Sample Dissertation #3

(Front Matter removed)

Table of Contents

1. Introduction 8
    - 1.1 – Overview 8
    - 1.2 – Objectives 9
    - 1.3 – Challenges 10
        - 1.3.1 – Dataset Acquisition 10
        - 1.3.2 – Overfitting 10
        - 1.3.3 - Inaccurate Data 11
        - 1.3.4 - Model Bias 11
        - 1.3.5 - Time Management 11
    - 1.4 – Structure of Document 12
2. Background Research 13
    - 2.1 – Research into Problem and Solution 13
        - 2.1.1 – Sentiment Analysis 13
        - 2.1.2 – Machine Learning 13
        - 2.1.3 – Similar Projects and Studies 14
    - 2.2 – Technologies Researched 15
        - 2.2.1 – Languages 15
        - 2.2.1 – Machine Learning approaches 17
        - 2.2.3 – Data Acquisition 18
        - 2.2.4 - Text Parsing 18
    - 2.3 – Dataset Research 19
    - 2.4 – Existing FYPs 19
        - 2.4.1 – Anti-Bullying 19
        - 2.4.2 – Twitter bots 20
        - 2.4.3 – Coin Classification 21
    - 2.5 – Technologies Chosen 22
3. Design 24
    - 3.1 – Design Methodology 24
    - 3.2 – Component Design 25
        - 3.2.1 – Use-Case 25
        - 3.2.2 – Mock-up 26
    - 3.3 – Features 26
        - 3.3.1 – Prototype 26
        - 3.3.2 – Version 1 27
        - 3.3.3 – Version 2 28
        - 3.3.4 – Final Version 29
    - 3.4 – Architecture 30

4. Implementation 33
    - 4.1 – Development Components 32
    - 4.2 – Training Data 33
    - 4.3 – Machine Learning Models 34
    - 4.4 – Parliamentary Data Preparation 35
    - 4.5 – Parliamentary Data Classification 36
    - 4.6 – Django Web Application 36
    - 4.7 – Source Code Layout 39

4.8 – External APIs 40
5. System Validation 41
5.1 – Ad-Hoc Testing 41
5.2 – Unit Testing 42
5.3 – Initial Test Cases 42
5.4 – Web Application Test Plan 44
5.5 – Demonstration 46
6. Critical Evaluation 47
6.1 – Initial Project Plan 47
6.2 – Changes from Initial Approach 47
6.2.1 – Project Plan 47
6.2.2 – Parliamentary Data 48
6.2.3 – Strength of Polarity 48
6.2.4 – Results storage 48
6.3 – Project Versions 48
6.3.1 – Prototype 48
6.3.2 – Version 1 48
6.3.3 – Version 2 49
6.3.4 – Final Version 49
6.4 – Outcome 50
7. Conclusion 50
7.2 – Future Work 50
7.2 – Conclusion 51
8. Bibliography 52
8.1 – Bibliography 52
9. Appendix 55
9.1 – Installation Guide 54
9.2 – User Manual 55
9.3 – Table of figures 57

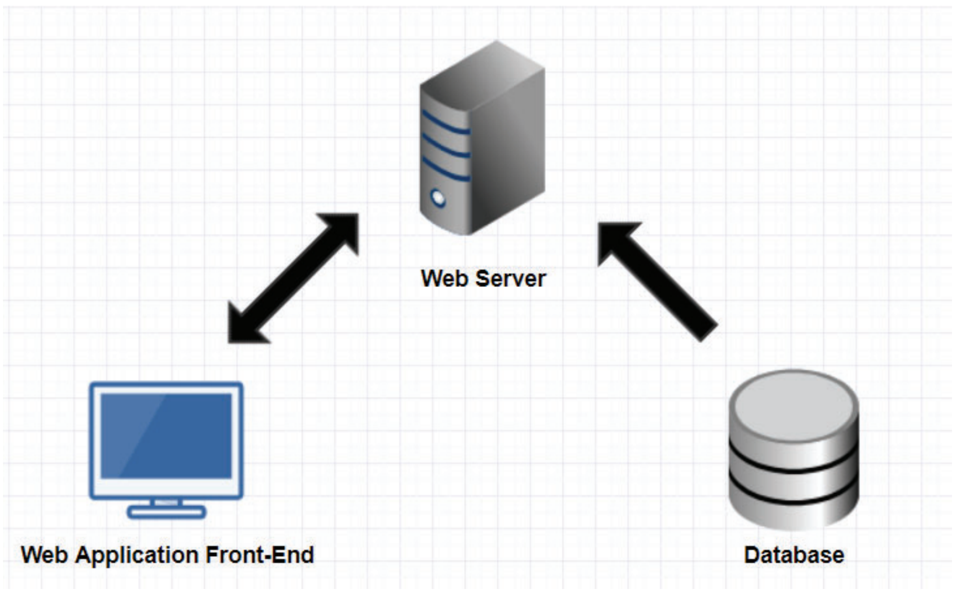
1 - Introduction

1.1 – Overview

The applications for an automated approach to sentiment analysis are almost endless. In the time of big data, sentiment analysis allows businesses to process thousands of documents to determine their sentiment in mere seconds. This means that huge quantities of unstructured data can be categorized and classified with ease.
Sentiment analysis is also important in the studies of news coverage, public opinion and political campaigning.

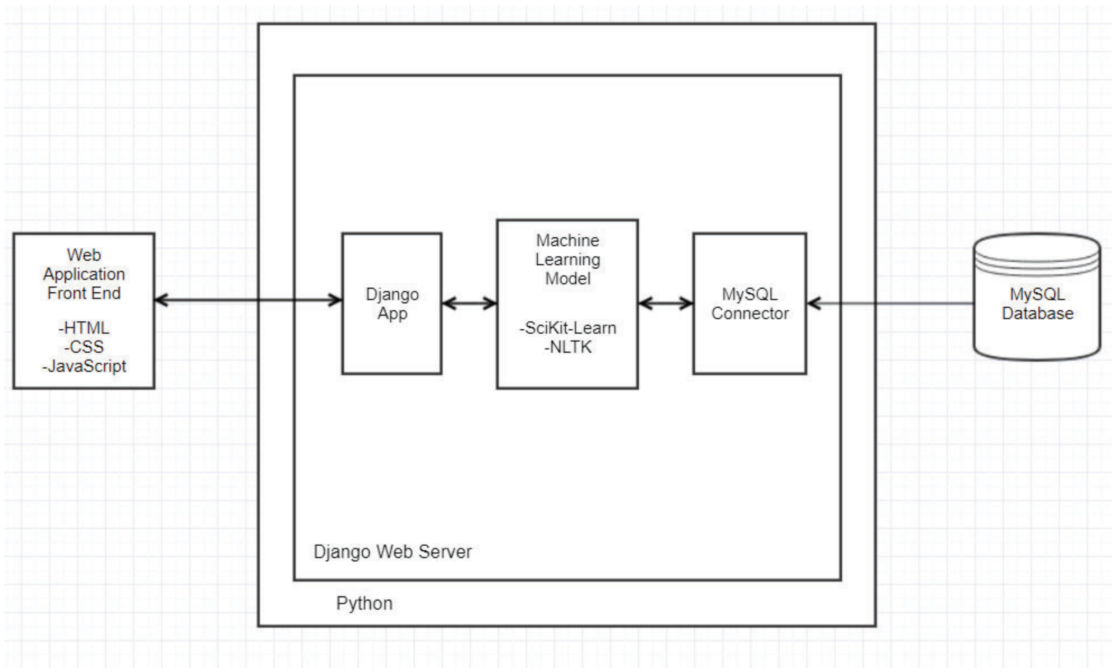
An automated approach is unique in that it relies on machine learning algorithms rather than on predefined rules [2]. Sentiment analysis using machine learning is represented as a classification task where the input to the classifier is text and the output is a classification of sentiment e.g. Subjective or objective, negative or positive.

Figure 1 - System Overview



The purpose of the project is to examine how an application can determine the sentiment of different types of text using machine learning. In addition to performing analysis of historical texts, an application will be developed to determine the sentiment of any entered text.
This application will require a classification model to be trained on pre-labelled training data. The technical architecture for the application can be described as follows.

Figure 2 – Software Architecture Diagram



A sentiment analysis tool such as this is broadly applicable to many different areas. Sentiment analysis can be used in social media analysis, tracking psychological and linguistic trends and for survey research.

1.2 – Objectives

The aim of this project is to provide a tool that can be used to analyse the sentiment of text such as subjectivity and polarity. I plan on using machine learning rather than a lexicon-based approach. This classification tool will be implemented in the form of a web application.

In addition to delivering this simply user application, a study will be conducted where statements made in parliamentary debates are classified based on their sentiment using the algorithms that have been written for the sentiment classification tool. Pre-labelled training data will be need to obtained. All parliamentary data will have to extracted from public online archives.

The results of the analysis of the parliamentary debates should be displayed and visualised as part of the user application. The application should also provide the user with some analysis of the features of their entered text to illustrate why it has received the prediction for its sentiment.

1.3 – Challenges

1.3.1 - Dataset Acquisition

There needs to be a way to download the transcripts of parliamentary historical parliamentary debates in order to perform analysis on them when my sentiment analysis model is completed. Although this data is publicly available online, you can only download one debate at a time. This presents the problem of having to manually pull over 10,000 files a website. This would be a very time-consuming process.
The solution to this is to automate the process using a selenium web crawler to navigate the website and download each transcript. Although this is a complex task in itself, this solution saves a lot of time.

1.3.2 - Overfitting

Overfitting or over-optimization is a common machine learning problem which occurs when a model is trained for maximum accuracy on a training set only to become too closely corelated to that particular set of data. This is a risk as the blocks of text in the training data are considerably shorter than the ones that will be in the test data.
To avoid overfitting, precautions are taken not to over-optimize by not including more parameters in the model than are justified by the data.

1.3.3 - Inaccurate Data

The accuracy and validity of the data used to build a machine learning model is an area of risk. If you don’t know where your data comes from or how it is produced, then you run the risk of training your model on bad data. This could be a disastrous mistake.
To avoid this, there needs to be an awareness of the lineage and pedigree of all the data used. The training set needs to be very well documented. The parliamentary data will be downloaded from the UK parliament website, since this is a direct source, the data can be trusted as being legitimate and authentic.

