Hansard Historical Sentiment Analysis and Comparison

Final Report for CS39440 Major Project

Author: Adam Neaves (adn2@aber.ac.uk)
Supervisor: Dr. Amanda Clare (afc@aber.ac.uk)

4th March 2018
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Artificial Intelligence and Robotics (GH7P)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Adam Neaves

Date: 04/05/2018

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Adam Neaves

Date: 04/05/2018

Acknowledgements

I am grateful to...

I’d like to thank...

Abstract

Politics affects all aspects of a persons life. The results of debates in Parliament may have a profound influence on an individuals life, and knowing how your local MP speaks and the opinions they express during these debates could prove useful. Websites, such as They Work For You, show a user how their local MP votes, and how often they attend debates, ask questions, and other information that may allow the user to make informed decisions when voting during elections.

The aim of the Hansard Sentiment Analysis Tool is to provide a tool to automatically detect the sentiment of statements made during political debates, and to compare it with sentiment expressed about the topic previously, or compare it with sentiment expressed by the same MP. The sentiment analysis will use a machine learning approach, where a model will be trained on labelled datasets generated from the source data.

Being able to view how sentiment changed over time, expecially about certain topics, could prove useful for a user who wants to not only know how an MP votes, but also how they represent themselves and their constituency in parliament. If an MP is seen to change opinion on a topic as time passes, this information could be used to keep voters informed.

This report will document the process of designing and developing the Hansard Sentiment Analysis Tool, highlighting any challenges encountered during development, and also the results of the technical work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Hansard Dataset

The Hansard Dataset is a set of documents produced by the British Parliament, which began in the 18th and 19th century. These documents contain reports and details of debates in the House of Commons, going back to the year 1803. Eventually, in 1907, these reports were made official and started being produced by Parliament itself, becoming The Official Report, though still unofficially known as Hansard. Along with becoming official, a report was officially defined as being one:

> which, though not strictly verbatim, is substantially the verbatim report, with repetitions and redundancies omitted and with obvious mistakes corrected, but which on the other hand leaves out nothing that adds to the meaning of the speech or illustrates the argument" [1]

Hansard is available in a variety of versions. The most commonly used and best known version is the Daily Hansard, which appears each morning and reports of the previous days proceedings. However, access to this is via an API that only provides the most recent 7 days. For this project, most training and processing will be done on the Historical Hansard dataset, which is a dataset containing all 6 series of Hansard, between 1803 to 2004, though it is expected that some of the older documents will be less useful for this project due to the likelihood of them using outdated speech that would no longer be relevant.

The Historical Hansard Dataset is available online in an XML format. Multiple documents per series are available, each document covering a few days debates at most. It is a large dataset, reaching around 10Gb in size in total. Most of the documents available are scanned from hard copies, rather than typed up directly, meaning there is a possibility of small errors from the scanning process that may have to be dealt with. Additionally, from preliminary looks, the data itself appears to be only loosely formatted, and each of the six series appear to be formatted in a slightly different way, so any system designed to read this data will have to be capable of dealing with any changes to formatting.

1.1.2 Natural language Processing

Natural Language Processing (NLP) is the process of getting a computer to read and understand written text. Computers are very good at dealing with numbers and performing complex calculations at high speed but are not as good at understand spoken or written language. Because of this, a large part of NLP is the act of processing the data, or text, to make it easier for the computer to understand and work with. NLP covers multiple topics, such as Named Entity Recognition, part of Speech Tagging, and Sentence Boundary Disambiguation. However, the part this project is mainly interested is the act of Sentiment Analysis.

Sentiment Analysis, also known as Opinion Mining, is the process of identifying and extracting the opinions expressed in a piece of text. It aims to determine the attitude of a speaker or writer towards a topic, or the overall polarity of a piece of text. This can be a judgement made by the writer or speaker, in the case of reviews, or the emotional state of the speaker or writer.

A basic version of Sentiment Analysis classifies the polarity of a piece of text, classifying it as either positive, negative or neutral. A more advanced version would be, for example, looking at emotions expressed in the text, classifying it as angry, happy, or sad, as some examples. A basic method used can be to compare a piece to two lists of words, one a list of words that usually denote a positive polarity, and one that usually denotes a negative polarity. A system can then simply count the number of positive and negative words in a piece of text, account for any negation (Saying not great would change the word great from a positive to a negative word, for instance) and whichever type of word was most common would denote the piece of texts sentiment. However, this method is likely only useful for those pieces of text where its known that strong sentiment is likely to be expressed in a simple enough manner, in text such as a review.

