Introduction

Motivation

While computational approaches are often applied to political texts, very few studies have ever specifically concerned a British corpus (a collection of written or spoken material stored on a computer and used to find out how language is used). After reading various papers that used natural language processing techniques on political corpora, I noted the most common studies concerned sentiment analysis of short texts, such as newspaper headlines or Tweets. This motivated me to investigate the task of performing sentiment analysis on longer pieces of text, to make the project more unique still.

The 2003 invasion of Iraq was an issue that cut across the political spectrum, which makes it an interesting topic for a sentiment analysis project, since someone's stance on the war cannot be easily determined from their views on other issues. Furthermore, the recent publication of the Iraq Inquiry (commonly known as the Chilcot Inquiry) and the ongoing situation in Iraq and Syria means that such a project is particularly timely. In addition to this, (as far as I'm aware) there haven't been any previous studies which have carried out computational sentiment analysis with a focus on war, which makes the project more unique still.

Problem Formulation

In this project, I look at how best to apply machine learning techniques to the carry out sentiment analysis on texts about the Iraq war. Since I can obtain labelled data relevant to the project without great difficulty (e.g. having to manually label it), I decided to consider this project as a supervised learning program. We can naturally simplify all stances on the Iraq war to be either pro-war or anti-war, thereby allowing the problem to be formulated as a binary classification task. One of the simplest implementations for a binary classifier is the naive Bayes classifier (described in prep-bayes), so I use this as a baseline. For reasons outlined in prep-svm, I use a support vector machine classifier as the default model.

When performing sentiment analysis, the corpus used is the most important resource. In prep-changes, I justify my choice of the Hansard as the primary corpus for the project. The Hansard is the set of transcripts from British parliamentary debates. As MPs vote on individual issues (including the invasion of Iraq), we can use an MP's voting record to determine their view on a topic and therefore automatically label the stance of any of their speeches on that topic. Since, manually labelling data is laborious, I label the speeches using voting records, despite the additional difficulty of matching up two datasets and the noise that this introduces to the data (discussed in prep-changes). Using this dataset means that we can essentially view the classifier produced as a system to predict how MPs will vote on an issue, given what they have said about the issue in the House of Commons.

Related Work While many studies into sentiment analysis have been based around political issues, since 2009 the majority of such research has concerned Tweets. The first study to use Twitter as its primary corpus was 'Twitter power: Tweets as electronic word of mouth' and there have since been countless studies following suit. In 2010, Pak, Alexander and Paroubek, Patrick proposed that Twitter could be used to determine public opinion, which was proven true later that year when sentiment analysis of Twitter provided predictions that paralleled the results of traditional election polls for the German federal election. This focus on Twitter is useful, but since most political decisions are made in Government and not on the internet, we should also use computational methods to learn more about how our MPs represent us in Parliament. Unfortunately, although it makes the project more interesting, analysing longer political texts presents more challenges than analysing Tweets, in part due to the lack of guidance from similar previous work.

In the US, there have been a small number of papers detailing sentiment analysis on transcripts of Congress debates. The results of these studies indicate that determining an MP's stance on the Iraq war from their speeches in the House of Commons may be possible, however these papers use transcripts to determine a politician's political party, which is likely to be more clear-cut than their stance on a particular issue.

The lack of relevant works to this project highlights its uniqueness, which is one of the principal motivations for the project.

Overview of the Project

In [prep]Chapter , I formally define the project, then outline the relevant models and algorithms before discussing decisions I made about how best to implement the project. [impl]Chapter details the development of the system, while [eval]Chapter assesses the success of the project, in part by comparing the performance of various classifier optimisations and viewing these results in the context of other similar work. [conc]Chapter summarises what the project accomplished and the implications of its results, commenting on the potential for further work related to the project.

Preparation

There were three main stages to the preparation of the project:

Defining and planning the project. A project of this scale needs clear definition of its goals and a well defined plan designed to achieve these goals. Sections prep-requirements, prep-changes prep-start detail this stage of the preparation.

Learning about the relevant concepts and methods. This was useful as it helped me to make informed decisions about implementation decisions. This required considerable work, as most of the skills and knowledge required to undertake the project are not taught in the Cambridge BA Computer Science course and the parts that are taught are Part II courses. The time scale of the Part II project meant that I had to learn the courses ahead of the lectures. Sections prep-supervised through prep-bow detail this stage of the preparation.