1.3.3 - Model Bias

In a predictive model, bias is an error derived from inaccurate assumptions made by the algorithm. It’s hard to avoid the bias of the developer getting build into the model, there needs to be an awareness of this to try to minimize bias during the training phase.
The best way to mitigate bias is to choose a suitable machine learning model. The best model to use is somewhat subjective and highly dependant on the task at hand. Based on research on classification algorithms, I think Naïve Bayes or Support Vector Machine would produce the least bias for the purposes of this system.
A balanced training set has also been chosen partially to reduce the risk of bias towards an underrepresented class.

1.3.4 - Time management

As with any medium to large scale project, the challenge of planning development and allocating time for each task is an important one. Failure to organise research and development could mean I simply won’t have enough time left to satisfy all my requirements.
To ensure everything is completed in a timely manner, the project has been separated into various stages of a plan with time allocated allowsa generous

amount of time to complete each step. This project plan has been documented thoroughly so it can be referred to or changed at any point during development. There are some aspects of the system with a higher priority to others so that they can be prioritized as a submission deadline approaches if necessary.

1.4 – Structure of this document

- Chapter 1 of this document provides an introduction to the project, briefly outlining the background, objectives and challenges involved.
- Chapter 2 will focus on the initial background research as well as the research performed throughout the project.
  This includes research in sentiment analysis, text classification, machine learning and specific machine learning concepts for text analysis and the coding languages and libraries investigated.
- Chapter 3 describes the methodology used in the implementation as well as outlining each planned version of the final system and the requirements of each one.
- Chapter 4 identifies the key development components and outlines the development of the user application.
- Chapter 5 details the testing and evaluation carried out on each component of the system at various stages of development.
- Chapter 6 discusses the planning and organization of the project and describes the general outcome of the project.
- Chapter 7 concludes the document by evaluated and reflecting on various parts of the research, development and validation processes.
- Chapter 8 consists of a bibliography containing references for the entire document.
- Chapter 9 is the appendix of the document which contains a user manual and installation guide

2 – Background Research

2.1 – Research into Problem and Solution

2.1.1 – Sentiment Analysis

Sometime known as Opinion Mining, Sentiment Analysis is a field of Natural Language processing that focusses on identifying the opinions being expressed in text by extracting features present in a piece of the text.
The main attributes of text identified by sentiment analysis are [3]:

- Polarity: whether an opinion is negative or positive
- Subject: the topic that is being discussed
- Opinion Holder: who is expressing the opinion

Sentiment Analysis is a topic of relevance today because of its many useful applications. In the age of big data, the amount of text data expressing opinions in publicly available over the internet. Using sentiment analysis, all of this unstructured raw information can be processed and transformed into useful structured data of the general public's opinion in relation to wide variety of things. This information has the potential to be highly useful and valuable for commercial purposes like market research and public relations.

2.1.2 – Machine Learning

Machine learning is a type of algorithm that become more accurate at predicting results or outcomes without needing to be explicitly programmed
Machine learning algorithms can be classified as either supervised or unsupervised [2]. Unsupervised algorithms require a human to provide the inputs and desired outputs to the algorithm. The human developer of the model determines which features the model should consider when making predictions. Supervised algorithms are often used for simple tasks such as text classification.

Unsupervised algorithms (sometimes called neural networks) are different in that they do not need trained with data labelled with desired outcomes, they instead use an iterative approach called deep learning to analyse data. These algorithms are more often used for more complex tasks than the ones in which supervised learning is often used, E.g. Image recognition and speech dictation software.
Neural networks are trained by processing massive amounts of training data and automatically identify relationships between various features of the data. The algorithm can then use this knowledge base of relationships to interpret new data and predict outcomes. This technique is considerably more complicated than supervised

learning and has only recently started to become more feasible due to the arrival of big data.

Figure 3 – Machine Learning Diagram

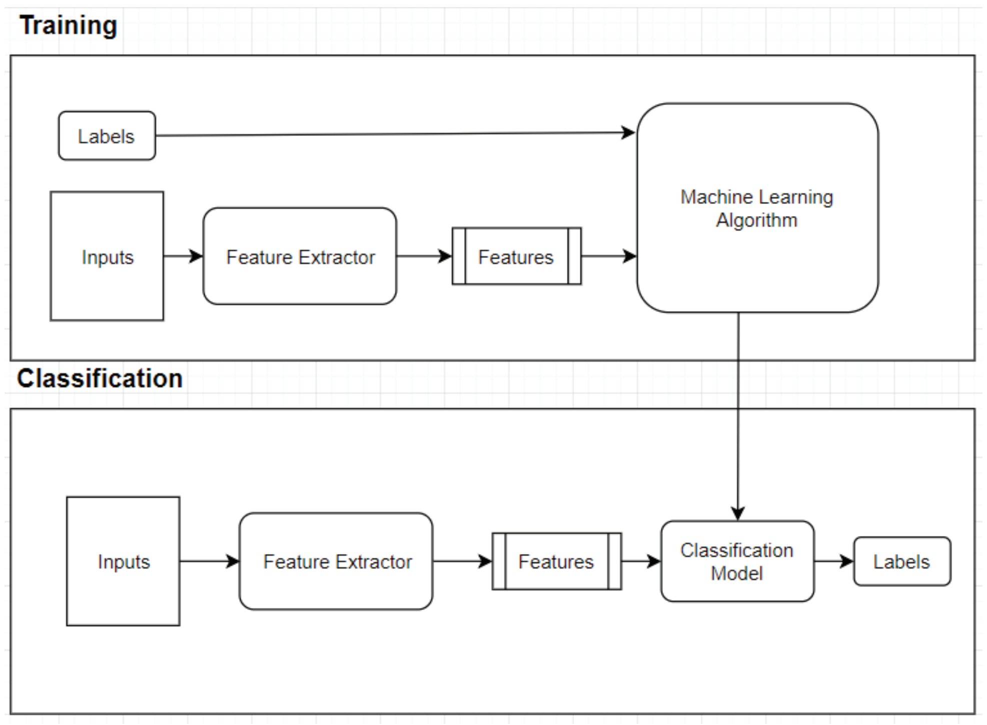


Diagram of the processes involved in the development and training of the sort of machine learning classification model that this system will use to classify text by its sentiment.

2.1.1 – Similar Projects and Studies

The Initial phase of research was focussed entirely on the broad areas relevant to the project like Machine Learning, Text Parsing and Opinion Mining.

To gain a better understanding of these areas, some technical tutorials on machine learning tasks were completed and several studies on sentiment analysis were read. The initial idea to use parliamentary data came from reading a 2010 study on the Dail [4]. Here a support vector machine was used to classify Dail questions on a local-national basis. It seems SVM algorithms are well suited to text classification in this example, this instance played a part in the choice of machine learning algorithm later.

According to SentiWordNet [5], a lexical resource for sentiment analysis, opinion mining problems can be organized into three categories:

1. Determining Polarity: whether a text expresses a positive or negative sentiment
2. Determining Subjectivity: whether text expresses an opinion or is factual.
3. Determining Strength of Polarity: how strongly a negative or positive opinion is stated.

Of these three categories, Subjectivity seemed like it would be the most interesting when applied to political discourse. However, it was determined that the sentiment analysis tool must be able to perform analysis in all three categories.

While researching other studies of sentiment analysis carried out on political speech, an example of sentiment analysis on Dutch parliamentary data [6] was found. One aspect of study was to do with assessing the subjectivity of arguments made in the Dutch house of representatives.

To accomplish this, transcripts of parliamentary proceedings were scraped from a website and transformed into a machine-readable format, in this case HTML, as I plan to do.
To evaluate the performance of their machine learning algorithm, a gold standard corpus was developed by two human annotators manually categorizing chunks of text. Several machine learning algorithms were used in classifying the data:

1. Naive Bayes
2. IBK nearest-neighbour
3. Support Vector Machine
4. ZeroR

Of these, Naive Bayes seemed to produce the best results while ZeroR classified all paragraphs as objective. The support vector machine algorithm also produced viable results.

2.2 – Technologies Researched

During research, lots of reading was done on the tools and libraries available for machine learning and sentiment analysis. Research was also done into perhaps the biggest design decision of the project. Which programming language will be used?

2.2.1 - Languages

- R – R is one of the most popular languages in data science. It is an open source language rooted in statistics and data visualisation. Although still a mainstay in the industry, R is increasingly being used as a complimentary language. [7]

- Python – Designed to be simple and readable, Python has become the most popular language for machine learning. This is because it is well suited to dealing with raw unstructured data. There are many useful and easy to implement packages you can use to process all kinds of data.

Despite this being such an important decision, not many alternatives were researched as there was already a strong preference for one language before research had even begun.

Some Python Libraries

- NumPy – NumPy is the fundamental package for scientific computing in python. This library adds support for large multi-dimensional arrays and matrices as well as a large amount of high-level mathematical functions.

- Scikit-learn - this open source library comes with several built-in machine learning algorithms and is commonly used in machine learning projects, particularly ones involving text classification of some sort. This is on of the most widely used technologies in machine learning projects. It will feature heavily in this project

- Keras - a neural network library written in Python, this technology was designed to enable easy and fast creation of and experimentation with deep neural network models. This library could be useful if it is decided that one of the classifiers will use a deep learning approach.

- Theano - released back in 2007, Theano is a python library and optimizing compiler for evaluated mathematical expressions. Due to its age, Theano is considered an industry standard in machine learning. [8] Theano seems to simplify working with mathematical expression in python. It also comes with code testing capabilities which might come in handy for this project.

- Pandas – pandas is a powerful Python data analysis toolkit, it provides flexible data structures designed to make working with labelled data easier. Built on NumPy, pandas provides tools for loading data from various file types and handles missing data well.

- Natural Language Toolkit – The NLTK is a suite of Python libraries and programs that apply English language data to statistical natural language processing. It provides a suite of text processing libraries and interfaces to a large number of corpora and lexical resources.

- Bokeh – this open-source interactive visualisation library provides elegant and versatile data visualisations. Bokeh graphics are dynamic and interactive as well as being easily embedded in web applications.

2.2.2 - Machine Learning Approaches for Sentiment Analysis

- Naïve Bayes Classifier:

  This is a simple classifier based of Bayes' theorem of conditional probability. The classifier assigns a class to a document and uses the probability of features appearing in each class to determine its prediction [9]. The probability of a feature appearing in each class is calculated independently, this is where the "Naïve" part of the name comes from, the assumption is that the probabilities for each feature occurring are completely independent.
  Theses individual probabilities are combined to find the probability of belonging to a class.
  Despite the fact that feature probabilities being conditionally independent clearly does not reflect reality, Naïve Bayes still performs well in text classification problems.