Stance Detection is another aspect of Natural Language Processing, similar to Sentiment Analysis. However, the difference here is that Stance Detection sets out to classify the relative stance of two pieces of text, classifying whether the texts agrees with, disagrees with, discusses, or are unrelated to each other. An example would be detecting the stance of a news article compared to its headline. This may be more applicable to the Hansard Dataset than Sentiment Analysis as the members of parliament are likely to be expressing some form of stance on a topic that they are debating, but also somewhat more complicated to do, due to the additional classes involved.

1.1.3 Related Work

In reasearching the potential design of project, a few relevant pieces of work done by others were discovered, some of which had a useful impact on the design of this project.

Towards sentiment analysis on parliamentary debates in Hansard [2] is a paper which discussed the progress made by a group of PHD researchers towards applying classic sentiment analysis techniques to the Hansard dataset, such as word association. The paper details the proposed approach to sentiment analysis, by using *heuristic classifiers based on the use of statistical and syntactic clues in the text* and using a sentiment lexicon base known as the MPQA corpus to identify sentences containing known positive or negative words. They first classify a sentence using this lexicon, annotating sentences as possitve or negative depending on the number of positive or negative words, before then applying syntactic clues to improve the classification, such as the presence of negations, such as not or never, and the inclusion of intensifying adverbs such as very. The paper reports an average of 43% correctly annotated sentences, claiming that the correctly annotated

sentences were those without compound opinions, sarcasm and comparative sentences, showing that the style of debate speech renders their syntactic and lexical based approach insufficient for the task.

They Work For You [3] is a website that allowed the user to search for their local MP via post code, and the site can then display the voting patterns for that MP, along with information about how often their votes align with their parties votes, and shows examples of appearences made by that MP and what they said. The source code is publicly available on Github and uses python for a large part of their code base. Whilst it does not appear that they use any form of sentiment analysis, its still a good example of the sort of thing that can be done using the parlimentary data, and would likely be well suplimented by my project, allowing them to also show how an MP might speak in debates, as well as how they vote.

The Fake News Challenge [4] is a challenge set up to explore how artificial intelligence technologies could be leveraged to combat fake news. and aims to eventually produce a tool that can help human fact checkers tell if a news story is a hoax, or intentionally misleading. The first part of the challenge involved the use of Stance Analaysis on a series of news articles, comparing the contents of the article with the headline, to tell if the article contents agree with the headline or not. As the project is set up as a competition, multiple teams submitted solutions to the problem, showing a variety of techniques in solving this problem.

1.1.4 Technical Research

Before any form of planning could begin for the project, some research on the kinds of technologies available was required. Three topics had to be researched, namely the language to be used, what methods were available for sentiment analysis, and what was available to extract the data from its original form to something more usable.

For sentiment analysis, and other required NLP tools, the Natural Language Toolkit (NLTK) [5] was found. This Python package provides methods and classes for a majority of NLP tasks, including everything required for this project. Additionally, its well documented, as it is commonly used by other projects that require some form of NLP. This means it would be easy to find solutions to any problem encountered during development, as it is highly likely someone else using the same package has encountered a similar issue and documented a solution online. Due to this, it was quickly decided that the NLTK package would be used for all NLP requirements, which also meant the language to be used would be Python.

Once a language was selected, some form of parsing tool had to be discovered. As the data is provided in XML, a commonly used semi-structured database format, the chosen parser solution had to be able to parse XML. An often used package for Python that could do this is lxml, a package which could read in XML data structures and translate them into its own set of classes to be used in Python. However, lxml expects well structured data, whereas the hansard dataset is not as well organised. For this reason, it was decided that BeautifulSoup4 [6] would be used, a module designed to parse less structured data, in HTML or XML. It makes use of the lxml module, but provides methods that allow the parsing of data whose structure is unknown.

1.2 Analysis

Following along from the background research, some decisions were made on how this project would proceed, and what challenges were expected. Additionally, the design of the overall system had to be developed, and the developmental process.

Due to the size and complexity of some of the source data, it was decided that the system would not be able to work directly with the original data without some form of intermediary parsing system. Thus, part of the project was to develop a parser that would get all relevant data from the original files, and reorganize them into a more useful format. It was therefore also necessary to decide on the structure of the data once parsed, and how this data would be stored and accessed by the rest of the system. It would also be necessary to decide what exactly from the original data was relevant to the project. Additionally, it was important that speech be attributed to the correct member of parliament. It often appeared that the way an MP was referenced in the data would change, going from a full title, honorific and name to just surname. It was important that these different forms of name be recognized as the same person, otherwise speech attributed to just one person would be seen as being said by different people, reducing the accuracy of any searching. This could be done using Named Entity Recognition, an aspect of NLP.