Specifying the details of the implementation. Sections prep-sweng prep-tools detail this stage of the preparation.

Requirements Analysis The primary goals of this data are to:

Construct a database that comprises British texts on the Iraq war

Develop a classifier that can determine the stance of the texts in the database.

I will be using the Hansard (discussed further in prep-changes) as my the corpus from which to construct the database. This allows me to refine the goals above as follows:

[]

Breakdown of the project's core tasks

The tasks in Table are in order of descending priority, due to their dependence on each other.

With any software project, it is necessary to consistently consider both the project's requirements and how these will be evaluated. In prep-bayes I discuss the Naive Bayes Classifier, which I use as a baseline for the classifier and in prep-eval I consider further aspects of evaluation.

Changes from the Initial Proposal In the proposal (see Appendix ), I wrote about using the dataset produced by Robinson, Goddard, Brown and Taylor in which they "evaluated media performance during the 2003 Iraq War". As part of their evaluation, they manually annotated the stance of 4,893 British newspaper articles on the Iraq war. They published the resulting dataset, but it didn't contain the body of the articles - only its headline, author, newspaper and publication date. I consequently investigated resources containing the text of the relevant articles and tried to cross-reference the data from these sources with the manually annotated stance. At the time, many newspapers published different stories online and in print, meaning that I could not rely on these. A few newspapers maintain electronic archives of their printed editions on the internet, however not enough newspapers had such archives. The final resource I looked into was Dow Jones Factiva, a "global news database". Upon inspection, this database contained the vast majority of the articles I needed and it was possible for me to cross-reference the articles in it with the labels annotated by Robinson, P. and Goddard, P. and Brown, R. and Taylor, P.M.. I initially accessed the dataset through the University of Cambridge's subscription. I therefore (falsely) assumed that this subscription would be sufficient for use in my project, however I later discovered that an academic licence did not permit me to use the API or to carry out text-mining. I consequently contacted Dow Jones and was told that the licence I required would cost in excess of 20,000.

After exhausting all other options, I turned my attention to the House of Commons Hansard archives, which contains transcripts of debates between members of Parliament in the Commons Chamber. One of the benefits of this dataset is that the texts can be labelled using MPs' voting records.

Due to the licensing problems I encountered with Dow Jones Factiva, I immediately looked into the licence required to scrape data from the Hansard and found that it is covered by the Open Parliament Licence. Since the Hansard archives are available under this licence, I was permitted to:

"copy, publish, distribute and transmit the information"

"adapt the information"

"exploit the information commercially and non-commercially, for example, by combining it with other information, or by including it in your own product or application".

Starting Point - DEADLINE: 11TH APRIL For the reasons described in prep-changes, the actual starting point for this project differs from what I stated in the proposal (see Appendix ). In intro-related, I discussed previous research that is potentially useful to this project. The project builds on the Hansard and Parliamentary voting records to produce a dataset which combines the two. The project also uses various Python libraries, which are specified in prep-tools-libs.

Introduction to Supervised Learning A supervised learning problem the task of determining the label of a given input. This is split into two phases: Learning and predicting.

In the learning phase, the system receives inputs of feature vectors and their associated labels. A feature vector of length is usually denoted by where

A feature vector contains encodes the information necessary to predict a label. In the context of this project, there is a feature vector for each speech we consider, which contains information about the words in the speech. The label is usually denoted by . The set of values that can take varies depending on the context of the supervised learning problem. For example, in a regression problem, . This project concerns binary classification, since we simplify the problem so that we consider all speeches to be either pro-war or anti-war. Because of this, from now on, we will only consider binary classification problems, that is where .

The supervised learning system creates a function that takes a feature vector as an input and outputs a label. That is

This definition allows us to intuitively view each feature vector as a point in k-dimensional space. We consider each point to be either negative () or positive (). In this analogy, is a function that determines whether a point is negative or positive, depending on where it is in the k-dimensional space. The more points that sees, the better its estimation of whether new unseen points are negative or positive.

The figure below shows a visualisation of our intuition of feature vectors, where . In this diagram, the supervised learning system learns a function to distinguish the '-' and '+' points. Given a new, previously unseen point, this function would be able to estimate whether it is a '-' or a '+'.

Introduction to the Naive Bayes Classifier This is one of the simplest classifiers to understand and implement. It uses the assumption that all features are independent of each other:

We say that the classifier is 'naive' because of this assumption. Although the assumption is very rarely true, the classifier still provides good performance.