- Maximum Entropy:

  Maximum entropy is a probabilistic machine learning classifier based on empirical data which has been shown to be effective in text processing applications, even outperforming Naïve Bayes. Maximum entropy makes no assumption as to the conditional independence of feature occurrences.
  This classifier is based on the principal of maximum entropy, being that one should choose the most uniform model that will satisfy and constraints. [10]
  Training a maximum entropy model can be difficult and take a long time due to the problem of estimating the parameters of the model. However, once the model has been trained it is fast and produces good results.

- Support Vector Machine:

  Support Vector Machines are based on the structural risk minimization principal [8], balancing the model's complexity with its success of fitting the training data. Unlike the classifiers previously mentioned, SVM is a form of large margin nearest neighbour classification. SVM plot each instance in either two- or three-dimensional space. The training procedure involves finding the maximum margin hyperplane and represent it as a vector. This will separate each instance vector into classes and insures the margin of separation is a large as possible.
  Where there is little variance in class instances, a linear SVM can produce a well-defined hyperplane which can separate each instance into classes clearly.
  When there is much inter-class variance, a non-linear SVM can be used to generate a curved margin hyperplane, resulting in more accurate classification

and less non-binary labels. Support Vector Machines generally require more parameter tweaking than other classifiers.

2.2.3 – Web Application Frameworks

- Django

  Django is a full stack development framework for developing web applications using python. This open-source framework uses the model view controller architecture. Django emphasises reusability of components, reducing the total amount of code needed in an application.
  An object relational mapper is used with a URL based controller to operate the switching between different data models. Django's configuration allows for external code to be ported into applications with relative ease. This frameworks scalability makes it ideal for small development projects while also being used on many large-scale websites [11].

- Flask

  Flask is a web framework which provides tools and libraries used to build web applications. Flask uses a template engine to maintain consistency in the style of a site with many pages. It is considered to be a micro-framework, this means that it has very few dependencies or external libraries. Flask's status as a micro-framework means that it is very lightweight and fast. Although offering less dependencies than Django, flask is considered to adhere to philosophy of Python more closely with its minimalistic approach.

2.2.4 - Data Acquisition Technologies

- Selenium- Selenium is an open source software testing framework for web applications. It comes with its own domain specific language to write test scripts in many different programming languages. Selenium requires a driver for the browser it's being used in for it to work.

.

- AutoIt- AutoIt is an automation language for windows. Its primary use is to write automation scripts for windows applications which can be converted into executable files. Like Selenium, AutoIt has web crawler capabilities. Crucially, AutoIt allows for the automation of the saving of files from a web page with its 'inetget' function.

2.2.5 - Text Parsing Technologies

- Java DOM Parser- The (Document Object Model) Parser is an interface in Java that allows programs to access and update the format and style of XML documents. The parser works by loading the XML object into memory as the document, allowing the programmer to navigate through the elements of the XML object. The DOM parser is easy to use and performs well with small documents. However, since it loads the full file into memory, its performance gets significantly worse as the file size gets bigger.

- Java SAX Parser- The SAX Parser is a Java API used to parse XML documents, it's different to the DOM parser because it does not load the full XML document into memory. This makes SAX faster and less memory intensive. [12] SAX uses the XMLReader interface and event handlers to parse documents.

- BeautifulSoup - Beautiful Soup is a Python library for pulling data out of HTML and XML files. It uses a constructor to take an XML or HTML document in the form of a string and parses the document to create a corresponding data structure in memory. BeautifulSoup provides simple ways of searching and navigating this data structure.

2.3 – Dataset Research

One of the most import aspects of the development of an accurate machine learning model is the selection of a good training dataset. Luckily, there are many publicly available datasets of text labelled by its sentiment. Most of these datasets are compiled by extracting text from online sources such as tweets and reviews.

Cornell University [13] have published several widely used datasets for text subjectivity and polarity that can be applied to this project. Despite these sources being seemingly adequate for the task at hand, several other datasets were acquired and used to create additional models in order to compare and contrast their accuracy and performance.

2.4 – Existing FYPs

During research, several other final year projects relating to the fields of machine learning and sentiment analysis were studied in order to compare the methodology and approach used in these projects to the planned methodology and approach for this one.
Some of the projects studied are as follows:

Anti-Bullying with Machine Learning [14]

Student: Shane O'Neil

Description:

The goal of this project was to use various machine learning and data mining techniques to build predictive models to detect cyberbullying or other abuse content.

Complexity:

Text data is highly unstructured compared to numerical data. It can be difficult to apply any sort of structure to this data in order for a classifier to understand it.
Abuse content is a somewhat subjective evaluation which could prove difficult for a model to classify.

Architecture:

The dataset was cleaned and features were extracted using the Python programming language.
The text representation was implemented in a Bag of Words form using Sci-Kit Learn's CountVectorizer module.
Several different classification algorithms were used and implemented using Python.

Strengths and Weaknesses:
I thought the strength of this project was the experimentation phase after the final evaluation had been performed. There were several pertinent experiments carried out and their description was clear and comprehensive.

I think the weakness of this otherwise excellent project was the relatively inconclusive level of accuracy in prediction for most datasets.

Detecting bot twitter accounts using Machine Learning [15]

Student: Emmet Hanratty

Description:

The goal of this project was to create a machine learning program that would determine if a twitter profile was genuine or a bot account. There would also be a user interface where a user could test an account by inputting its URL.

Complexity:

The dataset of twitter accounts used for this project is very complex. There are many different use cases for twitter accounts, genuine accounts can post rarely or very frequently, this could make training a machine learning to classify accounts with unusually low or high levels of activity more challenging.

Technologies:

Python, Numpy, MatPlotLIb, Scikit-Learn, Pandas, MySQL, Tweepy, Yandex

Strengths and Weaknesses:

The strength of this project was that it delivered a functional and sophisticated user application that fulfils a very tangible consumer need.
From reading the excellent project documentation however, it seems the weaknesses lie in the performance of application. Along with issues using the Twitter API, account followers need to be fetched one at a time making the feature to show how many of an account's followers are bots run very slowly on accounts with a lot of followers.

<u>**Euro Coin Classification Using Image Processing & Machine Learning**</u> [16]

Student: Yumin Chen

Description:

The goal of this project was to investigate the visual features of euro coins and develop models that can be used to classify coins by their denomination from viewing natural images. This is accomplished using image processing with machine learning

Complexity:

Object recognition using image processing is a highly complex and difficult task. In order to distinguish between denominations of coin, a model had to be designed that could recognise subtle differences between the visual properties of each denomination.
Extensive research and experimentation were also required to determine which features were useful.

Architecture:

The primary programming language that was used to develop the classification system is Python. In particular, the OpenCV computer vision library was used to implement the image processing tasks.
To implement the demo application, Ionic frameworks was used to develop a cordova-based cross-platform mobile app that communicates with a specifically developed web-based API service.
.
Strengths and Weaknesses:

The biggest strength of this project was the outcome of the image classification system. The models built for object recognition showcase an impressive level of complexity.
Although difficult to find weak areas of this excellent project, but it seems that planning was the main issue resulting in forced shifts in implementation as new technologies were researched

2.5– Technologies Chosen

Requirement	Technology chosen + Reason
I will need a database management system to make up the data tier of my system, which will store all training and evaluation sets, as well as my unlabelled parliamentary data.	MySQL is an open source database that is stable, reliable, powerful and I have plenty of experience using it. I think MySQL will be suitable for this project as it is well suited to use in web applications.
I'll need to use a programming language well suited to machine learning.	Here Python is the obvious choice since there is such an extensive collection of open source libraries available such as NumPy, Sci-kit-learn and Pandas suitable for data science and machine learning.

	I also have a lot of experience using python which made it by far the most attractive option from the very beginning. R would also be suitable for this project. Had I not been proficient in python already I may have gone down the R route. However, it's very easy to use R within python using the package rpy2 so I still have the option of using some R code if I need to.
I'll need to implement a machine learning algorithm to classify statements.	From the algorithms I have researched, I think the most suitable would be Naïve Bayes and Support Vector Machine as both have shown to produce robust results in sentiment analysis applications. [17] I think Naïve Bayes will work well for my purposes. Sci-Kit-Learn allows for Multinomial Naïve Bayes to be implemented, using multinomial distribution should garner better results since the conditional independence that is assumed by normal Naïve Bayes will not reflected in my data. [18] A Support Vector Machine based model is still a possibility and may be used as a backup if the NB algorithm doesn't produce good results for whatever reason.
To implement the web-based user interface I'll need to use a framework.	Since I'll be using Python for the rest of the system, it makes sense to use a framework that is written in Python and integrates all its libraries This is why I chose to use the Django framework. This means I'll be able to implement all of my machine learning logic within the Django python code easily. The interface will need to graph and visualize results, Django makes it easy to implement front end technologies such as Javascript, CSS and HTML to implement this.