Any form of supervised machine learning method requires training and testing datasets. Due to the nature of the data being used, there are no preexisting sets of annotated data available, and thus the datasets used must be hand annotated. A tool designed to do this must therefore be produced that can assist in this lengthy process, allowing a user to generate a set of annotated data that can then be used to train an Artificial Intelligence.

1.3 Process

The process selected for this project was Feature Driven Development (FDD), an agile styled methodology designed to focus on delivering working blocks of software repeatedly. Though this methodology is not usually designed for a single developer, its overall design worked well enough for this project. It began with an overall design of the system (see the diagram above), and then this design was broken down into a list of features, by breaking down the overall design into functional sections, represented by the blocks in the diagram. Each of those is decomposed further into a list of features for each one, which can then be targeted as milestones during development. Because these features are relatively small, completing them is a small task, and keeping track of the progress on each feature can give a decent report on the progress of the project as a whole.

For version control, git, and Github.com was used to maintain versions of code. This can then also be used as a backup service, with the online version of the code hosted on Github being a form of backup, and allowing the code to be transferred between multiple machines with relative ease.

A to-do page was also maintained on Trello, which is an online resource for producing Kanban style to-do lists. There were four columns in the Kanban Board:

- A to-do column for tasks that needed doing that were not yet complete,
- A Doing column for tasks currently in progress,

- A Blocked column for tasks that cannot be started until another task has been completed, or tasks that have been started but cannot continue until another task is done or some event is complete,
- A Done column for tasks that have been completed.

In utilizing this tool for the organisation of the project, development was planned out. Each ”Card” of the board represented one task, and coule be moved between columns. More complicated tasks were broken down into separate tasks, or a checklist could be added to a card, should the task need to remain as a single task for whatever reason.

Chapter 2

Design

In FDD, large, detailed design documents are not necessary. A simple overview of desired features and a breakdown of the overall system are sufficient, and are what will be detailed in this chapter.

2.1 Overall Architecture

As the project was developed using Feature Driven Development (FDD), the initial design of the system focused on the desired features of the system. These features were:

- Download the dataset.
- Parse the original data into a more useful format.
- Allow a user to annotate the data with sentiment, to generate training and testing datasets.
- Train an AI algorithm on the datasets generated.
- Search the parsed data to find speech about a certain topic, or by a particular Member of Parliament.
- Use the trained AI algorithm to extract sentiment from the speech found.
- Display a comparison of the sentiment expressed about a topic or by an MP over time.

Using these features, the overall system can be designed, and an unofficial class diagram can be created that can help guide implementation.

As seen in fig.2.1, the system is broken down into functional blocks, each representing a distinct part of the system. They were split this way to maintain readability in the source code. Whilst it would have been possible to keep all functionality in a single file, it would have made maintaining the code very difficult. Each functional block is therefore a separate class in the Python source code, so any interaction between them is easy to manage.

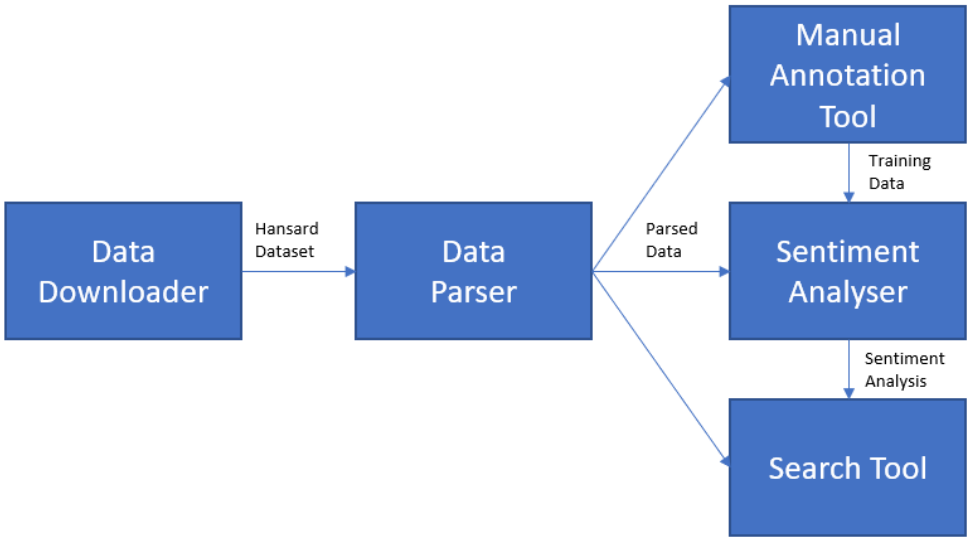

Figure 2.1: Functional block diagram of the overall system. Arrows represents data movement

2.1.1 Data Downloader

The Data Downloader is designed to download all of the Hansard Dataset from the site on-line, and store it locally for use by the rest of the system. It is likely that this first block will only need to be run a single time, but still proves useful in ensuring all the data is downloaded.

