Chapter 1

Introduction

This dissertation describes the implementation of the two components of a system to perform sentiment analysis on texts on the Iraq war. The first component constructs a dataset of speeches and votes in the House of Commons. The second is a support vector machine classifier used to determine the stance of a given speech on the Iraq war. The classifier achieves an F1 score of 0.894 on the test data, which is a far better performance than the naïve Bayes classifier, which is used as the project's baseline.

1.1 Problem Formulation

In this project, I look at how best to apply machine learning techniques to sentiment analysis on texts about the Iraq war. To optimise the performance of the sentiment analysis, I created a dataset using the Hansard (which is a collection of transcripts of House of Commons debates) and data from data.parliament.uk. I developed this dataset in a way that made it easy to label an MP's speeches using their voting record, thus giving a labelled corpus (a corpus is a collection of written or spoken material stored on a computer and used to find out how language is used [12]). This meant that I could consider the sentiment analysis as a supervised learning classification problem. One of the simplest implementations of a classifier is the naïve Bayes classifier (described in §2.5), so I use this as a baseline. For reasons outlined in §2.6, I use a support vector machine classifier as the default model. Throughout this dissertation, I discuss how I optimised the classifier while maintaining good software engineering practices.

1.2 Motivation

The Iraq war was an issue that cut across the political spectrum, making 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 [27]. The recent publication of the Iraq Inquiry (commonly known as the Chilcot Inquiry) [7] and the ongoing situation in Iraq and Syria [33] mean that such a project is particularly timely.

The development and publication of the project's dataset will facilitate further NLP

work on House of Commons data, which will hopefully result in increased accountability of MPs.

1.3 Related Work

There have been various NLP studies which have used the Hansard. The 2014 work by Wattam et al. is most similar to my use of the Hansard, since it also develops a new dataset. The dataset they construct is a tagged version of the Hansard, which facilitates faster retrieval of the data [40].

While many sentiment analysis studies have been based on 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' [18] and there have since been countless similar studies. In 2010, Pak et al. proposed that Twitter could be used to determine public opinion [29], 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 [38]. This focus on Twitter is useful, but since 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. 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 previous work.

In 2016, Zhou et al. published a paper on their work in which they performed sentiment analysis of Wikipedia articles on wars by comparing Wikipedia pages in different languages [41]. Although this study used a different methodology to my project (since it compared texts in different languages), it provides evidence that sentiment analysis can work on corpora concerning war.

There have been a some papers detailing sentiment analysis on transcripts of US Congress debates [4, 17]. The results of these studies indicate that determining an MP's stance on the Iraq war from their Parliamentary speeches may be possible. However, these studies determine a politician's political party, which is likely to be more clear-cut than their stance on a particular issue.

1.4 Overview of the Project

Chapter 2 defines the project, then outlines the relevant models and algorithms before discussing implementation decisions. Chapter 3 details the system's development, while chapter 4 assesses the project's success. Chapter 5 summarises the project's accomplishments and implications, commenting on potential further work related to the project.

Chapter 2

Preparation

There were three main stages to the project's preparation:

1. Defining and planning the project. A project of this scale needs a clear definition of its goals and a well-defined plan designed to achieve these goals. Sections §2.1, §2.2 & §2.3 detail this stage.

2. 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 timescale of the Part II project meant that I had to learn the courses ahead of the lectures. Sections §2.4 through & §2.8 detail this stage.

3. Specifying the details of the implementation. Sections §2.9 §2.10 detail this stage.

2.1 Requirements Analysis

The primary goals of this project 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 [28] (discussed further in §2.2) as the corpus to construct the database from. This allows me to refine the goals as in table 2.1. The tasks in table 2.1 are in order of descending priority, due to their dependence on each other.

With any software project, it is necessary to consistently consider the project's requirements and how these will be evaluated. In §2.5 I discuss the Naïve Bayes Classifier, which I use as a baseline for the classifier and in §2.7 I consider further aspects of evaluation.

Task	Section
Scrape the relevant data from the Hansard	§3.3.1
Wrangle the textual data so it is in a more consistent form	§3.3.2
Collate the data from the transcripts with voting data	§3.3.3
Construct a database of the dataset I have created	§3.3.4
Develop a system that can predict the stance of a text on the Iraq war	§3.4

Table 2.1: Breakdown of the project's core tasks

2.2 Changes from the Initial Proposal

In the proposal (see Appendix A), I wrote about using the dataset produced by Robinson et al. in which they "evaluated media performance during the 2003 Iraq War" [32]. As part of their evaluation, they manually annotated the stance of 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" [19]. Upon inspection, this database contained the 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 et al. I initially accessed the dataset through the University'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 didn't 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 Hansard archives, which contain transcripts of House of Commons' debates [28]. 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 [19], I immediately looked into the licence required to scrape data from the Hansard and found that it is covered by the Open Parliament Licence [16]. 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 applica-

tion".

2.3 Starting Point

For the reasons described in §2.2, the actual starting point for this project differs from what I stated in the proposal (see Appendix A). In §1.3, I discussed research that is relevant to this project. The project builds on the Hansard [28] and Parliamentary voting records to produce a dataset which combines the two. The project also uses various Python libraries, which are specified in §2.10.3.

2.4 Introduction to Supervised Learning

This project is a supervised learning project since it uses labelled speeches to determine the class (i.e. pro-war or anti-war) of previously unseen texts. Supervised learning 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 k is usually denoted by $\mathbf{x}$ where

$$\mathbf{x} = (x_1, x_2, ..., x_k) \quad \forall i \in \mathbb{Z}^+ . \forall x_i \in \mathbb{R}. \tag{2.1}$$

A feature vector 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 y. The set of values that y can take varies depending on the context of the supervised learning problem. For example, in a regression problem, $y \in \mathbb{R}$. 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 $y \in \{-1, +1\}$.

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

$$h(\mathbf{x}) = y. \tag{2.2}$$

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 ($y = -1$) or positive ($y = +1$). In this analogy, h 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 h sees, the better its estimation of whether new unseen points are negative or positive.

Figure 2.1 shows a visualisation of our intuition of feature vectors, where $k = 2$. 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 '+'.

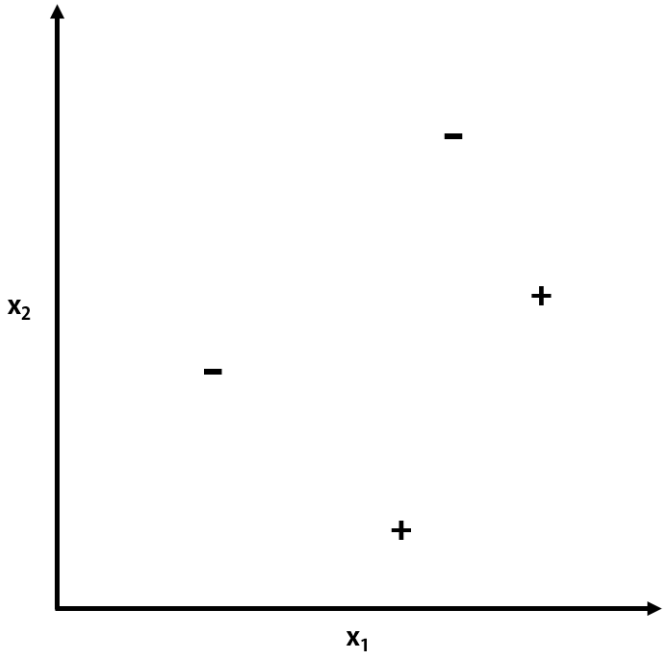

Figure 2.1: A simple 2D plot of four labelled features.

2.5 Introduction to the Naïve 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:

$$p(x_i|x_j) = p(x_i) \quad \forall i, j \in \mathbb{Z}^+. \tag{2.3}$$

The classifier is 'naïve', since this assumption is rarely true. Despite this, the classifier still achieves good performance [25].

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

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.4}$$

The intuition behind the classifier is that given a set of features $\mathbf{x}$, 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:

$$\begin{aligned}
p(C = y|\mathbf{x}) &= \frac{p(\mathbf{x}|C = y)p(C = y)}{p(\mathbf{x})} \\
&= \frac{p(x_1, \ldots, x_k|C = y)p(C = y)}{p(\mathbf{x})} \\
&= \frac{p(x_1, \ldots, x_k, C = y)}{p(\mathbf{x})} \\
&= \frac{p(x_1|x_2, \ldots, x_k, C = y)p(x_2, \ldots, x_k, C = y)}{p(\mathbf{x})} \\
&\vdots \\
&= \frac{p(x_1|x_2, \ldots, x_k, C = y)p(x_2|x_3 \ldots, x_k, C = y) \cdots p(x_k|C = y)}{p(\mathbf{x})} \\
&= \frac{p(x_1|C = y)p(x_2|C = y) \cdots p(x_k|C = y)}{p(\mathbf{x})} \\
&= \frac{\prod_{i=1}^{k} p(x_i|C = y)}{p(\mathbf{x})}.
\end{aligned} \tag{2.5}$$

This shows that we can compute y using:

$$\begin{aligned}
y &= \underset{y}{\operatorname{argmax}} \left(\frac{\prod_{i=1}^{k} p(x_i|C = y)}{p(\mathbf{x})} \right) \\
&= \underset{y}{\operatorname{argmax}} \prod_{i=1}^{k} p(x_i|C = y).
\end{aligned} \tag{2.6}$$

We can estimate each $p(x_i|C = y)$ trivially using the training data. Given that in this project I am only considering binary classifiers, where $y \in -1, +1$, we can write this as:

$$\max \left(\prod_{i=1}^{k} p(x_i|C = -1), \prod_{i=1}^{k} p(x_i|C = +1) \right). \tag{2.7}$$

Due to its simplicity and good performance, I use the naïve Bayes classifier as the baseline for this project.