Figure 4 – Chose Technologies Table

Technology	Version
Python	3.7.1/3.7.0
Java	1.8.0_144
Django	2.1
Sci-kit-learn	0.20.0
NLTK	3.3

NumPy	1.15.4
Pandas	0.23.4
SciPy	1.10
MySQL	8.0.13
Selenium	3.141.59
AutoIt	3.3.14.5
Jinja	2.10
Bootstrap	4.1.3
Bokeh	1.0.4

3 - Design

3.1 – Design Methodology

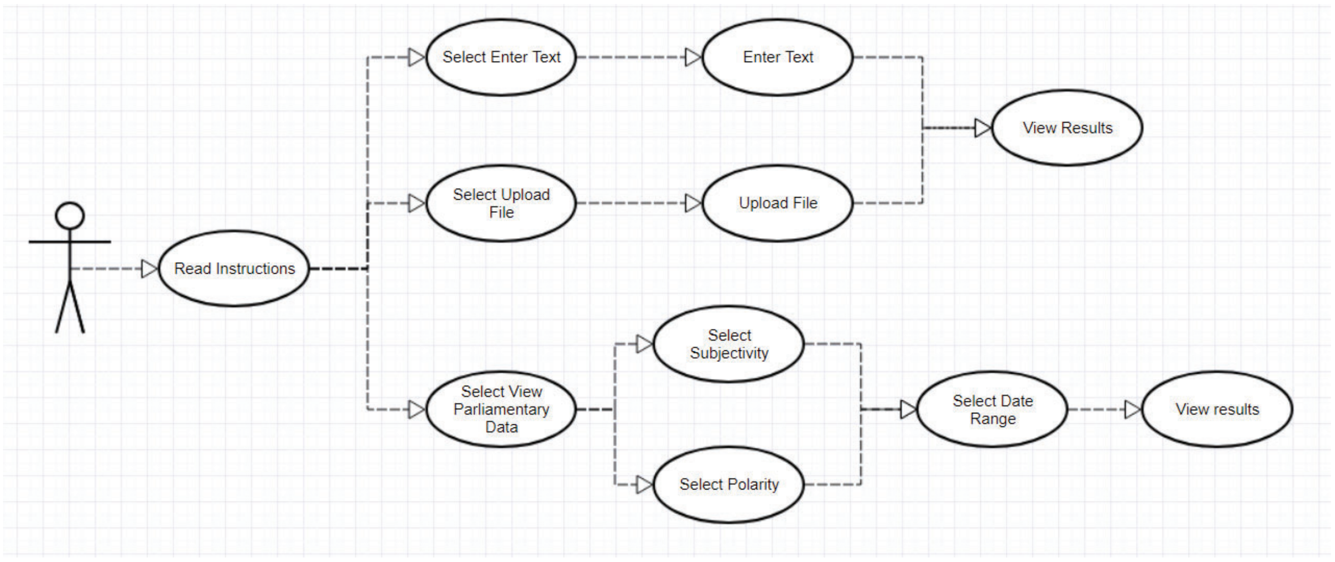
This project will take an iterative approach to the development of my system. The project will be split into several different version or iterations each one containing

more features than the last. Each of these iterations will be designed, implemented and tested before the development of the next iteration will begin.
Before any design or implementation can take place, an extensive reach phase must be completed. The goal of this research is for the developer to gain a working knowledge of relevant technologies in order to select appropriate ones for use in development, define requirements and select an appropriate methodology to use.
Some technologies that must be selected in this phase: Programming languages, Machine learning models, web development frameworks,
While the user application and machine learning models will be developed iteratively, a more traditional incremental approach will be used to extract, clean and parse the parliamentary data needed as these tasks may take a long time to complete. The developer will work on preparing this data and developing the early iterations of my system simultaneously. The parliamentary data analysis is not scheduled to be included in my system until late in development although it may be implemented sooner if the data acquisition and preparation tasks are completed sooner than anticipated.
Using this methodology, functional prototypes will be developed early, allowing the developer to get their hands on a functioning version of the system should help them spot any potential defects or problems early in the project lifecycle. Once an iteration is completed, testing is carried out to see how well the current version satisfies the requirements. Whatever is learned or noticed about the system during testing will help in designing the next iteration of the system.

3.2 – Component Design

3.2.1 – Web Application Use Case:

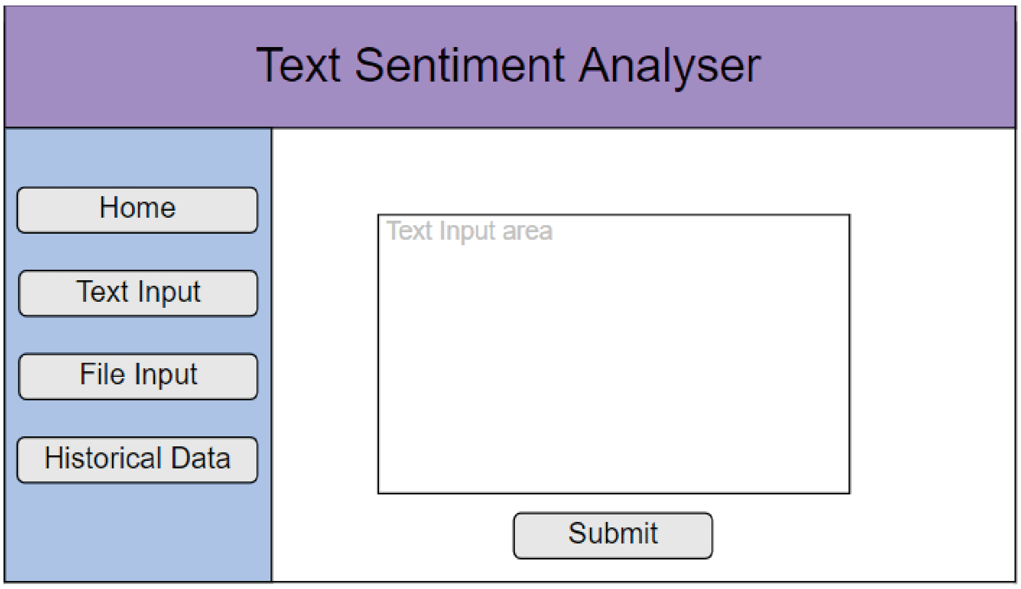
Figure 5 – Use Case Diagram



The above use case diagram represents the experience of a user of the web application. Upon arriving at the home screen, the user is presented with a landing screen containing some information about the application, and three options to either enter their own text to be classified or view the analysis of the parliamentary text. Enter Text and Upload file are similar screen's prompting the user to either enter their own text in an input field or select a local file to upload. The results page is the same for both options, displaying the sentiment classification of the text to the user.
If a user chooses to view the parliamentary data, they are prompted to specify exactly what data they would like to be visualised and in what range of dates before the data is visualised in a results screen.

3.2.1 – Web Application UI Mock-up:

Figure 6 – Senti-Text Mock-up



Above is a mock-up of the user text input screen of the web application. It was created after the prototype was developed to serve as a visual draft for subsequent versions of the application.

3.3 – Features

Since the development of the system will be implemented in iterations, the features will be listed below in order of which version they will be implemented with.
Below are the current plans for each iteration of the system, each one satisfying more requirements than the last. However, after each iteration is completed, there is another design phase where the knowledge gained from the development of the previous version is used in designing the next one. This means that theses versions and their requirements are not set in stone and may change significantly as each version is completed. Some features with a lower priority may also be excluded if time becomes an issue.

3.3.1 - Version 0/Prototype:

The goal for this basic prototype is to connect all the layers of the architecture so that a user can complete the simple action of entering a short string of text through a web app and receiving a binary classification from the machine learning classifier. In this iteration, the text entered by the user will be classified only by subjectivity.
Features to be implemented for this version:

Functional Requirements	
Requirement:	Description:
Training Data	The system must have access to a large set of labelled training and evaluation data to train the model on.
Text Parsing	The system should be able to implement text-parser functions that will separate the full text into sentences, words and n-grams.
Feature Engineering	The system should provide text to feature functions which can take relevant text characteristics and generate feature vectors.

Subjectivity Classifier	The system must include a well-trained text classifier which can classify text based on subjectivity. This is a key feature of the system.
Web Application	The system must provide a server-based web application for the user to interact with.
User Interface Input Text	The website must provide functionality for a user to input text into a textbox so that the text can be used as the test data in the classification model.
User Input Result	The system should return the user a classification of the sentiment of their inputted text.
Non-Functional Requirements	
Requirement:	Description:
Usability	The system should be easy to navigate and use. A user should be able to get a prediction on the text they've entered in just one click. The user interface for the visualisation screen should be just as simple.

3.3.2 - Version 1:

Whereas in the prototype the goal was for the model to be able to classify a sentence as a proof of concept, in the next iteration, the user should be able to input longer pieces of text and receive an accurate prediction. This will require extensive parameter optimization of the algorithm.
In addition, this version should also classify the text entered by the user by its polarity.

Features to be implemented for this version:

Requirement:	Description:
Polarity Classifier	The System must include a model which can classify text based on polarity. This is a key feature of the system.
User input Error Handling	The system should ensure only valid text data is accepted by the user interface. Relevant exception messages should be shown.

Requirement:	Description:
Reliability	Users should always be able to access the web application running on the server.

3.3.3 - Version 2:

The next iteration should also return a prediction for strength of polarity for the text that the user entered. In addition to returning these predictions, the system should provide information on the features of the text which influenced the predictions made by the machine learning models. The user input field should also be changed so that it provides an option for the user to submit a text file instead of entering text.

Features to be implemented for this version:

Requirement:	Description:
Strength of Polarity Classifier	The System must include a model which can assign a value for the strength of polarity. This is a key feature of the system.
User Interface Input Text File	The website must allow a user to instead upload a text file to the site instead of typing their text.

Requirement:	Description:
Performance	Calculation time and response time should be as low as possible. Server capacity should be great enough to handle multiple sessions at the same time.
Presentation	The web application should be visually pleasing and follow basis user interface design principals

3.3.4 - Version 3/final version:

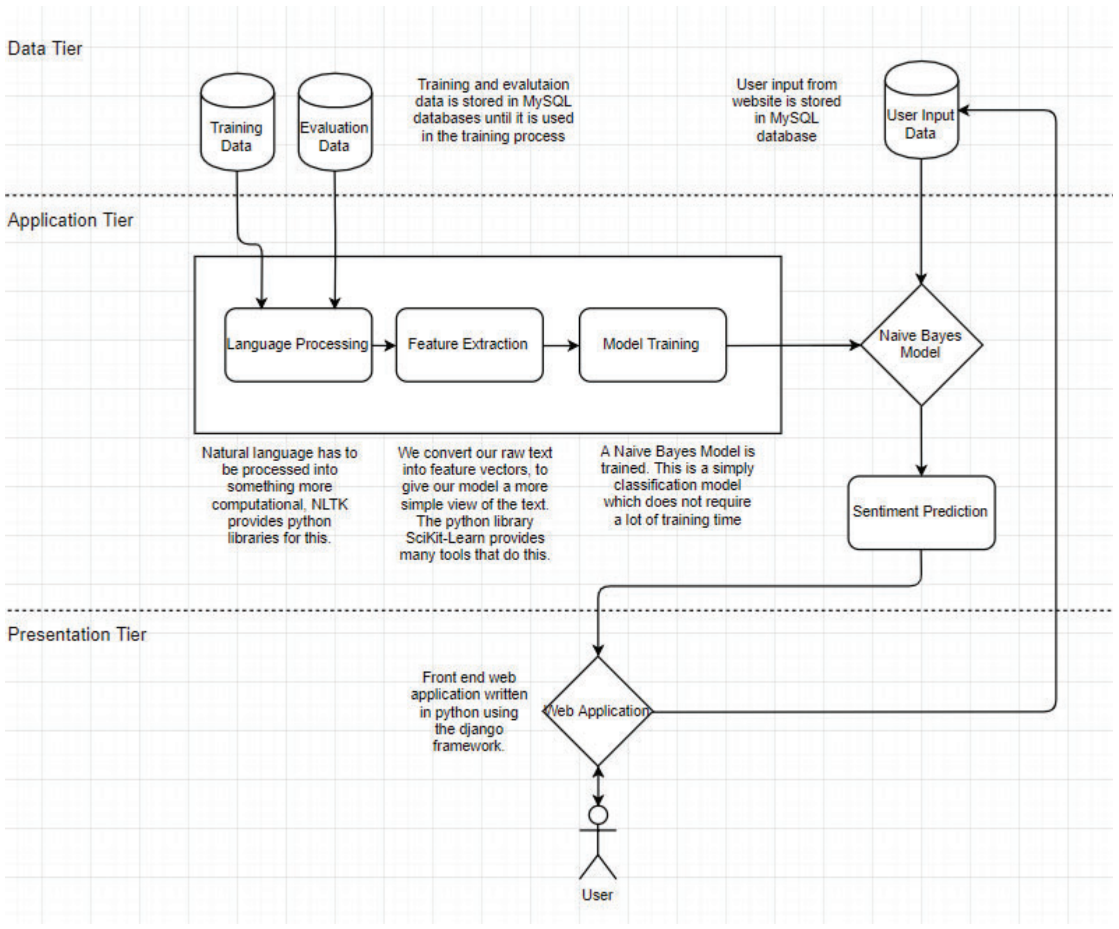
In this final version of the system, the analysis of the parliamentary data will be performed and visualised in the web application. The visualisations should be dynamic and allow for the user to change what information is being displayed.

Features to be implemented for this version:

Requirement:	Description:
Parliamentary Data Acquisition	The system needs to extract a large collection of parliamentary debate transcripts to be classified later.
Parliamentary Data Parsing	The debate transcripts must be parsed into some readable form and separated into individual statements and store them in a database.
User Input Analysis	In addition to a classification, the user should receive some information on the features of their text which influenced the model's prediction.
Parliamentary Data Classification	The system should classify the cleaned parliamentary debate statements by their sentiment and store the results in a database
Data Visualisation	The web application should pull the results of the parliamentary data classification from the database and visualise them.
Visualisation User Control	The user of the website should be able to change how much data is shown and how it is visualised using a control panel featuring buttons and sliders.

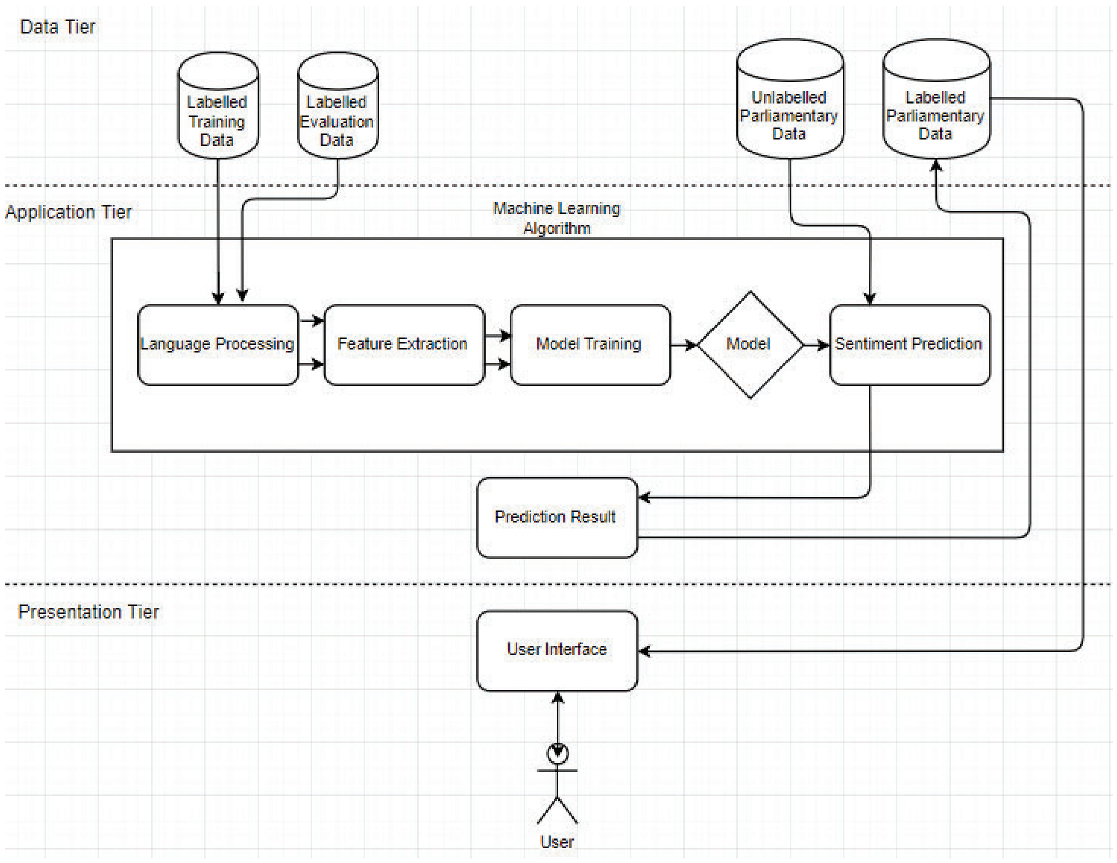
3.4 – System Architecture Overview

Figure 7 – Initial Proposed Architecture 1


Data Tier
Training Data
Evaluation Data
Training and evalutaion data is stored in MySQL databases until it is used in the training process
User input from website is stored in MySQL database
User Input Data
Application Tier
Language Processing
Feature Extraction
Model Training
Naive Bayes Model
Natural language has to be processed into something more computational. NLTK provides python libraries for this.
We convert our raw text into feature vectors, to give our model a more simple view of the text. The python library SciKit-Learn provides many tools that do this.
A Naive Bayes Model is trained. This is a simply classification model which does not require a lot of training time
Sentiment Prediction
Presentation Tier
Front end web application written in python using the django framework.
Web Application
User

Here is the initial outline of the architecture for the user text classification system. The system is represented as a three-tiered architecture in which all data is stored in MySQL databases to that it can be used either in training the model or as input data in order to receive a sentiment prediction. These databases make up the data tier
The middle tier is an application layer containing all the python code used to read data from the database, process the data for training purposes, build the Machine Learning model and finally train the model using the processed data.
The presentation tier consists of the web application used by the user to interact with the system by entering text and receiving analysis.

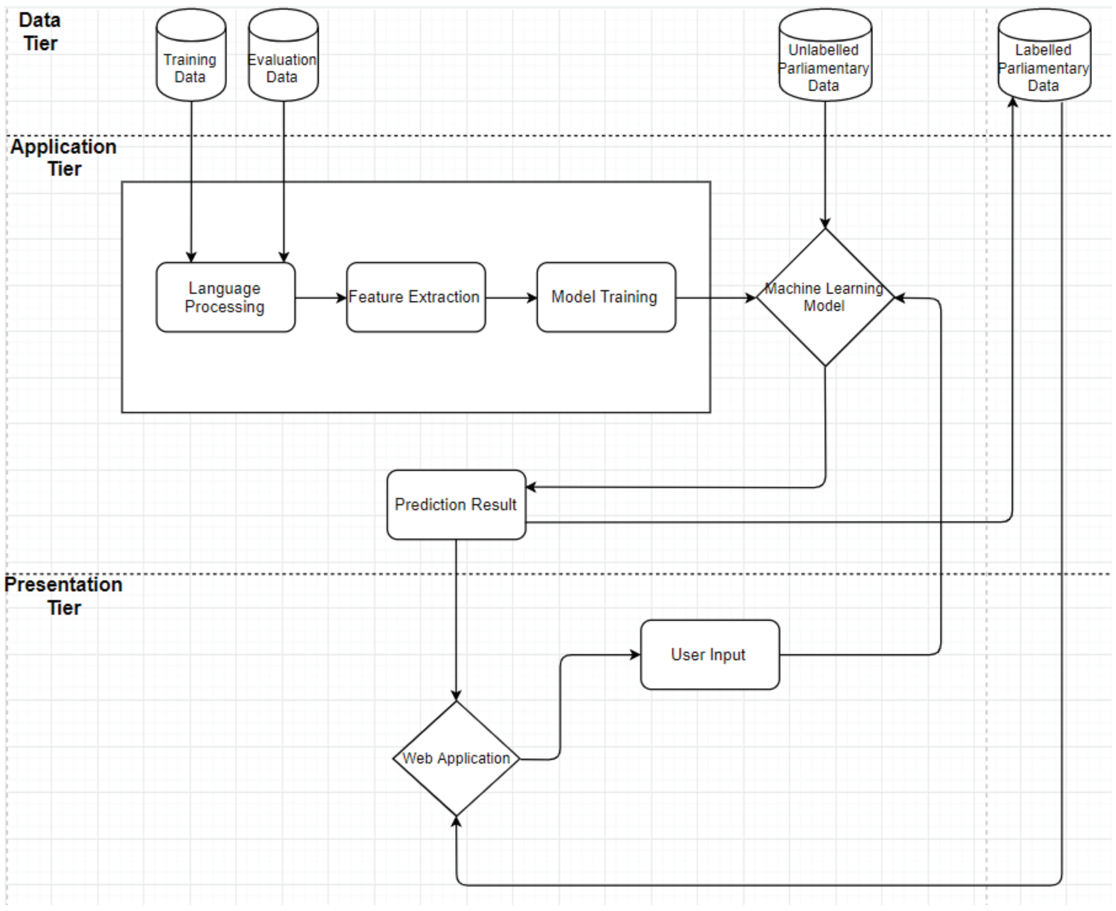
Figure 8 – Initial Proposed Architecture 2



The initial representation of the architecture for the data analysis and visualisation functionality showing how the parliamentary data is classified by the machine learning model and the results of this classification are stored in a separate database in the data tier.

The front-end user application does not perform this classification of the parliamentary data, it merely fetches the results from the database in order to display them in the presentation tier.

Figure 9 – System Architectural Diagram



The complete architecture of the finished version of the system outlining the process of both user text classification and analysis and visualisation of historical data.