In addition to this assumption, the classifier uses Bayes Theorem:

The intuition behind the classifier is that given a set of features , we should assign it to the class that has the highest probability, given the set of features. Using the assumption of conditional independence and Bayes Theorem, we can compute this probability as follows:

Clearly, this shows that we can compute y using:

We can estimate each trivially using the training data. Given that in this project I am only considering binary classifiers, where , we can write this as:

Due to its simplicity and good performance, I will use the naive Bayes classifier as a baseline for my project.

Introduction to Support Vector Machines Support vector machines (SVMs) are widely used, state-of-the-art classifiers which were designed for binary classification (although they have since been modified to work for multi-class classification). Since I am viewing the task of determining the sentiment of speeches on the Iraq war as a binary classification problem, using a SVM is a natural choice.

In contrast to the naive Bayes classifier, the SVM approach to classification is not inherently probabilistic. Instead, they are a form of maximum margin classifier. A maximum margin classifier computes a hyperplane of the form

where is a normal to the hyperplane. and are determined by the maximisation () and is a point on the hyperplane. This hyperplane separates the training data, so that for all positive examples

and for all negative examples

The idea of the maximum margin classifier is that it maximises , the distance between the hyperplane and the closest examples to it. That is, it computes

The figure below illustrates this problem in a 2D space (i.e. where )

To determine whether feature vector should be positively or negatively labelled, we simply need to determine which side of the hyperplane it lies on. This gives us the decision function

where is the feature vector being classified. The support vectors are defined as the training examples that lie closest to the hyperplane. From the equation of the hyperplane (), we see that we have the freedom to scale and by a constant factor without changing the hyperplane itself. We can therefore define this scaling by imposing the following constraint on all training examples for mathematical convenience:

For the support vectors, we then have

& y\_i(w x\_i + b) - 1 = 0

& w x\_i = 1y\_i - b.

& b = 1y\_i - w x\_i.

We now need to compute the width of the margin so we can then form an expression to maximise it. The figure above gives us some intuition as to how we can achieve this. Since is perpendicular to the hyperplane, must be the unit normal to the hyperplane. We can then use a positively labelled support vector, and a negatively labelled support vector, to get an expression for the margin width:

We can now substitute in the result from () to give

Our goal is to maximise the width given by (). For mathematical convenience, we can instead solve the equivalent problem of minimising . This optimisation is subject to the constraints in (). In order to solve this constrained optimisation problem, we must use Lagrange multipliers

where is the number of training examples. This results in the Lagrange function

Our task is now to solve the unconstrained maximisation problem

Using the Karush-Kuhn-Tucker conditions, we can show that for all feature vectors that are not support vectors. This results in fast computation and means that after training, we only need to store the support vectors. Therefore, from now on, we will sum over , the set of indices corresponding to the support vectors.

In order to solve the optimisation problem defined in (), we must find the partial derivative of L with respect to both and , setting the resulting expressions to 0 (since we want to vary and in order to find the maximum L).

Lw & = w - \_i S \_i y\_i x\_i = 0

w & = \_i S \_i y\_i x\_i

Lb & = \_i S \_i y\_i = 0 We can now substitute () into () to obtain a new expression for (and simplify using ()) as follows:

We now need to find the values which maximise L:

I won't go into the details of how to find these values, but this can be done numerically. Further to this, it can be shown that the space of is convex, so we will not find a local maximum. This is a significant advantage of using SVMs over neural networks.

We can then find by substituting into () and using the support vectors and their labels. From this, we can find using () and substituting in along with any support vector and its label. We can substitute our values for and into the initial decision rule to obtain a new decision rule:

Thus far, we have been working under the assumption that our data is linearly separable. In practice, this is very rarely the case and for this project due to the inherent noise in our data (described in ), this assumption is very unlikely to hold.The figure below illustrates a simple example for which the data are not linearly separable.