The files are to be downloaded from the Hansard Archive page. Each of the six series has a sub-directory under that root page, that contain all the data files. The data files have a distinct naming pattern that can be referenced using something like Regular Expressions in order to automate the downloading and searching of the files.

Originally, this was not going to be a part of the main system, as it is possible to download the data manually. However, only ten files are displayed on the web page each time, and some of the series have up to a thousand files to download. This can be seen in fig.2.2

2.1.2 Data Parser

The Parser is designed to read the original files downloaded by the Data Downloader, and save all relevant data in separate location, in a more consistent layout than that of the original dataset. This should allow any usage of the data from other parts of the system to be much simpler, and thus faster and less prone to error. It should ensure speech is always attributed to the correct person, even if their names are presented differently. This means the Data Parser will have to utilize a part of NLP known as Name Disambiguation, which is not a simple task.

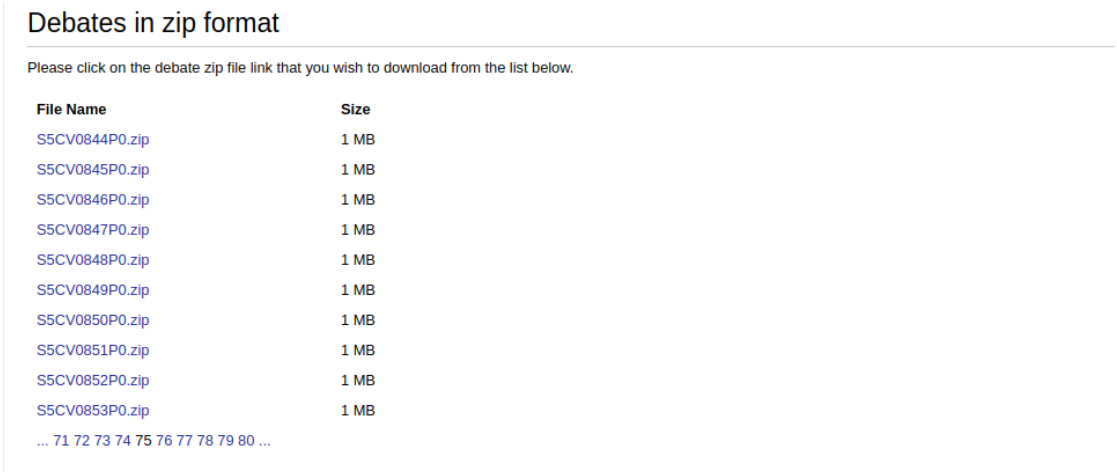

Figure 2.2: Screen capture from the Hansard Archive website, showing the number of files available

2.1.3 Manual Annotation Tool

The Manual Annotation Tool (the MAT) is designed to allow a user to create a testing and training dataset from the parsed data. It should show a selection of speech to the user, and ask them to annotate it as either a positive sentiment, negative sentiment, or neutral sentiment. These choices will be recorded, and can then be used to train a machine learning algorithm to extract sentiment from the remaining data. It would also provide the option to edit the text, the MP's name, or the topic title, to allow the user to correct typos transfered from the original dataset.

Originally, the MAT would display an entire paragraph of speech, displaying everything a Member of Parliament said about a topic on a particular day. However, this would often display too much text, with parts showing positive sentiment and other parts of the same text showing negative sentiment. Additionally it was realized that attempting to train an AI algorithm on these large blocks of text would be far too complex. For those reasons, the MAT was modified so that it would only display a single sentence at a time, allowing a more accurate way of annotating sentiment.

2.1.4 Sentiment Analyzer

The Sentiment Analyzer will use the datasets generated by the MAT and produce a model based off the data. It should also provide the ability to load a previously trained model, rather than spend time retraining every time the system is run. Once trained or loaded, it should be able to accept blocks of text, which it can then extract sentiment from, and return it.

As AI algorithms are trained on "Features" rather than plain text, the Sentiment Analyzer must have some method of converting a sentence or paragraph into a set of features for the algorithm to use. Following on from a tutorial found on Youtube [7], it appears that a good way to turn the text into a set of features is to use a dictionary, where the keys are the most used words from the training set, and the value is a True or False, for if the word is contained in the sentence or paragraph given. The potential issue with this is the loss of context, as separating the words completely will lose any negation, such as in the sentence "This is not good".

2.1.5 Search Tool

The Search Tool will be used by the user to search through the data for a particular Member of parliament, to get their speech and its sentiment, or for a particular topic, to get the sentiment expressed about that topic.

2.2 Data Design

As parts of the system were designed to modify the layout of the original data at times, the formatting of said data had to be designed as well.