4 - Implementation

4.2 –Development Components

The development of the system can be broken down into three separate key components:

- Data preparation

  In order to perform sentiment analysis on statements made in parliament, a large database of statements must be compiled. This is done by acquiring the raw transcripts and using a parsing api to separate and store each individual statement as its own instance.

- Machine Learning Model development

  The key features of the system are the machine learning classification models that enable the user to see a sentiment prediction for their own text as well as performing the analysis of the historical data.
  The training of these classifiers relies on obtaining a good dataset of text data labelled by the sentiment expressed within it.
  The training data must be cleaned and fed into a feature extractor which transforms the text into a feature vector. These feature vectors are used in combination with their corresponding label (e.g. subjective or object) to generate the classification models.

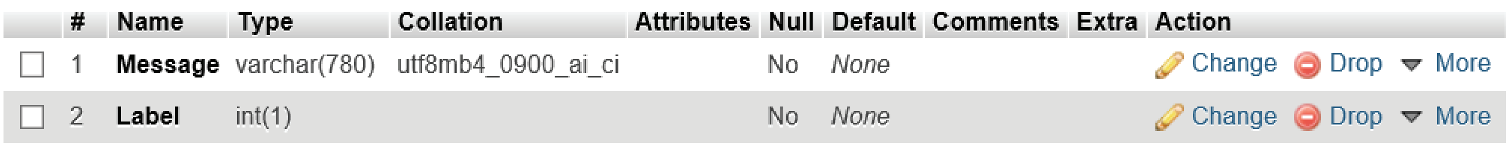
- Web application development

  The final user application is a web application developed using html, CSS, and JavaScript all within the Django framework. Although this is perhaps the most straightforward of the three key components, the application is how the user will interact with and view the previously mentioned sentiment analysis functionality, so it is important that the user interface is well formed and appealing.
  The web application visualises the results of the analysis of parliamentary data and provides the mechanism for the user to enter and view the analysis of their own text.

4.3 – Training Data

Training Data for Subjectivity of text was very easy to find. Of all the possible publicly available datasets, a collection of Online movie reviews labelled by their subjectivity compiled at Cornell University was chosen [13]. The dataset was extracted in the form of two text files containing 5000 subjective and 5000 objective statements respectively. These files were converted into csv format with one column containing the text and another containing either a 0 or 1 indicating whether the text has been labelled as objective or subjective.
This CSV file was then imported into a MySQL database where it would later be accessed for training purposes. See the Structure of the table below.

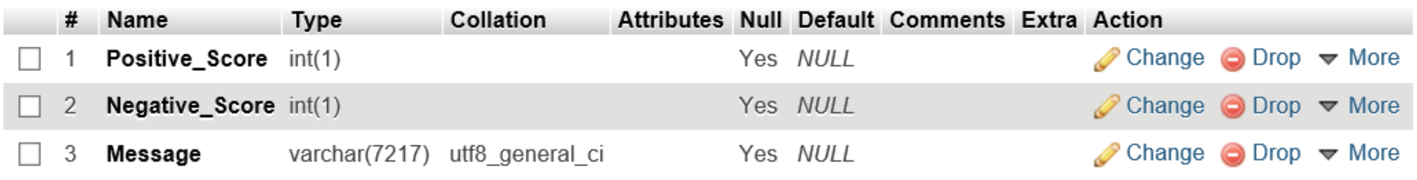
Figure 10 – Subjectivity Database



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**Message**	varchar(780)	utf8mb4_0900_ai_ci		No	*None*			Change Drop More
2	**Label**	int(1)			No	*None*			Change Drop More

Training data for the polarity of text was much harder to come by, there were many datasets available that provided binary classification, but this system would require more classifications than simply negative or positive.
The dataset that was chosen was taken from SentiStrength [19], a sentiment analysis tool similar to the one being developed in this project. It consists of sentences taken from six different popular websites each one each given a score on a 1-5 for both positive and negative sentiment.
Over 10,000 of these pieces of text were transferred into csv format and imported into a MySQL similar to the previous dataset. The idea behind using this dataset was that an aggregate polarity score could be obtained by subtracting the level of negative sentiment from the level of positive sentiment. Meaning that the polarity classifier could classify statements in the range of -4 to +4.

Figure 11 – Polarity Database

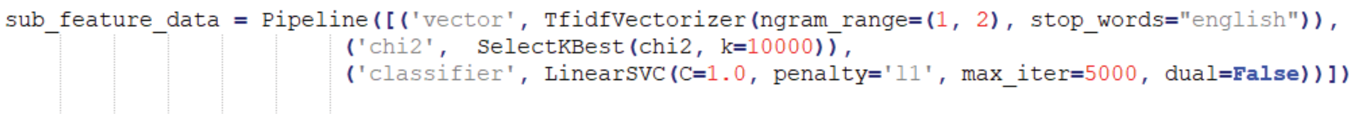


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**Positive_Score**	int(1)			Yes	*NULL*			Change Drop More
2	**Negative_Score**	int(1)			Yes	*NULL*			Change Drop More
3	**Message**	varchar(7217)	utf8_general_ci		Yes	*NULL*			Change Drop More

4.4 – Machine Learning Models

The preparation and storage of suitable training sets enabled the development of the required classification models to begin.
The data was loaded from the MySQL database using the mysql.connector to execute a simple select command and pandas dataframes to store the training data.

In order to perform the necessary classifications, a total of three models must be implanted, one for subjectivity, one for negative polarity and one for positive polarity. The models for positive and negative polarity are developed totally separately in parallel, the combined weight polarity score is a result of the combination of both classifications.
As part of the data cleaning process, a predefined NLTK corpus of stop words and stem words were removed from the texts before the data was split into training and evaluation sets.



```
sub_feature_data = Pipeline([('vector', TfidfVectorizer(ngram_range=(1, 2), stop_words="english")),
                             ('chi2',  SelectKBest(chi2, k=10000)),
                             ('classifier', LinearSVC(C=1.0, penalty='l1', max_iter=5000, dual=False))])
```

Above shows, the syntax used to vectorize the texts as well as parameterize the machine learning algorithm. Bigrams were used with a term frequency–inverse document frequency vectorizer to generate the feature data to be fed to a linear support vector machine.

Several different Algorithms were tested for each different model, either because they were not suitable for the data, or due to errors in preparing and parameterizing the algorithms, only the LinearSVC and the NaiveBayes produced good results. The LinearSVC in particular seemed very effective and was ultimately used for all three classifiers and with very similar parameters.

Although designed for Binary classification, the SK-Learn SVM uses the one vs rest [13] method by default to tackle multiclass classification problems.

The subjectivity classifier achieved an accuracy rating of ~0.70.

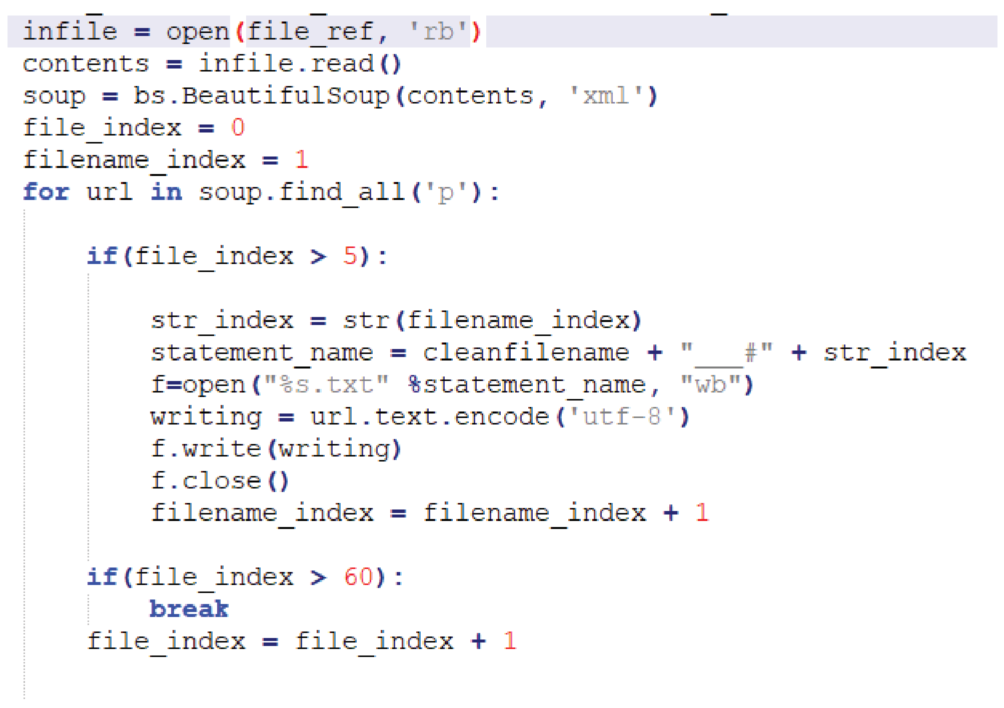
The negative polarity model achieved an accuracy score of ~0.49 while the positive classifier performed slightly worse with a score of ~0.45. Of course, these models were performing multi-class classification, so a lower accuracy score is to be expected.

4.5 – Parliamentary Data Preparation

In order to classify statements made in debates, each individual statement must first be separated from the debate transcript. After the entire UK Hansard archive was acquired in xml format [20]. The files had to converted from their original xml to something more readable before they can be parsed into individual statements for later classification.

Previous research identified multiple Java and Python APIs that can be used to extract plaintext from xml files.