2.6 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) [6]. Since I'm viewing the task of determining the sentiment of speeches on the Iraq war as a binary classification problem, using an SVM is a natural choice.

In contrast to the naïve Bayes classifier, the SVM approach to classification is not

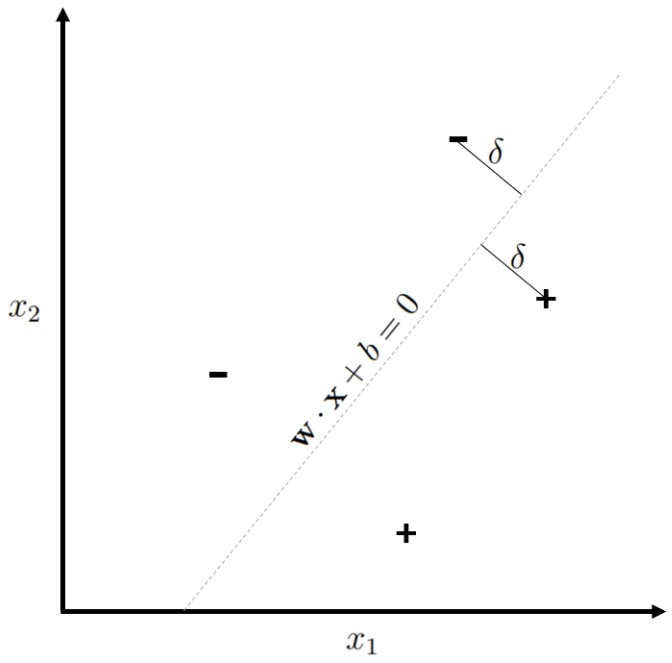

Figure 2.2: A simple 2D plot of four labelled features and a hyperplane (which is a line in two dimensions) distinguishing the positive and negative points.

probabilistic - SVMs are a form of maximum margin classifier. A maximum margin classifier computes a hyperplane of the form

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.8}$$

where $\mathbf{w}$ is a normal to the hyperplane. $\mathbf{w}$ and b are determined by the maximisation (2.11) and $\mathbf{x}$ is a point on the hyperplane. This hyperplane separates the training data so that for all positive examples

$$\mathbf{w} \cdot \mathbf{x} + b \geq 0 \tag{2.9}$$

and for all negative examples

$$\mathbf{w} \cdot \mathbf{x} + b < 0. \tag{2.10}$$

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

$$\underset{\mathbf{w},b}{\text{argmax}}(min(\delta)). \tag{2.11}$$

Figure 2.2 illustrates this problem in a 2D space.

To determine how the feature vector $\mathbf{x}_i$ should be labelled, we simply need to determine which side of the hyperplane it lies on. This gives us the decision function

$$y_i = \begin{cases} +1, & \mathbf{w} \cdot \mathbf{u} + b \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.12}$$

where $\mathbf{u}$ 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, (2.8), we see that we have the freedom to scale $\mathbf{w}$ and b 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:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0. \tag{2.13}$$

For the support vectors, we then have

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0 \tag{2.14}$$

$$\implies \mathbf{w} \cdot \mathbf{x}_i = \frac{1}{y_i} - b \tag{2.15}$$

$$\implies b = \frac{1}{y_i} - \mathbf{w} \cdot \mathbf{x}_i. \tag{2.16}$$

We now need to compute the width of the margin so we can form an expression to maximise it. Figure 2.2 gives us some intuition as to how we can achieve this. Since $\mathbf{w}$ is perpendicular to the hyperplane, $\frac{\mathbf{w}}{||\mathbf{w}||}$ must be a unit normal to the hyperplane. We can then use a positively labelled support vector, $\mathbf{x}_+$ and a negatively labelled support vector, $\mathbf{x}_-$ to get an expression for the margin width:

$$\begin{aligned} \text{Margin width} &= \frac{\mathbf{w}}{||\mathbf{w}||} \cdot (\mathbf{x}_+ - \mathbf{x}_-) \\ &= \frac{\mathbf{w} \cdot \mathbf{x}_+ - \mathbf{w} \cdot \mathbf{x}_-}{||\mathbf{w}||}. \end{aligned} \tag{2.17}$$

We can now substitute in the result from (2.15) to give

$$\begin{aligned} \text{Margin width} &= \frac{\left(\frac{1}{y_+} - b\right) - \left(\frac{1}{y_-} - b\right)}{||\mathbf{w}||} \\ &= \frac{\frac{1}{y_+} - \frac{1}{y_-}}{||\mathbf{w}||} \\ &= \frac{1 + 1}{||\mathbf{w}||} \\ &= \frac{2}{||\mathbf{w}||} \end{aligned} \tag{2.18}$$

Our goal is to maximise the width given by (2.18). For mathematical convenience, we can instead solve the equivalent problem of minimising $\frac{1}{2}||\mathbf{w}||^2$. This optimisation is subject to the constraints in (2.14). In order to solve this constrained optimisation problem, we can use Lagrange multipliers

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \tag{2.19}$$

where n is the number of training examples. This results in the Lagrangian

$$L = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1). \tag{2.20}$$

Our task is now to solve the unconstrained maximisation problem

$$(\mathbf{w}_{opt}, b_{opt}) = \underset{\mathbf{w},b}{\operatorname{argmax}}(L) \tag{2.21}$$

Using the Karush-Kuhn-Tucker conditions, we can show that $\alpha_i = 0$ for all feature vectors that are not support vectors [6]. 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 $\mathcal{S}$, the set of indices corresponding to the support vectors.

In order to solve the optimisation problem defined in (2.21), we must find the partial derivative of L with respect to both $\mathbf{w}$ and b, setting the resulting expressions to 0 (since we want to vary $\mathbf{w}$ and b in order to find the maximum L).

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i\in\mathcal{S}} \alpha_i y_i \mathbf{x}_i = 0 \tag{2.22}$$

$$\implies \mathbf{w} = \sum_{i\in\mathcal{S}} \alpha_i y_i \mathbf{x}_i \tag{2.23}$$

$$\frac{\partial L}{\partial b} = \sum_{i\in\mathcal{S}} \alpha_i y_i = 0 \tag{2.24}$$

We can now substitute (2.23) into (2.20) to obtain a new expression for L (and simplify using (2.24)) as follows:

$$\begin{aligned} L &= \frac{1}{2}(\sum_{i\in\mathcal{S}} \alpha_i y_i \mathbf{x}_i)(\sum_{j\in\mathcal{S}} \alpha_j y_j \mathbf{x}_j) - \sum_{i\in\mathcal{S}} \alpha_i y_i (\sum_{j\in\mathcal{S}} \alpha_j y_j \mathbf{x}_j) \cdot \mathbf{x}_i - b\sum_{i\in\mathcal{S}} \alpha_i y_i + \sum_{i\in\mathcal{S}} \alpha_i \\ &= \frac{1}{2}(\sum_{i\in\mathcal{S}} \alpha_i y_i \mathbf{x}_i)(\sum_{j\in\mathcal{S}} \alpha_j y_j \mathbf{x}_j) - (\sum_{i\in\mathcal{S}} \alpha_i y_i \mathbf{x}_i)(\sum_{j\in\mathcal{S}} \alpha_j y_j \mathbf{x}_j) + \sum_{i\in\mathcal{S}} \alpha_i \\ &= \sum_{i\in\mathcal{S}} \alpha_i - \frac{1}{2}(\sum_{i\in\mathcal{S}} \alpha_i y_i \mathbf{x}_i)(\sum_{j\in\mathcal{S}} \alpha_j y_j \mathbf{x}_j) \\ &= \sum_{i\in\mathcal{S}} \alpha_i - \frac{1}{2}\sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{S}} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j). \end{aligned} \tag{2.25}$$

We now need to find the values $\boldsymbol{\alpha}$ which maximise L:

$$\boldsymbol{\alpha}_{opt} = \underset{\boldsymbol{\alpha}}{\operatorname{argmax}}(L). \tag{2.26}$$

We can then find these values numerically. Further to this, it can be shown that the space of L 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 $\mathbf{w}_{opt}$ by substituting $\boldsymbol{\alpha}_{opt}$ into (2.23) and using the support vectors and their labels. From this, we can find b_{opt} using (2.16) and substituting in $\mathbf{w}_{opt}$ along with any support vector and its label. Substituting our values for $\boldsymbol{\alpha}_{opt}$ and b_{opt} into the initial decision rule we obtain a new decision rule:

$$y_i = \begin{cases} +1, & \sum_{i\in\mathcal{S}} y_i (\boldsymbol{\alpha}_{opt})_i (\mathbf{x}_i \cdot \mathbf{u}) + b \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{2.27}$$

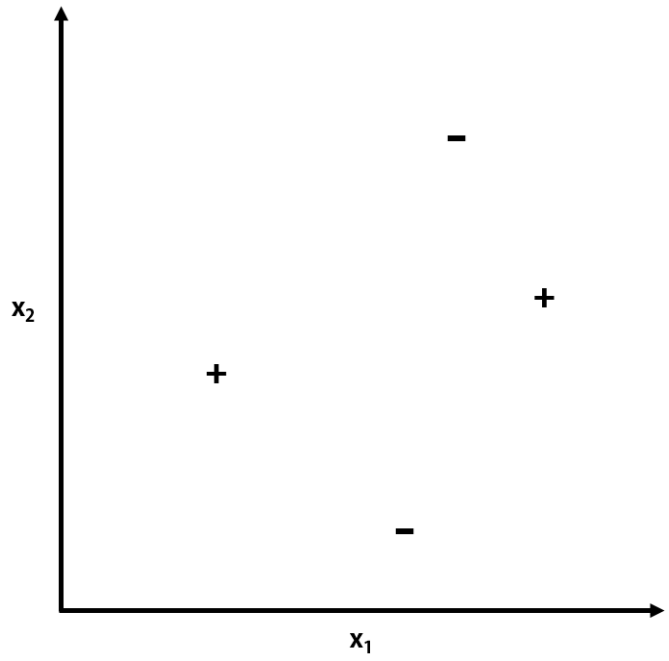

Figure 2.3: A simple 2D plot of four linearly inseparable labelled features.