2.2.1 Parsed Data

For the parser, the format of the data files that it produces was originally designed as thus:

```
<date dateformat="1984/04/13">
    <speech>
        <member>memberName</member>
        <topic>topicTitle</topic>
        <stance>POS/NEG</stance>
        "Actual Text of Speech would go Here"
    </speech>
    ....
</date>
....
```

The ellipses represent repeats, so inside the *Date* tags, multiple *speech* tags, all formatted like the one shown, can exist. A file may also contain multiple *Date* tags.

This, however, was causing some issues during development based around the size of the produced files, as each entire series of data was being parsed into the same file. In order to combat the slow speeds of accessing such large files, and other issues caused by the layout, the data format was changed. Now, each unique date found within the source had a separate file, and within this file the data was formatted in its final form. Each date file is an XML file, the structure of which can be represented as a tree, shown in fig.2.3.

Each file can contain multiple member tags, which each have the member of parliaments name as an attribute. Each member tag can contain multiple tags for the topics they discuss, each of which have the topic title as an attribute. Each topic tag can have multiple speech tags, each speech tag containing the verbatim copy of what the member said about the particular topic.

2.2.2 Annotated Data

The formatting of the annotated data files was also designed at this stage. It was decided that the files would be saved as CSV files (comma separated values), which is a simplistic database style

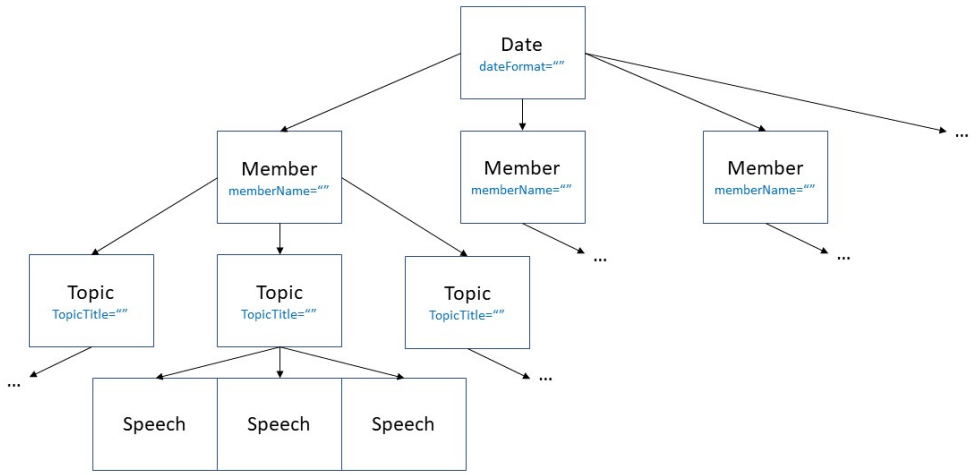

Figure 2.3: Diagram showing the structure of the parsed data, in XML. The *Date* tag is the root of the file

where a line in the file represents a single record, each value of the record being separated by a comma, hence the name. The layout was designed as the following:

Sentence; Sentiment; Member; Topic

As shown, it was decided that the best way of annotating full speech was to split it by sentence, so that each sentence would be annotated with its own sentiment. This would simplify the training process for the machine learning algorithms, as it was thought that trying to train it on too large a block of text would result in sub par results.

2.3 User Interface

Due to the complexity of the project, it was decided that the system would only be interacted with via the command line. This meant that, beyond basic menus, no GUI had to be designed for use. It was though that, should there be time during or after the project development, a GUI could be designed which worked by interacting with the command line interface in the background, but it was not considered system critical to have such a thing.

2.4 Algorithm Design

2.4.1 AI Algorithm Choice

There are many potential algorithms that may be chosen for this project. It may prove difficult to choose one, or more, without simply trying them. The design calls for the use of Naive Bayes at first, mainly as its one provided by the NLTK package. However, multiple other algorithms may be tested, and the results compared to decide on the final choice. Should multiple algorithms prove useful, a voting system could be implemented, which applies each algorithm, and they cast a vote to what sentiment they ”think” applies. The votes would then be counted and the result shown.

Naive Bayes Naives Bayes is a supervised algorithm with uses the statistical analysis of Bayes Theorem as to classify data. It treats each feature as completely distinct, which isn't accurate for NLP, but will may suffice for development of the system.