To parse these xml files, the python package BeautifulSoup was used. A statement parser was written to load the directory containing the xml debates into a list, then loop through that list calling BeautifulSoup to parse the current xml debate to create a corresponding data structure in memory. BeautifulSoup then searches this data structure for the xml tag indicating someone is speaking and writes the contents of that tag to a new file.



```
infile = open(file_ref, 'rb')
contents = infile.read()
soup = bs.BeautifulSoup(contents, 'xml')
file_index = 0
filename_index = 1
for url in soup.find_all('p'):

    if(file_index > 5):

        str_index = str(filename_index)
        statement_name = cleanfilename + "___#" + str_index
        f=open("%s.txt" %statement_name, "wb")
        writing = url.text.encode('utf-8')
        f.write(writing)
        f.close()
        filename_index = filename_index + 1

    if(file_index > 60):
        break
    file_index = file_index + 1
```

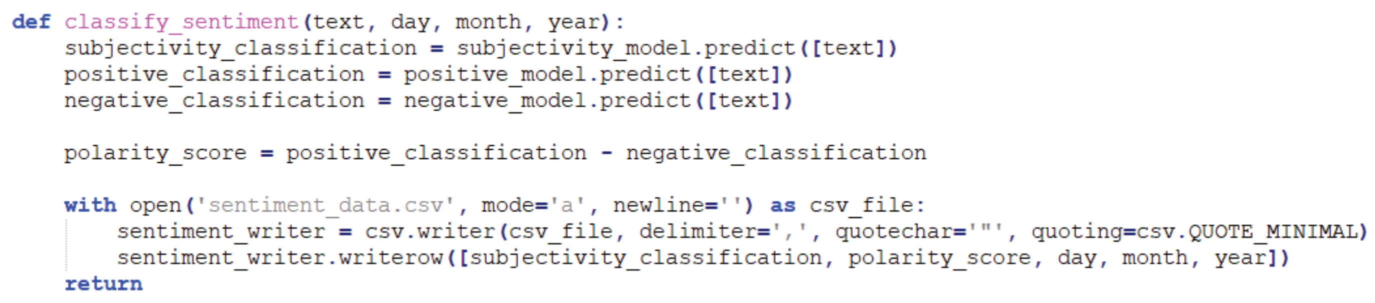
The first ‘p’ tags are ignored since they often did not actually contain in order to limit the set of statements to under 1,000,000, a limit was set for how many statements could be extracted from a single debate.

Given the large number of files involved in this process, this program was required to run for over an hour in order to parse each statement. Keeping almost a million text files in one directory proved to cause a lot of technical problems.

4.6 – Parliamentary Data Classification

After the debate statements had successfully been parsed and collected, each statement needed to be assigned a classification for subjectivity and polarity. Given the number of files that had to be parsed, an automated script was necessary. To implement this, a python program was written containing the three classifiers developed previously

The program worked by loading every statement into a list then looping through the list and for each file calling a method that would classify the text of the statement and



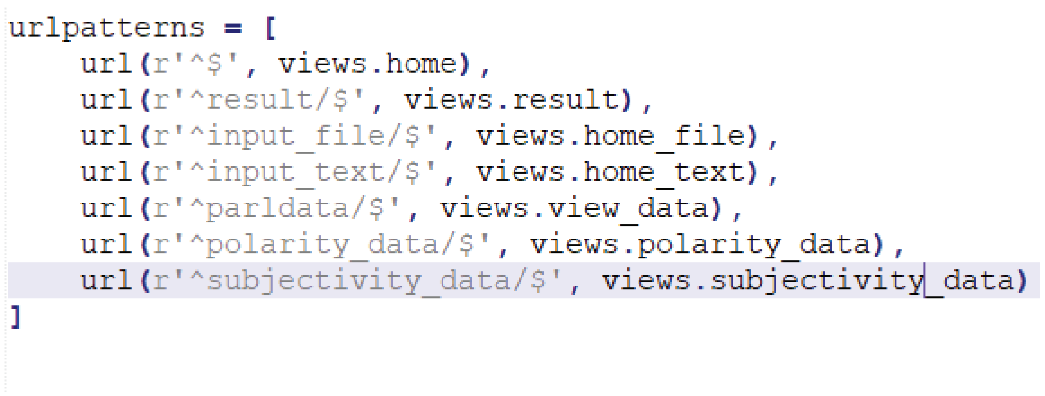
```
def classify_sentiment(text, day, month, year):
    subjectivity_classification = subjectivity_model.predict([text])
    positive_classification = positive_model.predict([text])
    negative_classification = negative_model.predict([text])

    polarity_score = positive_classification - negative_classification

    with open('sentiment_data.csv', mode='a', newline='') as csv_file:
        sentiment_writer = csv.writer(csv_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        sentiment_writer.writerow([subjectivity_classification, polarity_score, day, month, year])
    return
```

store the results in a csv file along with a reference to the date the statement was made.

Above is the method that performs the classification. Some String manipulation was required obtain values for the date from the filenames.

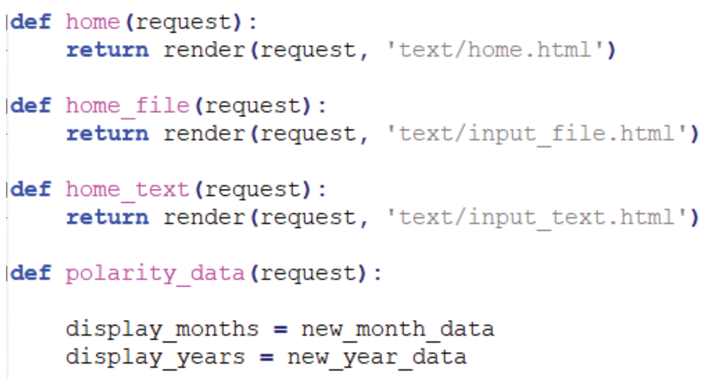
4.7 – Senti-Text Web Application

As Django is a full stack development framework, it was used to create the entirety of the web application. After initializing a new project, much of the boilerplate code necessary to run a web application is provided for you.
The two most important of files in a Django project are urls.py and views.py. Urls.py is used to organize the all url calls within a project. Any method in views.py must be registered in urls.py first.



```
urlpatterns = [
    url(r'^$', views.home),
    url(r'^result/$', views.result),
    url(r'^input_file/$', views.home_file),
    url(r'^input_text/$', views.home_text),
    url(r'^parldata/$', views.view_data),
    url(r'^polarity_data/$', views.polarity_data),
    url(r'^subjectivity_data/$', views.subjectivity_data)
]
```

Above is a look at all the url calls possible in my application along with corresponding method in views.py to be called.
For each screen in my web application, there is a views method that returns the html to be executed on that page. In other words, the views.py file is responsible for handling user action and returning the appropriate response.



```
def home(request):
    return render(request, 'text/home.html')

def home_file(request):
    return render(request, 'text/input_file.html')

def home_text(request):
    return render(request, 'text/input_text.html')

def polarity_data(request):

    display_months = new_month_data
    display_years = new_year_data
```

Here we see the basic operation of the views.py file, rendering basic html files based on url calls, every html file in the system uses the same template and styling in order to maintain a consistent user interface.

All three classification models are present in views.py as well as the results of the parliamentary data analysis. The data is prepared, and the models are trained as soon as the server starts. User input data is handled by html forms which pass their values to methods in views where the data is fed to the prediction models prepared earlier.

Similarly, html form data is passed to view methods and used to form the parameters for the parliamentary data to be visualised using bokeh.

```
if request.method == 'POST':
    start = request.POST['start']
    end = request.POST['end']
    scale = request.POST['scale']

start_year = int(start)
end_year = int(end)

if scale == 'months':
    for index, row in display_months.iterrows():
        if row['year'] < start_year or row['year'] > end_year:
            display_months.drop(index, inplace=True)

    plot = figure(title = 'Polarity of opinion expressed in the British Parliament' , x_axis_label= 'Time', y_ax

    x = display_months['date'].tolist()
    y = display_months['polarity'].tolist()

    plot.line( x, y, line_width = 2 )

    script, div = components(plot)

    return render_to_response('text/polarity_data.html', {'script' : script, 'div' : div})
else:
```

This method returns the html file to be displayed as well as a script for displaying the visualised data. Bokeh makes it very simply to represent data dynamically in graphs. Below is the result of one of these queries.

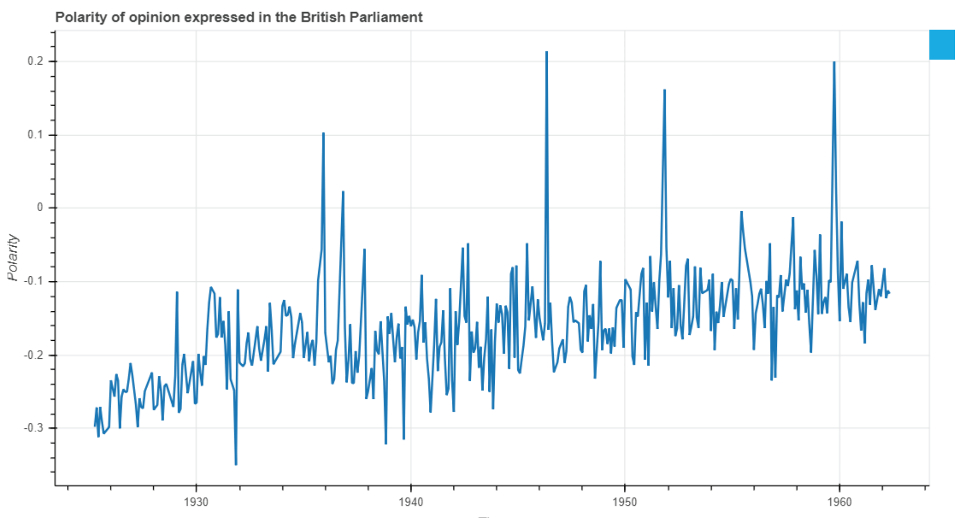


Figure 12 – Parliamentary Data Visualisation

4.8 –Source Code Layout

The Layout of my source code follows the Django framework closely. Most of complexity is in the views.py file of the web application which controls what templates are displayed in the application at all times.

Code for Parsing and classifying the parliamentary data exists outside of the web application it its own dedicated folder.
Below is an outline of the source code that makes up this system. A more detailed make file will provide a better understanding.

Figure 13 – Source Code Layout

```
-FYP
        -textProcessing
                Statement_Parser.py
                Classifier.py

        -sentiment
                -sentiment
                        settings.py
                        urls.py
                -text
                        -templates
                                Data.html
                                Result.html
                                Header.html
                                Home.html
                                Input_file.html
                                Input_text.html
                                Polarity_data.html
                                Subjectivity_data.html
                                Template.html
                        Views.py
                        Urls.py
                        Apps.py
                        Models.py
                        Sentiment_data.csv
```

4.9 –External APIs

Sci-Kit-Learn

Probably the most essential external API for this project, sk-learn provided the methods used for implementing machine learning algorithms in Python for use in both the web application and the parliamentary data classifier.

Bokeh

Bokeh is a visualisation library that made it really easy to embedded visualisations of the parliamentary data analysis within the web application.