Thus far, we have been working under the assumption that our data is linearly separable. In practice, this is rarely the case and for this project due to the inherent noise in our data (described in §3.3.3), this assumption is very unlikely to hold. Figure 2.3 illustrates a simple example for which the data are not linearly separable.

In order to fix this problem, we can use a transformation, ϕ, to map our feature vectors into a space in which our data is more easily separable. Applying this transformation to (2.25) gives us a new L:

$$L = \sum_{i \in \mathcal{S}} \alpha_i - \frac{1}{2} \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} \alpha_i \alpha_j y_i y_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)). \tag{2.28}$$

We maximise this as before, finding a new $\boldsymbol{\alpha}_{opt}$ from (2.26) and then using those values to find b_{opt}. We can then use these values of $\boldsymbol{\alpha}_{opt}$ and b_{opt} along with the transformation ϕ to obtain another decision rule:

$$y_i = \begin{cases} +1, & \sum_{i \in \mathcal{S}} y_i (\boldsymbol{\alpha}_{opt})_i (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{u})) + b \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{2.29}$$

We are yet to define ϕ, but if we consider the contexts in which it is used, we see that it is always in the form $\phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$. Therefore, rather than define ϕ itself, we define a kernel function

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}'). \tag{2.30}$$

From this, we can rewrite (2.28) and (2.29) as:

$$L = \sum_{i \in \mathcal{S}} \alpha_i - \frac{1}{2} \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.31}$$

Kernel	Hyperparameters
Linear	C
Polynomial	C, γ, r, d
Radial basis function	C, γ

Table 2.2: The hyperparameters of various kernels

$$y_i = \begin{cases} +1, & \sum_{i \in \mathcal{S}} y_i (\boldsymbol{\alpha}_{opt})_i k(\mathbf{x}_i, \mathbf{u}) + b \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{2.32}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}' \tag{2.33}$$

$$k(\mathbf{x}, \mathbf{x}') = (\gamma(\mathbf{x} \cdot \mathbf{x}') + r)^d \qquad \gamma \in \mathbb{R}_{>0}.r \in \mathbb{R}_{\geq 0}.d \in \mathbb{N}_{>0} \tag{2.34}$$

$$k(\mathbf{x}, \mathbf{x}') = e^{-\gamma ||\mathbf{x} - \mathbf{x}'||^2} \qquad \gamma \in \mathbb{R}_{>0} \tag{2.35}$$

From now on, I will refer to (2.33), (2.34) and (2.35) 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 ϕ. This shows that even with kernel functions, our data may not be linearly separable. It can also be shown that the polynomial kernel does not necessarily transform the data so that into a space in 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 C 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 C, the more training examples the SVM will fit correctly. C 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 given in table 2.6. The hyperparameter choice significantly effects the performance of the SVM, so we need an algorithm for choosing them. This is discussed further in both §2.7.3 & §3.4.

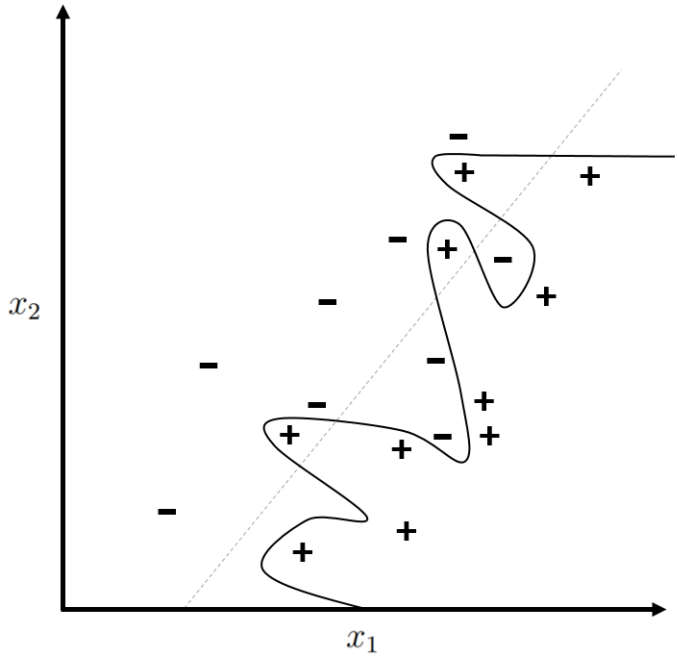

Figure 2.4: An example of the an overfitting function.

2.7 Introduction to Evaluating Supervised Learning Systems

2.7.1 Train/Test Split

In §2.4, 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 predictions. If we have training examples in the testing set results, we will not obtain a reliable 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.

2.7.2 Overfitting

Overfitting is a common problem in supervised learning problems. It occurs when a classifier learns to get a very high performance on the training set but does not generalise well to unseen data, meaning that the classifier achieves poor results on the test data. Figure 2.4 illustrates a simple example of a (two-dimensional) hyperplane that overfits the given data.

		Prediction	
		Positive	**Negative**
Actual Value	**Positive**	True Positive	False Negative
	Negative	False Positive	True Negative

Table 2.3: Defining true positives, true negatives, false positives and false negatives

2.7.3 Cross-Validation

Cross-validation is a way to mitigate overfitting. To use this scheme, we split the training set into k disjoint folds. Doing this means that we can iterate the process of training and evaluate without using the testing set. This is done by training on $k-1$ of the folds, then evaluating the system on the left out fold. This is repeated k times, so each fold is used for evaluation once. Averaging over all the folds 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. To do this, we repeat the method described for different combinations of hyperparameters and settings, selecting the combination whose average evaluation metric is greatest. In this project, I use stratified cross-validation - for this scheme, when splitting the data into folds, I ensure that each fold has a roughly even split of positive and negative examples.

2.7.4 Evaluation Metrics

Thus far, we have only spoken abstractly about evaluation metrics. There are various options and choice of the measure should be context-specific. The bases of the definitions used in most metrics are defined in the table 2.3.

From this, we can now define the following quantities:

$$TP = \text{Number of True Positives} \tag{2.36}$$

$$FN = \text{Number of False Negatives} \tag{2.37}$$

$$FP = \text{Number of False Positives} \tag{2.38}$$

$$TN = \text{Number of True Negatives} \tag{2.39}$$

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

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \tag{2.40}$$

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%.

We now define two further measures:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.41}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.42}$$

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, I use the F1 score as the evaluation metric, as it is defined as the harmonic mean of precision and recall:

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \tag{2.43}$$

2.8 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 [20].

To illustrate a bag-of-words model using an example, I will use a 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:

$$\begin{aligned}
\{&\text{the} : 2,\\
&\text{to} : 2,\\
&\text{best} : 1,\\
&\text{way} : 1,\\
&\text{avoid} : 1,\\
&\text{war} : 1,\\
&\text{is} : 1,\\
&\text{work} : 1,\\
&\text{through} : 1,\\
&\text{united} : 1,\\
&\text{nations} : 1\}.
\end{aligned}$$

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

2.9 Software Engineering Techniques

Before implementing the 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 similar work. Because of this, it was important to develop the first classifier 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 [1], so this is what I used. I used each of the tasks in table 2.1 to establish the project's first sprints, later defining further sprints as extensions.

Throughout development, I used a linter to maintain a high standard of code, with consistent documentation. I also used revision control and 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 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 §4.3) and then adapted this classifier for the purposes of this project, which ensured that the classifier generalised well.

2.10 Choice of Tools

2.10.1 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.

2.10.2 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. I justify this decision further in §3.3.4.

2.10.3 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 [5], 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 many NLP functions, so I used this for various things. I used functions from scikit-learn when implementing the classifier. 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.

2.10.4 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 LaTeX 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.

2.10.5 Table of Software Used

Software	**Version**
BeautifulSoup	4.6.0
git	2.8.1.windows.1
Matplotlib	2.1.2
Natural Language Toolkit	3.2.5
NumPy	1.13.3
Pandas	0.22.0
PyLint	1.7.4
Python	3.6.3
Requests	2.18.4
Scikit-Learn	0.0
SciPy	1.0.0
SQLite	3.10.1
Visual Studio Code	1.20.1
Windows 10	Home

Table 2.4: The versions of the software used in this project

2.11 Summary

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.