2.4.2 Parsing

The Data Parser needs to find and extract useful speech from the original data. Though said data is somewhat disorganized, there are exploitable patterns that the Parser can use to find as much speech as possible. The overall method is documented in the following pseudo-code:

```
FOR EACH FILE
    FOR EACH DATE TAG
        IF(DATE_FILE FOR DATE EXISTS)
            LOAD DATE_FILE CONTENTS INTO DATE_XML
        ELSE
            CREATE FILE
            CREATE DATE_XML
        FOR EACH CONTRIBUTION IN DATE_TAG
            GET CONTRIBUTION PARENT_TAG
            GET MEMBER NAME AS CHILD OF PARENT_TAG
            GET TITLE AS SIBLING OF PARENT_TAG
            GET SPEECH FROM CONTRIBUTION
            IF MEMBER HAS BEEN SEEN BEFORE IN CURRENT DATE_FILE
                IF TOPIC HAS BEEN DISCUSSED BY MEMBER BEFORE
                    ADD SPEECH TO TOPIC
                ELSE
                    CREATE NEW TOPIC
                    ADD SPEECH
            ELSE
                CREATE NEW MEMBER TAG
                ADD TOPIC TO MEMBER TAG
                ADD SPEECH TO TOPIC
            ADD SPEECH TO DATE_XML

    SAVE XML TO DATE_FILE (OVERWRITE)
```

This shows that, so long as the parser can find a tag in the source XML for the date, all other relevant information can be found in relation to the position of this date tag in the XML. It also shows the method of ensuring all speech by one Member of Parliament is correclty attributed to them, to avoid repeated mentions of the same MP in the same file.

Chapter 3

Implementation

3.1 Data Downloading

3.2 Data Parsing

3.2.1 Name Recognition and Disambiguation

Part of the difficulty in parsing the data accurately was ensuring speech spoken by one person was always attributed to that person, even though the references to that person may use multiple versions of their name. In order to correctly attribute speech to the speaker, without duplicating references to them, a method of NLP called ”Name Disambiguation” must be employed.

Name Disambiguation is the method of identifying proper names in text, and recognizing when two proper names refer to the same subject or person. [8] Many difficulties exist for such a task, as names can come in many forms. Proper nouns can often refer to multiple things. *Johnson and Sons* might refer to a business by that name; two people with the surnames *Johnson* and *Sons*; or someone called *Johnson* and their male children. In this example, the context of the sentence the name appears in may provide clues to which it is, but this is not always an option.

Thankfully, this project only needs to be able to recognize the names of the Members of Parliament mentioned in the Hansard Dataset, which displays more structure in the way it names people than normal human speech would.

```
<section>
<title>Disabled Persons (Air Travel)</title>
<p id="S6CV0001P0-00373">1. <member>Sir David Price</member> <membercontribution>asked the Secretary of State for Trade, in view of the fact
that 1981 is the International Year of Disabled People, if he will take steps with British airports and British airlines, respectively, to
improve facilities for air travel by disabled pepole, especially those confined to wheelchairs.</membercontribution></p>
<p id="S6CV0001P0-00374"><member>The Under-Secretary of State for Trade (Mr. Reginald Eyre)</member><membercontribution>: The provision of
facilities for disabled air travellers is a matter for the airports and airlines concerned, and a good dealhas been, and is being, done to
improve the lot of the handicapped when they travel by air. Airport and airline representatives have participated in the meetings reviewing the
special meeds of disabled passengers held under the auspices of my right hon. Friends the Secretary of State for Transport and the last
Minister for Social Security. We encouraged the widest possible airport and airline representation at last Thursday's conference on "Transport
without Handicap".</membercontribution></p>
```

Figure 3.1: An example from the original dataset, showing that names exist only within *member* tags

As can be seen in fig.3.1, the names needed for this project are always contained within a *member* tag. They still contain the full job title at times, but this means that the area the project needs to look for a name is well defined. It can be guaranteed that, if a member tag is encountered, it will

contain the name of the member.

The first attempted method of extracting the name from the *member* tag was using the *Named Entity Chunker* from NLTK, which is designed to extract the Named Entities from a piece of text and return them in a list. Using this, the name could be extracted, potentially along with other parts of the title (*The Under-Secretary of State for Trade (Mr. Reginald Eyre)* would be returned as a list of two names, *State for Trade* and *Mr. Reginald Eyre*). The thought was that the actual name desired could be chosen from that list by looking for the item that started with an Honorific, such as *Mr.*, *Mrs.* and other such titles. However, the Named Entity Chunker did not work consistently enough for it to be a viable solution for the name extractor, as it often separated the honorific from the name. Due to this, it could not be reliably used to extract the names, because there was no way to recognize, when it returned multiple names, which referred to the actual person, and which was a part of their job title.