In order to fix this problem, we can use a transformation, , to transform our feature vectors into a new space in which our data is more easily separable. Applying this transformation to () gives us a new :

We maximise this as before, finding a new from () and then using those values to find . We can then use these values of and along with the transformation to obtain another decision rule:

We are yet to define , but if we consider the contexts in which it is used, we see that it is always in the form . Therefore, rather than define itself, we define a kernel function

From this, we can rewrite () and () as:

The choice of kernel function can have a significant effect on the performance of a SVM. In order to ensure good results, I will train the SVM using different kernel functions, so the classifier learns which kernel function will work best for the data. The three kernel functions I will consider are:

k(x, x') & = x x'&

k(x, x') & = ((x x') + r)^d & R\_> 0 . r R\_0 . d N\_> 0

k(x, x') & = e^-x - x'^2 & R\_> 0 From now on, I will refer to (), () and () as the linear kernel, the polynomial kernel and the radial basis function (rbf) kernel respectively. Using the linear kernel is equivalent to our SVM before we introduced the transformation . This shows that even with kernel functions, our data may not be linearly separable. It can be shown that linear and polynomial kernels do not necessarily transform the data so that into a space for which it is linearly separable, but the rbf kernel can always map the feature vectors to a space where they are linearly separable. This means that for the linear and polynomial kernels, we may not be able to produce a classifier with the given constraints and for the rbf kernel the SVM is very susceptible to overfitting. Both of these problems can be solved by introducing soft-margins to our SVM. This means that we will allow the SVM to incorrectly classify some of the training examples. In order to do this, we introduce a parameter which trades off correct classification of training examples with a greater margin width. A greater margin width results in a smoother function, so means that the SVM is less likely to overfit. The higher the value of , the more training examples the SVM will fit correctly. is a hyperparameter - that is a parameter whose value is fixed before the classifier is trained. All of the hyperparameters for each of the kernels we are considering are shown in Table . The hyperparameter choice significantly effects the performance of the SVM, so we need an algorithm for choosing them. This is discussed further in both cross-validation impl-classifier.

[]

The hyperparameters of various kernels

Introduction to Evaluating Supervised Learning Systems

Train/Test Split In supervised-learning, we saw how a supervised learning system is trained on one set of data (the training set), then this trained model is used to predict the labels of previously unseen data (the testing set). In order to evaluate a system, we require the actual data labels, so we can assess the accuracy of the predictions. Having training examples in the testing set results will not provide a useful measure of the system's performance, as it would unfairly reward overfitting. In order to overcome this, we must split our labelled data before we start developing a model. Using 90 for training and 10 for testing is the most common way to split the labelled data.

Cross Validation It is useful to split the training set into disjoint folds. Doing this means that we can iterate the process of training and evaluating without using the data set aside for testing. This is done by training on of the folds, then evaluating the system on the fold left out. This is repeated times, so each fold is used for evaluation exactly once. Averaging over all the folds each fold gives a reliable evaluation metric. We can use this method to determine the system's hyperparameters and any other settings that need to be determined before training. This is called cross-validation. To do this, we repeat the method described for different combinations of hyperparameters and settings and select the combination whose average evaluation metric is greatest.

Evaluation Metrics Thus far, we have only spoken abstractly about an evaluation metric. There are various options and choice of the measure should be context-specific. The basis of the definitions used in most metrics are defined in the Table .

[]

Defining true positives, true negatives, false positives and false negatives

From this, we can now define the following quantities:

TP & = Number of True Positives

FN & = Number of False Negatives

FP & = Number of False Positives

TN & = Number of True Negatives

Accuracy is the simplest evaluation metric. We define this as:

If we have an unbalanced dataset (which is the case in this project), then accuracy is not a good evaluation metric. To illustrate this, consider the case where 1 of our data is positive and 99 of our data is negative. If we had a classifier that always predicted that an example was negative, its accuracy would be 99. Since accuracy is not a useful measure for unbalanced datasets, such as the dataset I am using in this project, I will not consider it any further.

We now define two further measures:

Precision & = TPTP + FP

Recall & = TPTP + FN

A good evaluation metric for an unbalanced dataset will incorporate some trade-off between precision and recall. Such a metric may give greater weighting to precision for a precision-critical task or greater weighting to recall for a recall-critical task. Since this project is neither precision-critical nor recall-critical, we can use the score as the evaluation metric, since the score is defined as the harmonic mean of precision and recall:

Introduction to the Bag-of-Words Model In mathematics, a bag is a synonym for a multiset - an abstract data type which is like a set, but differs in that it can contain duplicates. In this project, I will represent MPs' speeches using a bag-of-words model, meaning that the representation ignores the order of words. Despite its simplicity, the bag-of-words model is used successfully in a range of sentiment classification applications, as it can effectively capture the discourse of a text.