Chapter 3

Implementation

This chapter discusses the implementation of the project's components. §3.1 provides an overview of the system's implementation. §3.2 talks about some of the design decisions related to specific data structures. §3.3 gives details about compiling the project's dataset and §3.4 discusses the implementation of the classifier.

3.1 System Architecture

Through the requirements analysis in §2.1, I established the project's core tasks and their dependencies (see table 2.1). From this, I was able to create a project timetable, broken down into sprints. 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:

1. Scraped the relevant data from the Hansard.
2. Wrangled the textual data into a more consistent form.
3. Collated the data from the transcripts with voting data.
4. Constructed a database of the dataset I have created.
5. Parsed the spam email dataset.
6. Developed an SVM classifier to detect spam email.
7. Developed a naïve Bayes classifier to predict the stance of MPs' speeches.
8. Used the spam email classifier as a basis to develop an SVM classifier to predict the stance of MPs' speeches on the Iraq war.
9. Extension: Implemented stopword removal, stemming, n-grams and number-grouping.
10. Extension: Implemented an algorithm to allow the SVM classifiers to learn the best combination of settings and hyperparameters.

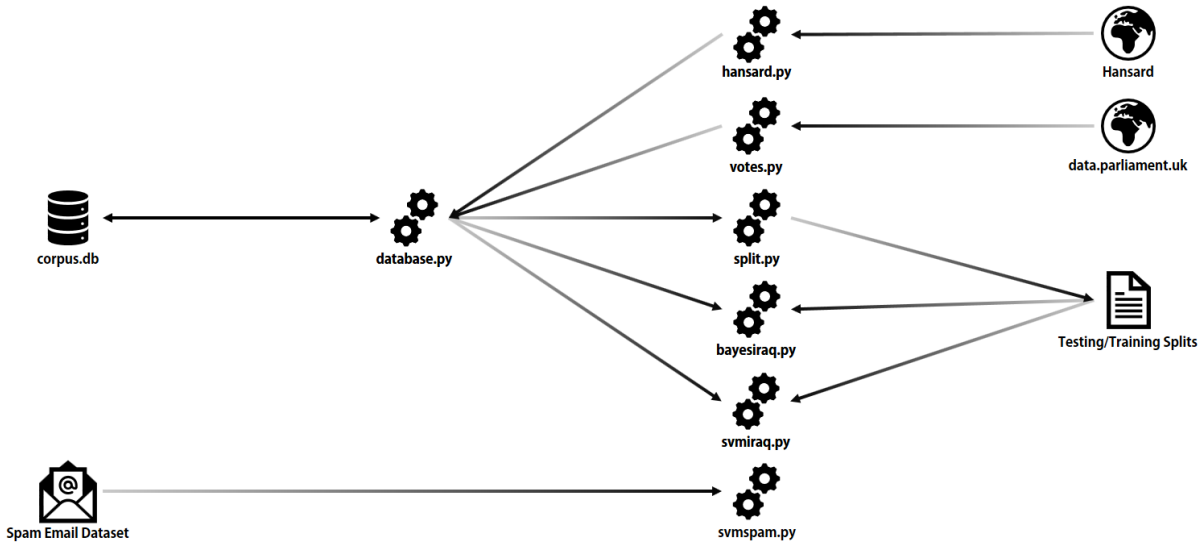

Figure 3.1: Data flow in the project.

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

In addition to these sprints, I also developed an entailment system, using the questions associated with Parliamentary votes, along with the features I use for the sentiment analysis. Since this sprint was tangential to the project and didn't produce very good results, I do not mention it further in this dissertation.

As this project is very data intensive, I defined how data would flow before commencing implementation. Figure 3.1 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 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 during the project.

Deciding upon this data flow, allowed me to develop a finer project structure. Figure 3.2 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 cyclic dependencies and maximises code-sharing between modules.

3.2 Data Structures

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

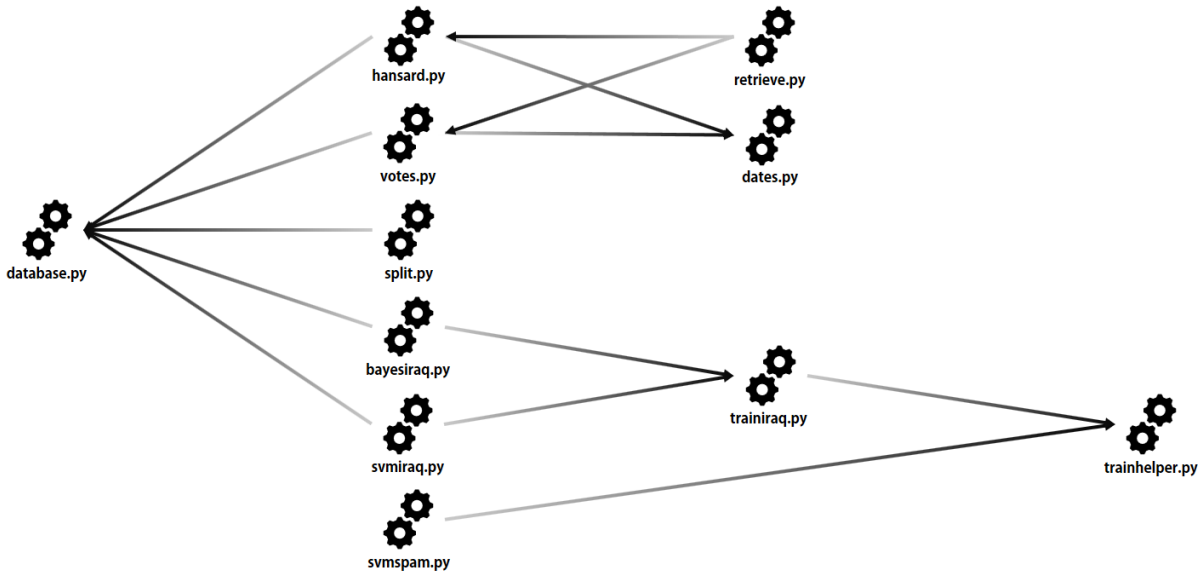

Figure 3.2: Internal dependencies within the project.

3.2.1 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 with performance in mind, which Python was not [39].
- They are more space efficient - their low-level implementation means that they use contiguous blocks of memory, rather than a series of pointers [11].
- They support matrix operations with no additional work.

These benefits 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.

3.2.2 Sets

The set data structure is built-in to Python and is implemented using a hash table [15], 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.

3.2.3 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 [14] - it is essentially a multi-set designed to count the number of each element. This makes it perfect for representing a bag-of-words, so it was useful in this project. As discussed in §3.4.4.5, it was necessary to reduce the size of the features used for classification, which could easily be done in $\mathcal{O}(k)$ time (where k is the number of dimensions on the reduced features) using Counter's `most_common()` method.

3.2.4 Sparse Matrices

The nature of bag-of-words features means that they are primarily populated by zeros. This means that although the dimensions of the feature matrices (matrices where each row is a set of features for a particular example) are very large, they can be compressed into 'sparse matrices', making them more space efficient. Matrix multiplications are also sped up since there are algorithms specifically designed for sparse matrices. Sparse matrices were most useful in dimensionality reduction, as they dramatically decreased the time taken for singular value decomposition. Listing 3.8 gives the code for this feature reduction using sparse matrices.

3.3 Data Acquisition

3.3.1 Scraping Data

This section discusses how I acquired the relevant data. The difficulty of this sprint 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 webpages. This meant that I had to develop defensive code to handle many corner cases. As discussed in §2.10.3, I opted to use BeautifulSoup to parse the `.html` files.

Initially, I developed a program that could traverse the Hansard to find the pages containing relevant debates. In order to do this, I iterated over the dates in the range I was considering[1], parsing the Hansard's webpage for each of these days to find links to relevant debates on those days. Figure 3.3 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 increased the running time of this part of the program. Listing 3.1 gives the high-level code used to traverse the Hansard archives. The Hansard's inconsistencies required me to program defensively, by using extensive exception-handling.

[1] 11/09/2001 to 18/03/2003 (inclusive)

Figure 3.3: A screenshot of incorrect structure in the Hansard.

```
def add_day(corpus, day, member_lists):
    '''Gets the speeches for a given day'''
    date_string = day.strftime('%Y/%b/%d').lower()
    url = 'http://hansard.millbanksystems.com/sittings/{}.js'.format(date_string)
    res = requests.get(url)

    try:
        obj = json.loads(res.text)
        try:
            sections = obj[0]['house_of_commons_sitting']['top_level_sections']
            for section in sections:
                try:
                    sec = section['section']
                    add_debate(corpus, 'http://hansard.millbanksystems.com/commons/{}/{}'
                               .format(date_string, sec['slug']), day, sec['title'],
                               member_lists)
                except KeyError:
                    pass
        except KeyError:
            pass

    except ValueError:
        print('No data for {}'.format(date_string))
```

Listing 3.1: High-level code used to scrape all the debates from a given day.