The second method attempted, which was the one settled on for the project, was the use of Regular Expressions to search for the expected name format. This method relies on a lot of assumptions about the organization of the original dataset, but appears to be good enough for the current dataset. The assumptions are as follows:

- All Names start with an Honorific (Mr, Mrs, Sir etc)
- All Names End with a Surname
- The first time a person is seen, their full name and job title, if relevant, are presented

Following these assumptions, Regular Expressions could be used to find the honorific at the start of the name, then get everything following it until either the end of the text, or a piece of punctuation not commonly used in names, such as a bracket.

```
31  def extract_name(text):
32      # find a name in a sentence or piece of text
33      # will assume the first name given that includes an honorific (mr., Mrs, etc) is the correct name
34      punctuation = string.punctuation  # we want to get all the text from Mr/Mrs etc till it finds some punctuation
35      punctuation = punctuation.replace("-", "")  # however, - and . are valid parts of a name, so we still want them
36      punctuation = punctuation.replace(".", "")
37      name_rex = re.compile(r'(\bMr[ .]|\bMrs[ .]|\bSir[ .]|\bDr[ .])[^{}\\\n]+'.format(punctuation), re.IGNORECASE)
```

Figure 3.2: Extract of code from the Name Extractor Method.

Fig3.2 shows the regular expression used to extract a name from a piece of text. It searches for an Honorific to start it off, either *Mr*, *Mrs*, *Dr* or *Sir*. Once it find one of these honorifics it gets everything after it that is not a punctuation mark listed, as shown by the `[^{}\\\n]+` part at the end of the regular expression. The `{}` brackets in that part are replaced with the string called *punctuation*, generated by the string package, so that it becomes a list of all punctuation possible, including new line characters.

3.2.2 Name Matching

Once names can be reliably extracted from the source text, a method of comparing two names to see if they referred to the same person had to be developed. This becomes a very complicated issue, simplified only slightly by the assumptions mentioned previously.

Name One	Name Two	Same Person?
Mr. Adam Neaves	Mr. Adam Neaves	True
Mr. A. Neaves	Mr. Adam Neaves	True
Dr. Adam Neaves	Mr. Adam Neaves	False, wrong Honorific
Mr. B. Neaves	Mr. Adam Neaves	False, wrong First Name
Adam Neaves	Mr. Adam Neaves	True, despite missing honorific
Mr. A. B. Neaves	Mr. Adam Neaves	True, though slightly ambiguous
Mr. A. B. Neaves	Mr. Neaves	True, though again ambiguous

Table 3.1: Table showing the different ways names might or might not refer to the same person

For instance, as shown in table 3.1, first names might be shortened to just an initial, or even fully removed. Honorifics are not guaranteed to be present, but are very important if they are, as two different names may differ from only that.

The implemented algorithm for this project follows the same assumptions mentioned in section 3.2.1, and is designed to allow for some false negatives (wherein two names that *do* refer to the same person might not appear as the same person) to avoid any false positives at all, as speech accidentally attributed to the wrong person would be worse than accidentally having duplicate references to a person. The basic algorithm implemented is as follows:

```
if BOTH NAMES ARE EXACTLY THE SAME
    return True
SPLIT BOTH NAMES INTO COMPONENT WORDS
for EACH NAME
    if NAME CONTAINS 3 WORDS
        FIRST WORD IS HONORIFIC
        SECOND WORD IS FORENAME
        THIRD WORD IS SURNAME
    else
        FIRST WORD IS HONORIFIC
        LAST WORD IS SURNAME
        IGNORE ANYTHING ELSE
if FORENAME ONE and FORENAME TWO BOTH EXIST
    FORNAME_MATCH = FORNAME ONE == FORNAME TWO
else
    FORNAME_MATCH = True
SURNAME_MATCH = SURNAME ONE == SURNAME TWO
HONORIFIC_MATCH = HONORIFC ONE == HONORIFIC TWO

if SURNAME_MATCH, HONORIFIC_MATCH and FORENAME_MATCH ARE True
    return true
else return False
```

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Provide information in the body of your report and the appendix to explain the testing that has been performed. How does this testing address the requirements and design for the project?

How comprehensive is the testing within the constraints of the project? Are you testing the normal working behaviour? Are you testing the exceptional behaviour, e.g. error conditions? Are you testing security issues if they are relevant for your project?

Have you tested your system on "real users"? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1 Overall Approach to Testing

4.2 Automated Testing

4.2.1 Unit Tests

4.2.2 User Interface Testing

4.2.3 Stress Testing

4.2.4 Other types of testing

4.3 Integration Testing

4.4 User Testing

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

For example, if you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document as an appendix. In the body of your report you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.

If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification. Perhaps you use stories to keep track of the functionality and the 'future conversations'. It might not be relevant to include all of those in the body of your report. Instead, you might include those in an appendix.

There is a balance to be struck between what is relevant to include in the body of your report and whether additional supporting evidence is appropriate in the appendices. Speak to your supervisor or the module coordinator if you have questions about this.