To illustrate a bag-of-words model using an example, I will use an quote from a House of Commons debate on 18th March 2003:

"The best way to avoid war is to work through the United Nations."

- Bill Tynan (Labour Party)

In a simple bag-of-words implementation, where case and punctuation are ignored, this sentence would be stored as:

\*

the&: 2,

to&: 2,

best&: 1,

way&: 1,

avoid&: 1,

war&: 1,

is&: 1,

work&: 1,

through&: 1,

united&: 1,

nations&: 1.

It is important to note that the order of the elements above is not relevant.

Software Engineering Techniques

Before implementing the first classifier, I couldn't be sure that it would be possible to develop a system to classify MP's speeches due to the lack of other similar work. Due to this, it was important to get to this point as quickly as possible and then change the project requirements at that point if the data couldn't be classified. This strategy lent itself to agile software development, so this is what I used. I considered each of the tasks in Table as my first five sprints (in the same order as the table), later defining further sprints as project extensions.

Throughout development, I used a linter to ensure that I maintained a high standard of code, with consistent documentation and also used revision control. I designed programs to be as modular as possible, which helped with testing and debugging.

Development of machine learning systems requires further good practices to be adopted, to avoid hard-coding rules that result in overfitting. To this end, as soon as I had constructed the database, I split the data into a training set and a testing set. Throughout the implementation, I solely used the training set, carrying out cross-validation across it to test code then using the testing set for evaluation. Further to this, before classifying MPs' speeches, I implemented a classifier for spam emails (see eval-spam) and then adapted this classifier for the purposes of this project.

Choice of Tools

Programming Language

I decided to develop the project in Python for various reasons:

There are many good natural language processing and supervised learning libraries available for Python.

Python is well-suited to agile development.

Python has a lot of community support available.

I have previously used Python for large software projects (working in industry).

I used Python 3.6 since it was the most recent stable version of Python when I started implementation.

Database Due to the structure of the data, it made sense to use a relational database. After considering various options, I opted to use SQLite due to its low set-up overheads (which is useful for agile development), its stability and its compatibility with Python.

Libraries I used BeautifulSoup to parse the Hansard's html and the .ems files from the spam email dataset. Although lxml is a faster parser, BeautifulSoup copes better with 'broken' html. Before implementation, I found various inconsistencies with the Hansard's html so BeautifulSoup was a better choice (especially since there were no significant time constraints on parsing). The Natural Language Toolkit is a widely used library with multiple functions, so I used this for various things. Similarly, I used functions from scikit-learn when implementing the classifier. In order to perform fast mathematical computations I used both NumPy and SciPy. In order to perform fast HTTP requests, I used the Requests library. In addition to the libraries mentioned, I also used different modules from the Python standard library.

Development Environment

I predominantly developed the project using my own laptop running the Windows 10 operating system. I used Visual Studio Code as the source code editor, with Python and PyLint extensions. One advantage of using Visual Studio Code is the easy integration with git, which I used for revision control of both the Python source code and this dissertation's source code. Since I used git for revision control, I used GitHub to backup the source code. I also synced all of my dissertation files to Google Drive and periodically backed them up to an external disk.

Table of Software Used

The versions of the software used in this project

Summary - DEADLINE: 13TH APRIL

In this chapter, I have defined the project and the steps I took before starting its development. These steps involved planning the project and how its success will be measured, thereby allowing smoother implementation and evaluation.

Implementation

System Architecture Through the requirements analysis in the preparation section (prep-requirements), I established the project's core tasks and their dependences (see Table ). From this, I was able to simply create a project timetable, broken down into sprints. Each of the project's core tasks was the milestone reached by a sprint. After completing the essential components of the project, I improved the classifier by implementing a series of extensions. The list below gives each of the sprints I carried out (including the extensions):

Scrape the relevant data from the Hansard.

Wrangle the textual data so it is in a more consistent form.

Collate the data from the transcript with voting record data.

Construct a database of the new dataset I have created.

Parse the spam email dataset.

Develop an SVM classifier to detect spam email.

Develop a naive Bayes classifier to predict the stance of MPs' speeches.

Use the spam email classifier as a basis to develop an SVM classifier to predict the stance of MPs' speeches on the Iraq war.

Extension: Implement stop word removal, stemming, n-grams and number grouping.