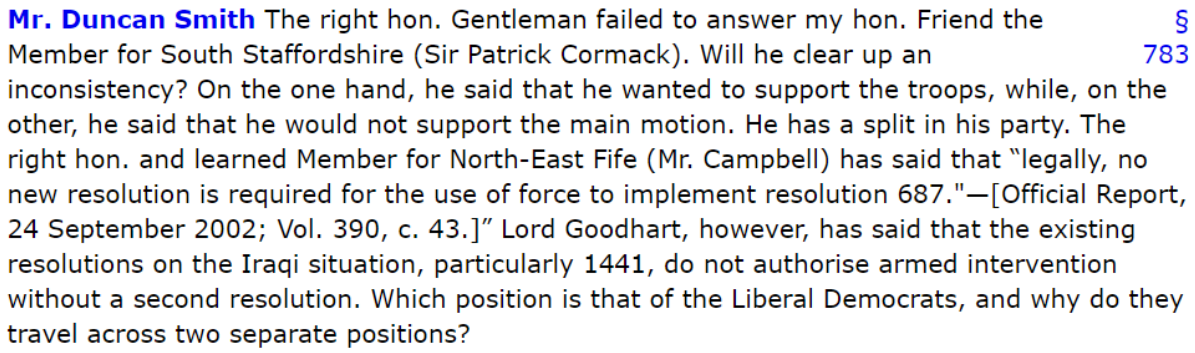

Figure 3.4: A screenshot of a quote in the Hansard.

```
<p class='first-para'>
    The right hon. Gentleman failed to answer my hon. Friend the Member for South
    <a class='permalink column-permalink' id='column_783'
       title='Col. 783 &mdash; HC Deb 18 March 2003 vol 401 c783'
       name='column_783' href='iraq#column_783' rel='bookmark'>783</a>
    Staffordshire (Sir Patrick Cormack). Will he clear up an...
    <q>legally, no new resolution is required for the use of force to implement
       resolution 687."&#x2014;[<span class="italic">Official Report</span>
       , 24 September 2002; Vol. 390, c. 43.]</q>
    Lord Goodhart, however, has said...
</p>
```

Listing 3.2: HTML paragraph tag of the quote in Figure 3.4.

3.3.2 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 figure 3.4 which is a screenshot of a quote in the Hansard, along with listing 3.2, 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 3.3, which uses a series of regular expressions to make textual replacements.

```
def get_paragraph_text(paragraph):
    '''Converts a paragraph tag to plain text'''
    paragraph = re.sub(r'<p.*?>', '', paragraph)
    paragraph = re.sub(r'</p.*?>', '', paragraph)
    paragraph = re.sub(r'<a.*?>.*?</a>', '', paragraph)
    paragraph = re.sub(r'<span.*?>.*?</span>', '', paragraph)
    paragraph = re.sub(r'<q.*?>.*?</q>', '\'', paragraph)
    return paragraph
```

Listing 3.3: Python code using regular expressions to clean the text from a paragraph tag.

3.3.3 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 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. One downside to labelling the data in this way is that it introduces noise into the data, as MPs do not vote consistently with their own views due to party whips [3], resulting in some mislabelled data. This noise can be accounted for by good selection of hyperparameters.

The main difficulty in labelling the data was matching the voting data with the transcript data, due to the fact that MPs' names varied greatly over time. For example, some MPs adopted married names during their time in Parliament and some had titles that were intermittently used to refer to them. Michael Kerr provides a good illustration of the multitude of names of an individual - he is referred to as 13th Marquess of Lothian in data.parliament.uk, the Earl of Ancram in the Hansard and is also known as Baron Kerr of Monteviot. Kerr was one of 36 MPs whose name differed 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 them. I matched the other 619 MPs using the function given in listing 3.4, which 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[2] name in the voting record dataset. If this name is sufficiently similar, the speaker is matched. If it is not, 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 3.4, meaning that `match_full_name()` also raises this exception. The code that handles these exceptions creates a file of any unmatched MPs. I manually looked up these MPs to find alternative names for them. In order to minimise run-time, I kept a match list, so that once a match was found for a given MP, the same match could be found in constant time, since I used a Python dictionary (which is implemented using a hash table [13]).

3.3.4 Database

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 ensure that my database would satisfy the ACID properties (Atomicity, Consistency, Isolation & Durability) by using a relational database. As described in §2.10.2, I opted to use SQLite due to its low overheads - it is referred to as a zero-configuration database [35]. The benefits of this are two-fold; it reduces development time and running time since there is no separate server process and

[2]Similarity is calculated using the Levenshtein distance, which is defined as the number of deletions, insertions, or substitutions required to convert one string into another.

```
def match_full_name(speaker, member_lists):
    ''' Returns the member_id for the given speaker where a match exists '''
    if speaker in member_lists['match_list']:
        mp_id = member_lists['match_list'][speaker]
    else:
        max_similarity = 0
        no_titles = remove_titles(speaker)

        for name in member_lists['name_list']:
            similarity = Levenshtein.ratio(remove_titles(name[1]), no_titles)
            if similarity > max_similarity:
                max_similarity = similarity
                best_match = name

        if max_similarity > 0.85:
            member_lists['name_list'].remove(best_match)
            mp_id = best_match[0]
        else:
            mp_id = match_first_and_family_name(no_titles, member_lists['name_list'],
                                                speaker)

        member_lists['match_list'][speaker] = mp_id

    return mp_id
```

Listing 3.4: Function that matches an MP's name in the Hansard

therefore no overhead from message passing to and from the database [36]. 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 [37], which is sufficient for this project.

When designing the database schema, I ensured that the database would be in third normal form, meaning that the data would preserve referential integrity and minimise the potential for data duplication. Figure 3.5 shows the entity-relationship diagram for the database schema I adopted. Due to the limited type system in SQLite [34], 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 but I could handle them easily as if they were of a type 'DATE'.

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. Manually sampling the data using DB Browser for SQLite and cross-referencing it with the original data sources was one of the ways which I evaluated the data acquisition.

To maintain good software engineering practices, I only accessed the database through the database class (as shown in Figure 3.1). Through doing this and keeping the class's member variables private, I only allowed database accesses through the class's

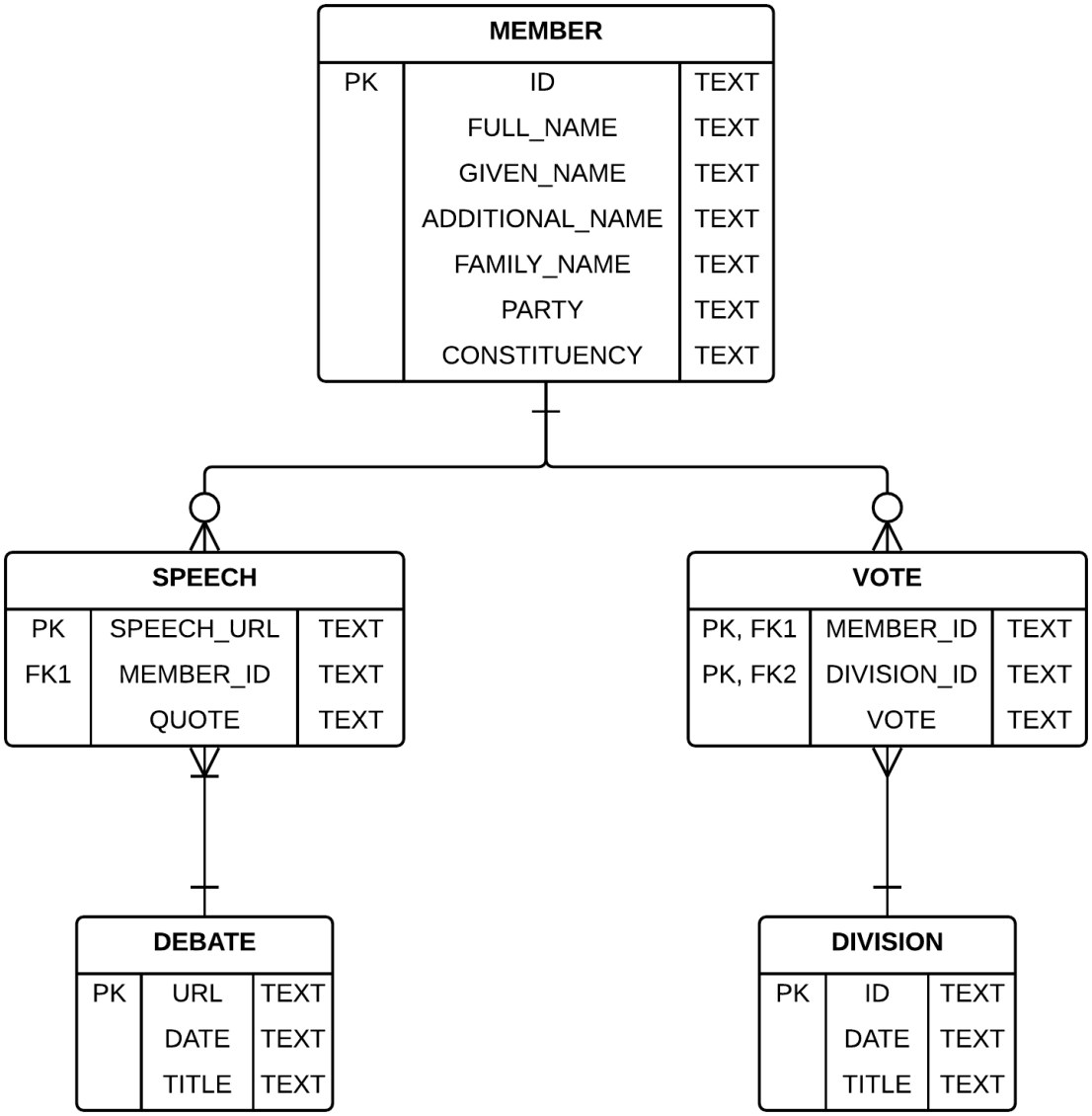

Figure 3.5: The entity-relationship (ER) diagram of the system's database.

```
def get_aye_members_from_term(self, term, division_id):
    ''' Returns a list of member ids corresponding to members who voted aye
        in a given division and spoke in a debate matching a given term '''

    self.__curs.execute('''SELECT DISTINCT MEMBER_ID FROM SPEECH
                           WHERE DEBATE_URL IN (SELECT URL FROM DEBATE
                                                WHERE TITLE LIKE ? COLLATE NOCASE)
                           AND MEMBER_ID IN (SELECT ID FROM MEMBER INNER JOIN VOTE ON
                                             VOTE.MEMBER_ID = MEMBER.ID
                                             WHERE VOTE.DIVISION_ID=? AND
                                             (VOTE.VOTE='AyeVote')) ''',
                           ('%{}%'.format(term), division_id))

    rows = self.__curs.fetchall()

    return [row[0] for row in rows]
```

Listing 3.5: 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.