Annotated Bibliography

[1] House of Commons Information Office, "Factsheet G17: The Official Report," *The Official Report*, no. August, 2010. [Online]. Available: https://www.parliament.uk/documents/commons-information-office/g17.pdf

[2] O. Onyimadu, K. Nakata, T. Wilson, D. Macken, and K. Liu, "Towards sentiment analysis on parliamentary debates in Hansard," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8388 LNCS, 2014, pp. 48–50.

A paper detailing the progress made on trying to apply standard sentimen analysis techniques to the hansard dataset. It discusses the fact that these normal approaches are inadequate for the style of speech used in parlimentary debates.

[3] "TheyWorkForYou: Hansard and Official Reports for the UK Parliament, Scottish Parliament, and Northern Ireland Assembly - done right." [Online]. Available: https://www.theyworkforyou.com/

[4] Fake News Challenge. (2017) Fake News Challenge. [Online]. Available: http://www.fakenewschallenge.org/www.fakenewschallenge.org

[5] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed., 2009, vol. 43, p. 479. [Online]. Available: http://www.amazon.com/dp/0596516495 9780596516499.

A guide to the Natural Language Toolkit for Python, a specific package that may be used for the project.

[6] L. Richardson. Beautiful Soup Documentation. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Documentation for BeautifulSoup4, a HTML/XML parser for Python, which could be useful for dealing with the large badly formatted XML documents.

[7] Harrison. NLTK with Python 3 for Natural Language Processing - YouTube - YouTube. [Online]. Available: https://www.youtube.com/playlist?list=PLQVvvaa0QuDf2JswnfiGkliBInZnIC4HL

[8] N. Wacholder, Y. Ravin, and M. Choi, "Disambiguation of Proper Names in Text," *Proceedings of the 5th Applied Natural Language Processing Conference*, 1997. [Online]. Available: https://pdfs.semanticscholar.org/1c17/946466ce0b17dec4bc79d3a7dfaf41dd843d.pdf

[9] H. E. Association. About Hansard - Commonwealth Hansard Editors Association. [Online]. Available: https://www.commonwealth-hansard.org/about-hansard.html

[10] C. J. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text." [Online]. Available: http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf

[11] Sentiment Analysis in Python with TextBlob and VADER Sentiment (also Dash p.6) - YouTube. [Online]. Available: https://www.youtube.com/watch?v=qTyj2R-wcks

[12] Word2Vec, Doc2vec & GloVe: Neural Word Embeddings for Natural Language Processing - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM. [Online]. Available: https://deeplearning4j.org/word2vec.html

[13] B. Riedel, I. Augenstein, G. P. Spithourakis, and S. Riedel, "A simple but tough-to-beat baseline for the Fake News Challenge stance detection task," jul 2017. [Online]. Available: http://arxiv.org/abs/1707.03264

[14] I. Augenstein, T. Rocktäschel, A. Vlachos, and K. Bontcheva, "Stance Detection with Bidirectional Conditional Encoding," jun 2016. [Online]. Available: http://arxiv.org/abs/1606.05464

[15] S. Dori-Hacohen, "Controversy Detection and Stance Analysis," *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1057–1057, 2015.

[16] W. Ferreira and A. Vlachos, "Emergent: a novel data-set for stance classification," pp. 1163–1168. [Online]. Available: http://aclweb.org/anthology/N/N16/N16-1138.pdf

[17] A. Aker, L. Derczynski, and K. Bontcheva, "Simple Open Stance Classification for Rumour Analysis," aug 2017. [Online]. Available: http://arxiv.org/abs/1708.05286

[18] P. Nagy, "Python NLTK Sentiment Analysis," 2017. [Online]. Available: https://www.kaggle.com/ngyptr/python-nltk-sentiment-analysis/notebook

[19] O. Onyimadu, K. Nakata, Y. Wang, T. Wilson, and K. Liu, "Entity-Based Semantic Search on Conversational Transcripts Semantic." Springer, Berlin, Heidelberg, 2013, pp. 344–349. [Online]. Available: http://link.springer.com/10.1007/978-3-642-37996-3{_}27

[20] J. Perkins. (2010) Text Classification For Sentiment Analysis - Naive Bayes Classifier. [Online]. Available: https://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/

An online article that describes how the author trained a Naive Bayes classifier on movie reviews using the NLTK for Python. It should serve as a good reference when looking to do something similar.

[21] H. M. Noble, *Natural Language Processing*, 1st ed., T. Addis, B. DuBoulay, and A. Tate, Eds. Oxford: Blackwell Scientific Publications, 1988, 0632015020.

This book provides a decent overview on Natural Language Processing for computer systems. It does not, however, mention any specifics for Sentiment Analysis.