Extension: Implement an algorithm to allow the SVM classifiers to learn the best combination of settings and hyperparameters.

Extension: Optimise the SVM classifiers by carrying out singular value decomposition on the feature vectors.

As this project is very data intensive, I defined how data would flow before commencing implementation. Figure shows the movement of data between modules and data sources (including local files). Data moves in the direction of the arrows. This diagram shows the project's dependency on the Hansard, data.parliament.uk and the spam email dataset. I therefore ensured that I had a backed-up copies of the data from these sources at the earliest possible stage in the project, to mitigate any issues if the servers hosting these datasets went down at any point throughout the project.

Data flow in the project.

Deciding upon this data flow, then allowed me to develop a finer project structure. Figure shows the internal dependencies of the modules in the project. An arrow from A to B indicates that A is dependent on B. Note that I designed the system in a way that avoids any cyclic dependencies and maximises code-sharing between modules.

Internal dependencies within the project.

Data Structures The size of the project means that I used a large variety of different data structures. In this section, I discuss the choice and use of a few of these data structures.

NumPy Arrays

I used various parts of the NumPy library throughout the project. For example, I frequently used NumPy arrays in situations where a simple Python list would have sufficed. For this project, NumPy arrays had various advantages over Python lists:

Operations on them are faster - they were specifically designed to with performance in mind, which Python was not.

They are more space efficient - their low-level implementation means that they use contiguous blocks of memory, rather than a series of pointers.

They support matrix operations with no additional work.

The benefits of the first two points were particularly clear given the large amount of data I was processing. Since the features were the largest data structures I used for numerical computations, they made the advantages of using NumPy arrays clearest. Listing gives a snippet of code which demonstrates my use of NumPy arrays.

A function using NumPy array operations to perform singular value decomposition on two sets of given featuresreduce\_features

Sets

The set data structure is built-in to Python and is implemented using a hash table, meaning that lookup operations are performed in constant time. This in turn speeds up the implementation of other set functions, such as union, particularly for data which significantly overlap. For this reason, I used the Python set data type wherever it would improve the efficiency of a function.

Counters

The Counter class is part of the collections module, which is part of the Python standard library, meaning that there is ample support available for its use. The data structure is optimised for counting hashable objects, meaning that it is essentially a multi-set designed to count the number of each element, which makes it perfect for representing a bag of words, making it useful throughout this project. As discussed in section impl-reduction, it was necessary to reduce the size of the features used for classification, which could easily be done in time (where is the number of dimensions on the reduced features) using Counter's mostcommon() method.

Data Acquisition

Scraping Data This sections discusses how I got acquired the relevant data. The difficulty of this sprint of the project was that the transcript data that I required was not available in an API, so I had to scrape the data from the Hansard's inconsistently (and often incorrectly) structured web pages. This meant that I had to develop defensive code that handled lots of corner cases. As discussed in prep-tools-libs, I opted to use BeautifulSoup to parse the .html files.

In order to scrape the House of Commons transcripts was to develop a program that could traverse the archives to find the pages containing relevant debates. In order to do this, I iterated over all dates in the range I was considering(11/09/2001 to 18/03/2003 (inclusive)) and on each of these days, parsed the Hansard's webpage for that day and used this to find links to any relevant debates on that day. Figure is one of many examples of incorrectly structured data in the Hansard - it suggests that all the debates on that day were prayers. Due to the Hansard's deficiencies, I had to exhaustively search through each House of Commons sitting, which significantly increased the running time of this part of the program. Listing gives the high level code used to traverse the Hansard archives. Note that the Hansard's inconsistencies required me to program defensively (by using extensive exception-handling).

A screenshot of incorrect structure in the Hansard.

High-level code used to scrape all the debates from a given day.add\_day

Wrangling Data When parsing the debates and quotes, I faced similar difficulties to parsing the Hansard's pages for each day, meaning that I again had to extensively use exception-handling. To illustrate some of the difficulties of using the Hansard to generate text that is suitable for classification, I have given provided Figure which is a screenshot of a quote in the Hansard, along with Listing , which is the corresponding HTML paragraph tag. I have trimmed the HTML slightly, so it is easier to read. After reading through the HTML for a representative sample of debates in the Hansard, I wrote some functions to extract the relevant text from the Hansard's HTML, including the method given in Listing , which uses a series of regular expressions to make textual replacements.