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. Since database accesses are expensive, I limited them as much as possible, in part by querying the database and storing results at the beginning of the program's execution, then accessing values from variables when they were needed. Listing 3.5 is an example of my implementation of a database query.

3.4 Classifier

3.4.1 Feature Generation

Once the database was formed, the first step to develop the classifier was determining which debates to use for the classifier. To do this, I wrote a function to query the database for any debates with titles containing one or more of a given set of terms. I then parsed all the speeches in these debates into lists of words, labelling them according to the MP who made the speech. This parsing was altered depending on the applied optimisations - I discuss some of these in §3.4.4. The word lists were then used to create bag-of-words features, in which each element corresponded to an n-gram. This process omitted any terms that were used less than 10 times in the training set. These bag-of-words features contain lots of zeros and have high dimensionality, so they can then be reduced to smaller features using singular value decomposition, as discussed in §3.4.4.5.

3.4.2 Avoiding Overfitting

I took appropriate measures to avoid overfitting. Firstly, before implementing the program to determine the stance of MPs' speeches, I wrote a classifier to determine whether a given email is spam. I developed this classifier using a bag-of-words representation and

a support vector machine, as this is how I planned to write the project's main classifier. Since both tasks are supervised binary classification sentiment analysis problems, I could adapt the spam email classifier, so it determined the sentiment of MPs' speeches. Using this development procedure meant that I avoided making the classifier overly specific. It also provided a further way to evaluate the project, as I discuss in §4.3. In addition to this, during development I only ever used the training data, comparing different implementations using averages of F1 scores across cross-validation folds. This meant that there was no scope for developing a classifier that was specifically tailored to produce good results on the testing set.

3.4.3 Scikit-learn

I used the Scikit-learn library to implement the classifier, as it provided a good implementation for both a support vector machine and a naïve Bayes classifier, which both have significant documentation [21, 23] and community support. Scikit-learn is built on other libraries which improves its performance. For example, uses Cython, which is supports C code within Python programs [2]. In their evaluation of machine learning libraries in Python, Pedregosa et al. showed that the Scikit-learn support vector machine was the faster than any other implementation they tested [30]. A further benefit of Scikit-learn is that its naïve Bayes classifier, which used similar syntax to the support vector machine, so it was easy to establish a baseline F1 score.

3.4.4 Optimisations

3.4.4.1 Stemming

Lemmatisation is the process of using morphological processing to reduce a word to its dictionary form (lemma). Unfortunately, lemmatisation is expensive, so stemming is often used as an approximation. Stemming is the process of reducing a word to its stem, using a collection of rules that adapt or remove a word's suffix. Stemming algorithms use heuristics to find a word's stem. In the bag-of-words representation used in this project, stemming means that words with the same stem are counted in the same feature. This is useful, as it is probable that words with the same stem will have similar semantics. Unfortunately, stemming does not perform as well as lemmatisation. For example, a lemmatiser might map 'are', 'be' and 'is' to the lemma 'be', since they have the same meaning, whereas the Porter stemmer would leave each of these words unchanged. A further example of where stemming does not have the desired effect is seen when stemming 'author' and 'authority' - using the Porter stemmer, both words map to 'author', despite their different meanings. I implemented stemming using the Porter stemmer algorithm [31].

3.4.4.2 Stopword Removal

Stopwords (such as 'the', 'a' and 'it') are frequently used words which carry less meaning than other words in a text. In many sentiment analysis applications, stopwords will

```
def remove_stopwords(word_list, black_list, white_list):
    ''' Returns a list of words (with stop words removed), given a word list '''

    stop = set(stopwords.words('english'))
    return [word for word in word_list
            if (word in white_list) or ((word not in stop) and (word not in black_list))]
```

Listing 3.6: A function which removes the stopwords and words on from a given list of words.

not provide any information about the sentiment of a text, due to the fact that they frequently occur in most texts (regardless of sentiment). However there are also cases where it is useful to consider stopwords in classification - for example, a lack of prepositions is often associated with less formal texts. I implemented stopword removal using the English stopwords provided by the Natural Language Toolkit. Further to this, I added functionality to whitelist words on the stopword list and add other words to the list. Listing 3.6 shows the simple function I implemented to remove stopwords from a given list of words.

3.4.4.3 N-grams

The major limitation of the bag-of-words model is that it doesn't represent the order of words. For this reason, many bag-of-words implementations use n-grams instead of just individual words. An n-gram is a sequence of n words, ordered as they are in the text they were sampled from. Using n-grams enriches the representation of a text; if we know the 2-gram 'British troops' is in a given text, it tells us more than just the fact that the words 'British' and 'troops' are both in the text since such a text will not necessarily mention British troops. N-grams also give context to words that are otherwise semantically ambiguous. For example, the stem 'author' is semantically ambiguous, since it is the stem of both 'author' and 'authority'. However, if the stem is part of an n-gram, such as 'palestinian author', in the context of the House of Commons debates related to the Iraq war (which do not discuss Palestinian fiction), 'palestinian author' is clearly derived from the phrase 'Palestinian authority' (or another word that is semantically related to authority). I use this example since this the 17th most pro-war n-gram used by the classifier (as shown in table 4.1). Due to these potential gains, I implemented functionality to extract n-grams from MPs' speeches.

3.4.4.4 Learning Settings and Hyperparameters

Once I had implemented stemming, stopword removal and n-grams as well as number-grouping and L2 normalisation, I devised a way for the program to automatically determine whether or not these potential optimisations should be applied. To facilitate this, I used a settings dictionary to determine the control flow of the program, as demonstrated in listing 3.7. I then used stratified cross-validation to determine the best value for each of the settings. In order to speed-up the training, I limited database accesses by retrieving and storing all the necessary data at the beginning, then getting stored data in each iteration of the cross-validation.

```
def generate_word_list(body, settings):
    ''' Returns a list of words according to the given settings, given a text '''

    body = remove_punctuation(body)
    body = body.lower()

    word_list = body.split()

    if settings['remove_stopwords']:
        word_list = remove_stopwords(word_list, settings['black_list'],
                                                settings['white_list'])

    if settings['stem_words']:
        word_list = stem_words(word_list)

    if settings['group_numbers']:
        word_list = group_numbers(word_list)

    return get_n_grams(word_list, settings['n_gram'])
```

Listing 3.7: A function which uses given settings to produce a word list of a given text.

I used the same stratified cross-validation scheme to determine the hyperparameters. The difference in choosing hyperparameters is that they are highly dependent on each other, so sequentially determining the value each hyperparameter would not work, as it does for choosing settings. Because of this, I decided to implement a variant on a grid search, which exhaustively considers every combination of hyperparameters within a range for each parameter. Rather than using a simple grid-search, I devised a three-phase search broken into the following steps:

1. Perform a standard exhaustive grid-search.

2. If there is one set of optimal hyperparameters and the value of one of the parameters is at the extreme of the initial range for that parameter, extend the search space to include more extreme values of the parameter. Iterate this step until the optimal set of hyperparameters does not lie on the boundary of the range of any of the hyperparameters.

3. Perform a grid search with a finer granularity around the current optimal combination of hyperparameters.

Figure 3.6 gives a graphic illustration of how this algorithm works. The search in the diagram is over ranges for two parameters (for example, C and γ in an RBF kernel), where each row represents a particular value for one parameter and each column represents a particular value for the other. The three grids represent the of hyperparameters considered at each of the three consecutive stages of the algorithm. The shaded squares indicate the optimal combination of hyperparameters in each phase. Since the best hyperparameters in the first step of the algorithm are in the furthest right column, we extend the search space to the right, as shown in the second grid. This algorithm extends to an arbitrary number of dimensions and in practice, I used log-scales where they were most appropriate.

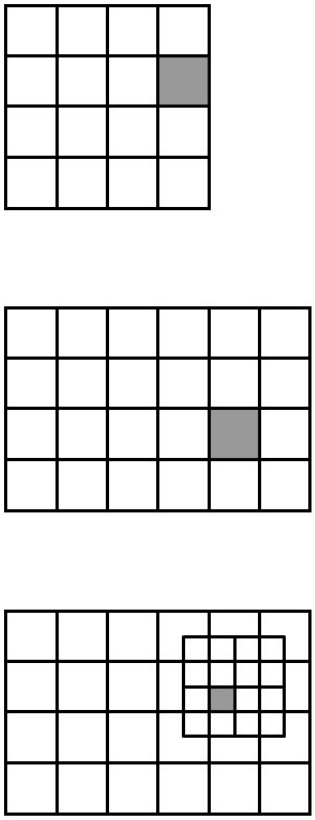

Figure 3.6: Diagrams illustrating how the hyperparameter search is conducted.

