007*"wallich" + 0.006*"work" + 0.006*"money" + 0.006*"european" + '
'0.006*"invest" + 0.006*"alloc" + 0.006*"deliv"),
(10,
'0.035*"wale" + 0.027*"leav" + 0.024*"vote" + 0.018*"welsh" + 0.013*"labour" '
'+ 0.011*"eu" + 0.011*"campaign" + 0.011*"remain" + 0.010*"voter" + '
'0.009*"uk" + 0.009*"peopl" + 0.009*"leader" + 0.009*"polit" + 0.009*"bbc" + '
'0.007*"support" + 0.007*"referendum" + 0.007*"area" + 0.007*"minist" + '
'0.007*"jone" + 0.006*"govern"),
(11,
'0.001*"wale" + 0.001*"eu" + 0.001*"fund" + 0.001*"vote" + 0.001*"leav" + '
'0.001*"brexit" + 0.001*"may" + 0.001*"peopl" + 0.001*"govern" + 0.001*"uk" '
'+ 0.001*"remain" + 0.001*"imag" + 0.001*"minist" + 0.001*"campaign" + '
'0.001*"welsh" + 0.001*"support" + 0.001*"labour" + 0.001*"european" + '
'0.001*"work" + 0.001*"money"),
(12,
'0.024*"eu" + 0.017*"wale" + 0.017*"brexit" + 0.016*"leav" + 0.012*"vote" + '
'0.012*"fund" + 0.011*"campaign" + 0.009*"may" + 0.009*"minist" + '
'0.009*"govern" + 0.008*"support" + 0.008*"uk" + 0.008*"10" + '
'0.008*"referendum" + 0.008*"peopl" + 0.007*"parti" + 0.006*"countri" + '
'0.006*"prime" + 0.006*"work" + 0.006*"european"]]
```

#####

Model of link to compare:

#####

```
[(0,
'0.036*"wale" + 0.028*"game" + 0.019*"win" + 0.017*"england" + '
'0.015*"warburton" + 0.015*"februari" + 0.013*"bbc" + 0.013*"25" + '
'0.013*"saturday" + 0.013*"team" + 0.013*"scotland" + 0.013*"improv" + '
'0.011*"jone" + 0.011*"march" + 0.011*"ve" + 0.011*"gmt" + 0.011*"nation" + '
'0.009*"minut" + 0.009*"manag" + 0.009*"cardiff"]]
```

#####

Would you like to insert this link into the ontology? (y/n):