A screenshot of a quote in the Hansard.

HTML paragraph tag of the quote in Figure .quote

Python code using regular expressions to clean the text from a paragraph tag.get\_paragraph\_text

Labelling Data One of my primary reasons for using the Hansard as the corpus for this project was the fact that I could label the data automatically, using MPs' voting records. This in turn meant that the project could be use supervised learning rather than unsupervised learning or semi-supervised learning. It was relatively trivial to retrieve voting data - I used the House of Commons Divisions API from data.parliament.uk. The main difficulty in labelling the data was matching the voting data with the transcript data. This issue was due to the fact that MPs' names varied greatly over time. For example, some MPs adopted married names during their time as an MP and some had titles that were intermittently used to refer to them. Michael Kerr is a good illustration of the multitude of names used by an individual - he is referred to as 13th Marquess of Lothian in the House of Commons Divisions API, the Earl of Ancram in the Hansard and is also known as Baron Kerr of Monteviot. Michael Kerr is one of 36 MPs whose name differed significantly enough between the two data sets that I couldn't algorithmically match their speeches with their voting record. These MPs were mostly unmatchable, due to maiden names, but I manually entered rules to handle these MPs. I matched the other 619 MPs using the function given in Listing . This method tries to match the given speaker with one of the MPs in the data.parliament.uk dataset, by removing any honorary titles from the speaker's name (since the data.parliament.uk dataset doesn't give these titles), then finding the most similar (Similarity is calculated using the Levenshtein distance, which is defined as the distance is the number of deletions, insertions, or substitutions required to convert one string into another.) name in the voting record dataset. If this name is deemed to be sufficiently similar, the speaker is matched. If it isn't, a similar function is called, which uses different data in the data.parliament.uk dataset to match the speaker. If this function cannot find a good match either, it raises a 'MatchException', which is unhandled by the code in Listing , meaning that matchfullname() also raises this exception. The code that handled these exceptions created a file of any unmatched MPs, which I subsequently manually looked up to find alternative names for the MPs. In order to minimise run-time, I keep a match list, so that once a match is found for a given MP, the same match can be found in constant time, since I used a Python dictionary (which is implemented using a hash table).

Function that matches an MP's name in the Hansard match\_full\_name

Database - DEADLINE: 2ND MAY Since the database would only be used locally (so there would only ever be one copy) and I planned on accessing it serially, I could easily ensure that my database would satisfy the ACID properties (Atomicity, Consistency, Isolation Durability) by using a relational database. As described in prep-database, I opted to use SQLite due to its low overheads - it is referred to as a zero-configuration database. The benefits of this are two-fold; it reduces development time and running time, since there is no separate server process and therefore no overhead from message passing to and from the database. Further to this, the authors of SQLite released the code under a licence that allows anyone to "copy, modify, publish, use, compile, sell, or distribute" SQLite, which is clearly sufficient for my needs in this project.

When designing the database schema, I ensured that the database would be fully normalised (in third normal form), which meant that the data would preserve referential integrity and minimise the potential for data duplication. Figure shows the entity-relationship diagram for the database schema I adopted. Note that due to the limited type system in SQLite, all fields are of type 'TEXT'. This is not a problem, because SQLite has built-in functions to handle data as if they were of different types - for example, the DEBATE and DIVISION entities had DATE attributes, which were represented as text in the database file, I could handle them easily as if they were of a type 'DATE'.

The entity-relationship (ER) diagram of the system's database.

When working with the database, it was useful to be able to manually check the data and therefore the functioning of any program using the database. To do this, I used DB Browser for SQLite, which is a lightweight GUI interface that allowed me to browse the data. This was particularly useful for 'evaluating' the data acquisition, since there is no formal way for me to automatically check the data in the database. This meant that manually sampling the data using DB Browser for SQLite and cross-referencing the data with that in the original data sources was the best feasible way for me to evaluate the data acquisition.

To maintain good software engineering practices, I only accessed the database through the database class (as shown in Figure ). Through doing this and keeping the class's member variables private, I only allowed access to the database through the class's public methods, thereby enforcing encapsulation and information hiding. This meant that I had to implement any database queries I used as functions in the database class. Listing is an example of my implementation of a database query.

Function that performs a query on the database to get the ids of all the MPs who voted 'aye' in a given vote and spoke in a debate whose title contains a given term.get\_aye\_members\_from\_term