3.4.4.5 Dimensionality Reduction

The running time of an SVM classifier increases with the size of the features. It, therefore, makes sense to use dimensionality reduction techniques to improve the efficiency of classification. In this project, I used principal component analysis (PCA), as it is a relatively efficient method and there is good library support for it. Initially, I used the Scikit-learn implementation of PCA [22], however after testing it I found that I could perform the feature reduction more efficiently using the NumPy singular value decomposition function [10] and using the resulting matrices to perform reduce the size of the features (which is how PCA is carried out [8]). This implementation also proved too slow to be practical, so I adopted an approach using sparse matrices, which significantly improved the performance. Listing 3.8 gives my Python code for this dimensionality reduction.

3.5 Summary

In this chapter, I described the overall system architecture and discussed the work I undertook to implement the project's components. Throughout the preparation and implementation, I made decisions to facilitate the evaluation, which is detailed further in the next chapter.

```
def reduce_features(train_features, test_features, rank=300):
    ''' Performs the same principle component analysis on given train and test features '''
    sparse_train_features = csr_matrix(train_features).asfptype()
    sparse_test_features = csr_matrix(test_features)

    _, _, v_transpose = svds(sparse_train_features, k=rank)

    truncated_v = v_transpose.transpose()

    return sparse_train_features.dot(truncated_v), sparse_test_features.dot(truncated_v)
```

Listing 3.8: A function using NumPy array operations to perform singular value decomposition on two sets of given features

Chapter 4

Evaluation

In this chapter, I assess the successes and failures of this project. To follow a similar order to the implementation section, I first analyse the data in §4.1, then evaluate the classifier's results in §4.2 and in §4.3 I evaluate the results for the spam email classifier. I conclude this chapter by assessing the extent to which the project met its goals in §4.4.

4.1 Database Analysis

Since the result of the data acquisition described in §3.3 was predominantly a textual dataset, it was difficult to quantitatively analyse this part of the project, but I performed various checks in order to establish that the dataset was constructed correctly.

Firstly, the defensive programming style I adopted when writing the code to fetch, collate and store the data was geared towards facilitating the evaluation - wherever there was an error with the data, I outputted the relevant information and didn't write the data to the database. I then manually checked these outputs to see if there were any data that should have been written to the database but were not due to the error. All of the outputs related to instances where there were problems with the Hansard, which indicates that my program worked correctly.

I also assessed the database by performing a series of unit tests on the database class's methods. For these tests, I used the original data sources (the Hansard and data.parliament.uk) to create a series of expected outputs from a set of queries I wrote. I then ran the queries and the actual results were 100% consistent with the expected results. Further to this, when accessing the data to develop the project's classifier, the database didn't produce any errors. Given that the primary function of the database was its use for this classifier, this supports the success of the data acquisition section of the project.

The main quantitative analysis I carried out on the database was considering the word (and n-gram) frequency distribution. According to Zipf's law, the frequency of the i^{th} most frequent term is inversely proportional to i (the word's rank). More specifically,

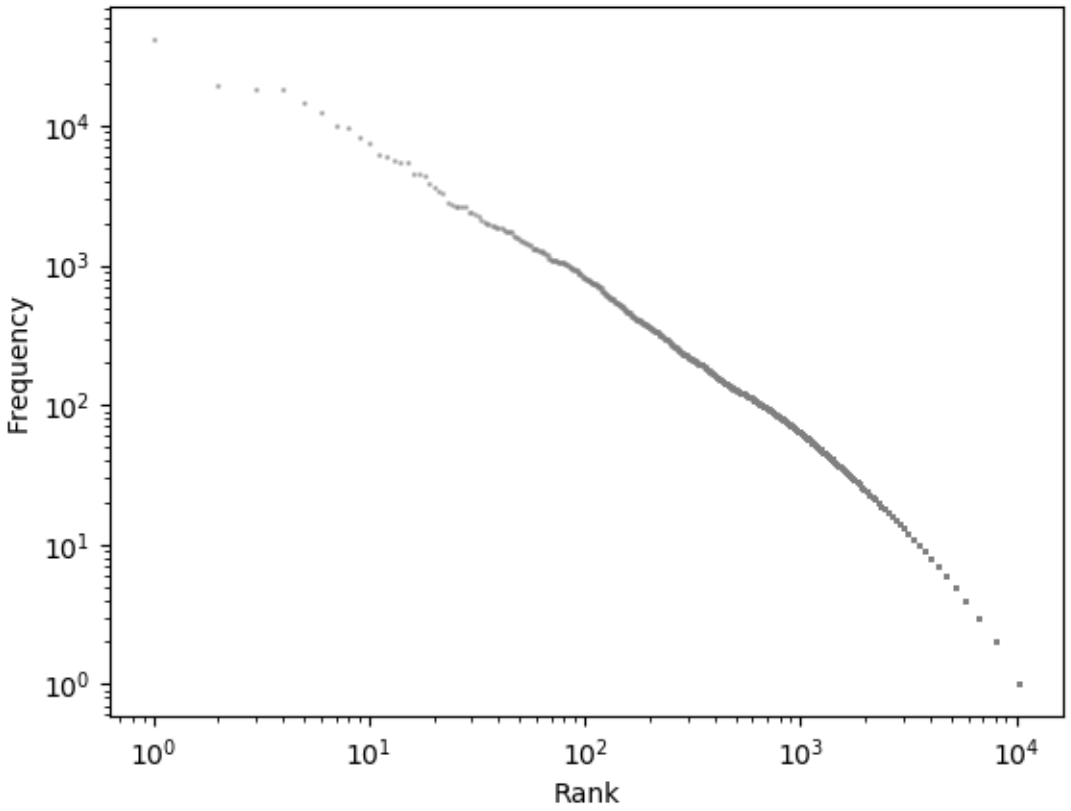

Figure 4.1: A scatter plot of frequency against rank for all the words in the debates considered in this project.

there is a linear relationship between the logarithm of the frequency of a word and the logarithm of the rank of the word [26]. Although Zipf's law is just an empirical observation, if the database was constructed correctly, we would expect it to obey this rule. Figure 4.1 shows a scatter plot of frequency against rank for all the words in the debates considered in this project, constructed using the project's database. The graph is plotted on a log-log scale, which is why we see the linear correlation that we would expect of a natural corpus. The coefficient of determination, r^2, for the plotted data is 0.97 ± 0.07[1], which provides further evidence for the project's corpus obeying Zipf's law and supports my hypothesis that it is properly constructed.

Figure 4.2 is a scatter plot of probability offset against their frequency (on a log scale) for all n-grams (where $n \leq 5$) that occur at least 50 times in the speeches used by the classifier. The probability offset of a given term is given by:

$$\frac{\sum_{s \in P} c_s(\text{term})}{\sum_{s \in P \cup N} c_s(\text{term})} - \frac{\sum_{t \in T} \sum_{s \in P} c_s(t)}{\sum_{t \in T} \sum_{s \in P \cup N} c_s(t)} \tag{4.1}$$

[1] Although the standard error suggests it could be, the coefficient of determination cannot exceed 1.

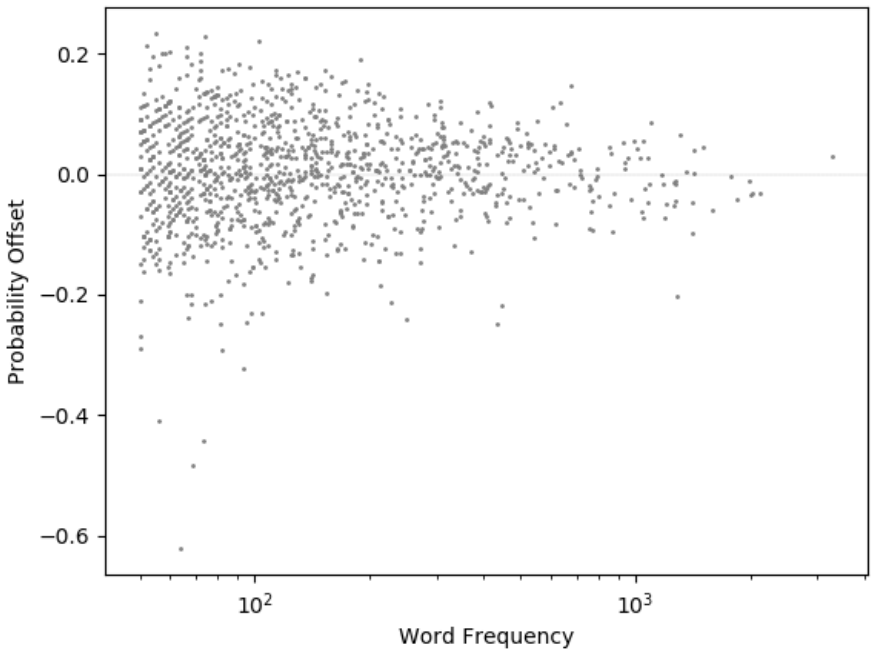

Figure 4.2: A scatter plot of probability offset against frequency for the n-grams in the training set.

where $c_s(t)$ is the occurrences of term t in speech s, T is the set of the n-grams in the training set (with frequency ≥ 50 and $n \leq 5$), P is the set of positively labelled (pro-war) speeches in the training set and N is the set of negatively labelled (anti-war) speeches in the training set. The first part of the expression is the probability of an occurrence of the given term being in a pro-war speech, while the second part is the probability that a given occurrence of any term is in a pro-war speech. This normalisation is necessary due to the fact that the dataset is unbalanced. Any terms with a positive probability offset have a pro-war bias and any terms with a negative probability offset have an anti-war bias. The figure shows that in general, the more common terms are less biased, but due to their frequency, their biases will likely be more significant, meaning they are still very useful for classification. Tables 4.1 and 4.2 show the most pro-war and anti-war terms respectively. Since the classifier uses stemming, stopword removal and n-grams up to $n = 5$, the terms in these tables are processed in the same way. Table 4.3 gives other statistics about the contents of the database, which I computed as part of the database analysis.

4.2 Classifier Results

Table 4.4 highlights the successful performance of the classifier I developed for this project - it significantly outperforms the naïve Bayes classifier, which I decided would be

Term	**Probability Offset**	**Frequency**
grate hon	0.233652194	55
royal marin	0.229475289	74
river	0.220996222	102
interim	0.212323522	52
chapter	0.209409769	66
state israel	0.203349163	60
brigad	0.201050313	58
complianc	0.200571386	72
board	0.199840391	57
regiment	0.195941756	54
novemb	0.194258254	66
avail	0.190650751	189
taliban regim	0.186682497	72
saddam	0.182103742	91
practic	0.181780536	68
emphasis	0.180730116	56
palestinian author	0.177232325	97
reconstruct	0.175898183	85
educ	0.175676207	53
deliv	0.173524602	114

Table 4.1: The 20 most pro-war n-grams that occur at least 50 times in the debates considered in this project.

Term	Probability Offset	Frequency
war iraq	-0.621	64
cluster bomb	-0.484	69
cluster	-0.442	73
british govern	-0.409	56
oil	-0.3219	93
go war	-0.291	82
billion	-0.290	50
mani us	-0.270	50
bomb	-0.249	432
georg	-0.249	81
nuclear weapon	-0.246	95
bush	-0.242	250
drop	-0.237	67
court	-0.230	104
conserv	-0.230	98
american	-0.219	444
tell us	-0.216	74
authoris	-0.215	68
administr	-0.212	228
fire	-0.211	77

Table 4.2: The 20 most anti-war n-grams that occur at least 50 times in the debates considered in this project.

Quantity	Total	Used by Classifier
MPs	655	276
Speeches	19,513	2,83
Debates	1578	36
Tokens	4,189,328	592,295
Distinct words	57,288	19,254

Table 4.3: A breakdown of the numbers in the database.

Classifier	F1 score by speech	F1 score by MP
Naïve Bayes baseline	0.392	0.182
SVM	0.894	0.809

Table 4.4: The F1 scores of the project's main classifier and the baseline classifier.

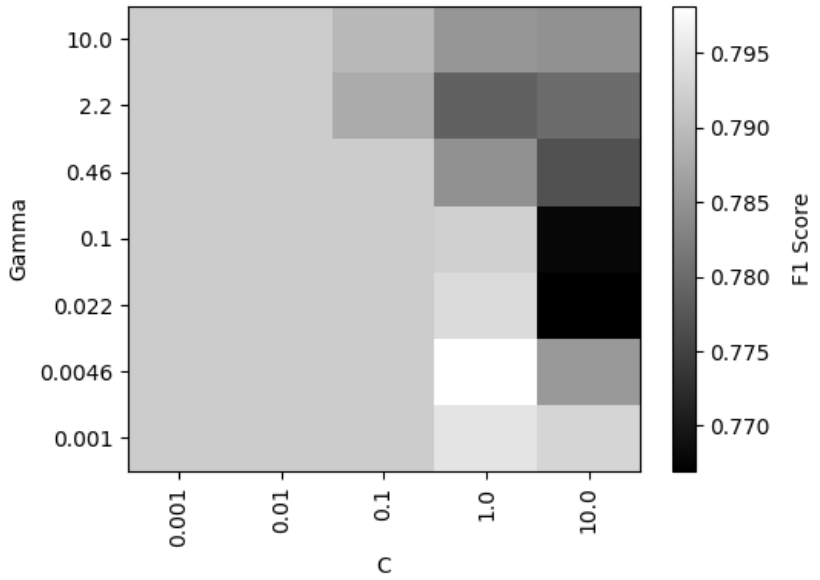

Figure 4.3: A diagram showing the grid of the first stage of the hyperparameter search for the RBF kernel.

the baseline in the initial stages of the project. Note that I have used F1 scores as the metric to compare classifiers, as they account for imbalances in data, as I discussed in §2.7.4.

The cross-validation learning algorithm established that the RBF kernel function would likely produce the best results. In doing so, it performed a search for the best combination of hyperparameters (as described in §3.4.4.4). Figure 4.3 illustrates the first step of the grid search for hyperparameters for the RBF kernel.

The learning algorithm which determined the kernel and hyperparameters also determined the settings, which are shown in table 4.5. As tables 4.2 and 4.1 demonstrate, n-grams were very useful in classification, so it makes sense that the learning algorithm chose to use them. Although using n-grams up to $n = 5$ produces a large number of features, this was easily handled using the dimensionality reduction I implemented. I detailed why stemming and stopword removal were useful in sections §3.4.4.1 and §3.4.4.2 respectively. Although it may be useful to group numbers for some applications, it was not in this case, as evidenced by the fact that 'resolut 1441' was one of the most pro-war

Setting	Value
n_gram	5
remove_stopwords	True
stem_words	True
group_numbers	False
normalise	True

Table 4.5: The settings learned by the SVM classifier.

terms (the United Nations Security Council Resolution 1441 was an offering to Iraq to encourage them to disarm [9]).

Interestingly, table 4.4 shows that for both classifiers, the F1 score was higher when calculated on predictions for each speech, rather than on predictions for each MP. This is counter-intuitive, since the MPs' predictions are made using an aggregation of their speeches - we might expect that having more data for a particular example would lead to a better classification. The most feasible explanation is that speeches made by MPs who make fewer speeches are more likely to be misclassified. This makes sense since if an MP is less sure of their stance on the war, their view is more likely to have changed over the period of time from which this project's corpus was created. Further to this, an MP who is less sure of their stance is more likely to be convinced to follow their party's line. I decided to test the hypothesis that MPs who made fewer speeches in debates related to the Iraq war were more likely to be misclassified, by carrying out a t-test. The mean number of speeches made for correctly classified MPs was 10, whereas it was only 5 for misclassified MPs. While this supports the hypothesis, the t statistic was 0.951, meaning it was only significant at the 80% level, which is not high enough to make any conclusive statements about the hypothesis.

4.3 Spam Email Dataset

As discussed in §3.4.2, developing a classifier to detect spam email before the project's main classifier had many benefits, including adding a further way to evaluate the system. This evaluation is possible since the spam email dataset I used was published by Medlock along with a paper containing state-of-the-art results for spam email detection [24], meaning that I could compare these results with my own. My classifier achieved 90% accuracy[2] on the spam email dataset, compared with the 93% given in Medlock's paper for SVM classification. Since Medlock's classifier was specifically tailored to detecting spam email and mine was not (since spam detection was not the focus of this project), the 90% accuracy my classifier achieved is a respectable result, providing further evidence for the success of this project.

[2]I give the accuracy rather than F1 score here, since this was the metric used in Medlock's paper.

4.4 Evaluation of Project Goals

The two primary goals of the project were defined in §2.1. They were to "Construct a database that comprises British texts on the Iraq war" and to "Develop a classifier that can determine the stance of the texts in the database". §4.1 provides evidence of my achievement of the first of these goals and §4.2 shows that I have successfully completed the second of the two goals. I refined these goals to give the project's core tasks in table 2.1, which I completed (as detailed in §3), before going on to complete extensions to the project.

4.5 Summary

This project was successful in achieving two main aims of creating a dataset of British texts on the Iraq war and developing a classifier to use this corpus. The novel method of using an MPs' House of Commons voting record to label their speeches proved to be successful, as demonstrated by the classifier's good performance.

Chapter 5

Conclusions

5.1 Achievements

Through carrying out this project, I created a dataset using data from the Hansard and data.parliament.uk. This dataset contains all the speeches and votes made between September 11th, 2001 and 18th March, 2003 (inclusive). I then used this dataset to develop an SVM classifier which achieved an F1 score of 0.894 on MPs' speeches, which is significantly better than the baseline 0.392, which was the F1 score of the naïve Bayes classifier on the same data.

5.2 Lessons Learned

The success of the project demonstrates that the Hansard is a great resource for NLP applications, especially as I showed that it can be labelled using House of Commons voting data. While the project did go well overall, the licensing issues with the initial dataset delayed the project from the off. If I were to do the project again, I would check the licences of any necessary resources before starting any other work, so as to avoid a similar problem. In addition to learning more about licensing of data, I learned a lot about NLP and machine learning techniques by doing this project.

5.3 Future Work

To improve the classifier further, I could adopt more complex models which could account for how MPs' stances change over time or how likely an MP is to follow their party's line in a vote where this contradicts the MP's own views. A new model could also use additional data from other sources, such as newspapers and social media to improve classification. The dataset could be extended over a longer range, thus making it useful for analyses of a wider range of political issues. I plan to create an API for this dataset, which will hopefully encourage further applications to be built using House of Commons data, which would, in turn, increase the accountability of MPs.