Django

Django provided the framework for the web application. It also provided almost all of the external code used in this project. Although the internal logic of the application and the presentation were all implemented as part of development, Django still provided the skeletal structure that the project was built around.

5 - System Validation

As development will take place in iterations, each version will have to be tested thoroughly before the next iteration can begin. Each iteration will have clearly defined features that it requires.

5.1 – Ad Hoc Testing

Outside of these scheduled testing phases, an ad-hoc testing approach will be taken to most of my system validation.
Ad-hoc testing is the most widely used testing method used in the validation of software. Ad-hoc testing is the running of code by the developer after it is written to

verify that the desired behaviour occurs. These tests are carried out by the developer themselves manually which can be more time consuming than automated methods.

Despite this, it is a very straightforward approach since the tests are performed by someone with knowledge and understanding of the system. Ad-hoc tests were used extensively during the development of the web application by performing actions in the application to test if the intended outcome occurred such as the prediction of the user's text being returned correctly. This more informal approach to testing requires less planning and documentation which was beneficial as it was allowed more focus on the technical side of the development process.

5.2 – Unit Testing

Unit testing is the testing of code in its smallest possible units. Typically, these units are functions or classes. The developer carrying out the test is not concerned with the actual code in the unit being test, merely that it produces the desired outputs given the inputs provided. Like Ad-hoc testing, unit testing is carried out by a developer with knowledge of the system.
One advantage of thinking of code in terms of small units of functionality with inputs and outputs is that it helps to increase modularity and decrease interdependencies between classes. Since there was only one developer involved in the implementation of this project, unit testing was a very applicable testing methodology. Given that several iterations of the web applications were implemented, unit testing identified much of the bugs in the application very early in development

5.3 – Initial Test Cases

During the design phase, the following list of high-level test cases was compiled which served as rough guideline for the ad-hoc testing being carried out throughout development

Feature	Test Description	Priority
Algorithm can access training and Validation data sets in MySQL database.	Use a python MySQL connector to reference entries in the database and ensure the data is returned.	High
System must be able to parse text into sentences, words and n-grams.	SK-Learn and NLTK libraries are used to parse text, compare the data before and after these functions are applied to ensure they have worked.	High
Feature vectors must be generated by python functions using scikit-learn libraries.	Features are extracted using SK-Learn functions, the appropriateness of these features can be tested by adding and removing which ones are included in model training.	High
Machine Learning model can classify text by subjectivity.	This is the key feature of the system, can be tested by inputting test data through the web application and awaiting subjectivity prediction	High
Machine Learning model can classify text by polarity.	This is the key feature of the system, can be tested by inputting test data through the web application and awaiting polarity prediction.	High
Machine Learning model can classify text by strength of polarity.	This is the key feature of the system, can be tested by inputting test data through the web application and awaiting strength of polarity prediction.	High
Model can make accurate predictions.	Use validation dataset to evaluate the accuracy of the model.	High
Web Application can access machine learning model	Enter text, click ‘submit’ and see if some analysis is returned.	High

User can input text into website.	Enter some text into the textbox and click 'submit', Should be taken to another page to view result.	High
User can upload text file into website.	Click the 'upload' button and choose a text file, Should be taken to another page to view result.	Medium
User input error handling.	Input invalid data in the textbox and upload invalid file types to see if relevant error messages are returned.	Medium
User receives sentiment prediction of their text.	Enter text, click 'submit' and see if some analysis is returned.	High
Parliamentary results are visualised.	Click on 'Parliamentary Debate Analysis' to see if the data visualisation is displayed	High
User can specify parliamentary data to be visualised.	Use the control panel to alter the parameters of the data displayed and see if the visualisation is updated.	Medium
Server can handle multiple sessions at once	Perform server load testing by connecting to the web app from several clients at once to ensure performance is not affected	Medium

5.4 – Web Application Test Plan.

A test plan was used to test each version of the web application. The following are the results for the final round of testing carried out on the finished version of the application.

Figure 14 – Test Plan

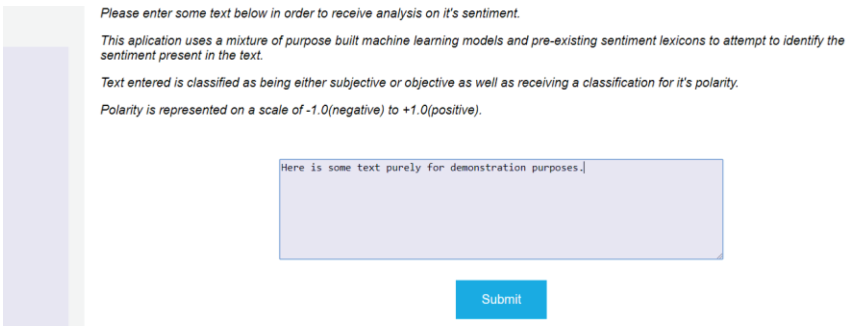
Test Number	Test Description	Result
1	Does the application run in localhost?	Pass
2	Does the application connect and extract training data from the MySQL database?	Pass
3	Does the application clean and split the training data?	Pass
4	Does the application vectorize the training data?	Pass
5	Are classifier models generated using feature data?	Pass
6	Can models give predictions?	Pass
7	Does the application load historical sentiment data from csv file?	Pass
8	Is the historical data transformed into desired format in padas dataframe?	Pass
9	Can the input user text screen be accessed through the navigation panel?	Pass
10	Can the input text file screen be accessed through the navigation panel?	Pass
11	Can the parliamentary data screen be accessed through the navigation panel?	Pass
12	Can the home screen be accessed through the navigation panel?	Pass
13	Can the user text prediction result screen be accessed using the input text form?	Pass
14	Can the user text file prediction result screen be accessed using the input file form?	Pass
15	Do the classification models generate a prediction of user input?	Pass
16	Are particularly negative or positive words in user text identified?	Fail
17	Are particularly subjective or objective words in user text identified?	Fail
18	Is the user text prediction displayed in the user text prediction screen?	Pass
19	Are particularly negative or positive words in user text displayed in the user text prediction screen?	Fail
20	Are particularly subjective or objective words in user text displayed in the user text prediction screen?	Fail
21	Is the user text file prediction displayed in the user file prediction screen?	Pass
22	Can the historical polarity results screen be accessed through data the visualisation form?	Pass
23	Can the historical subjectivity results screen be accessed through the data visualisation form?	Pass
24	Is the historical polarity data displayed in a graph in the polarity results screen?	Pass
25	Is the historical subjectivity data displayed in a graph in the subjectivity results screen?	Pass
26	Is the historical data used to generate polarity graph?	Pass
27	Is the historical data used to generate subjectivity graph?	Pass
28	Does the historical polarity visualisation form results change parameters of data visualisations?	Pass
29	Does the historical subjectivity visualisation form results change parameters of data visualisations?	Pass

5.5 – Demonstration

The demonstration of the system is fairly straightforward. It shouldn't take much longer than a few minutes to run through every use-case possible within the web application, inputting text and viewing the results, uploading a file and viewing the results and then having a look at the parliamentary analysis results and playing around with some of the different parameters for visualisation. The functions and capabilities of the application should be obvious to most people viewing its demonstration

Screen demonstrating the functionality to enter some original text.

Figure 15 – Text Input Screen



Please enter some text below in order to receive analysis on it's sentiment.

This aplication uses a mixture of purpose built machine learning models and pre-existing sentiment lexicons to attempt to identify the sentiment present in the text.

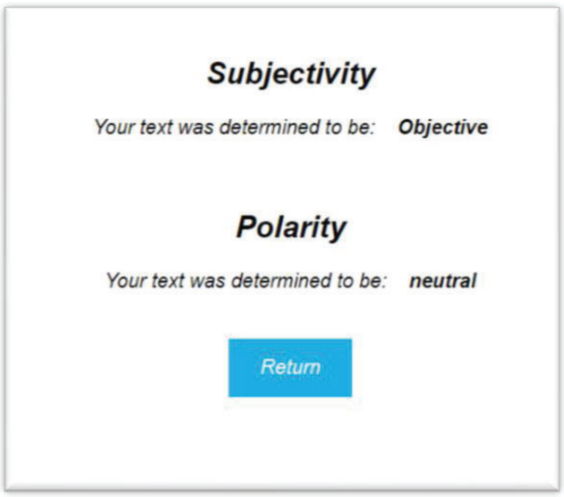
Text entered is classified as being either subjective or objective as well as receiving a classification for it's polarity.

Polarity is represented on a scale of -1.0(negative) to +1.0(positive).

```
Here is some text purely for demonstration purposes.
```

Submitting either written text or a text file returns the same result, a simple results screen. A good demonstration of this system would show off the classification of a wide variety of language to try and illicit interesting results and showcase the workings of the classification models.

Figure 16 – Results Screen



Subjectivity

Your text was determined to be: **Objective**

Polarity

Your text was determined to be: **neutral**

Results of the user text classification are displayed on a very simple results screen.

6 – Critical Evaluation

6.1 – Initial Project Plan

The original allocated ample time to extracting and parsing the relevant parliamentary data, while scheduling the implementation phase to be carried out in parallel to more research being conducted. This led to the methods used in the implementation changing drastically as new technologies were learned about. There should have been more research in the beginning stages of the project to avoid this issue.

6.2 – Changes from initial approach

6.2.1 – Project Plan

Later, a simplified and revised plan was drawn up to help focus development activities with the aim of getting a working version of the system completed and documented before the deadline.
This plan focusses solely on self-imposed deadlines for different development iterations. Although these deadlines were not strictly met and adhered to, they provided a guide to help keep the development on schedule.

6.2.2 – Parliamentary Data

Originally the plan was to analyse and visualize both UK parliamentary debates and Dail debates, during development it was decided that only UK debates would be necessary. It was found that the most interesting results came from comparing levels of subjectivity and polarity between different time periods rather than between different parliamentary systems.

6.2.3 – Strength of Polarity Prediction

In the initial plan for the system, each piece of text being classified was to be given a numerical score between -1 and 1 to represent the strength of polarity of the opinion expressed in the text. Instead, a polarity rating was found by combining a positive sentiment rating with a negative sentiment rating, resulting in 9 possible polarity classifications.

6.2.3 – Analysis Results Storage

In the initial proposed architecture, the results of the analysis of the parliamentary data were stored in a MySQL database. However, when the script used for classifying each individual parsed statement and storing the results was being developed, it was found that writing the results to a csv file was much faster adding them to a MySQL table. Since the script would have to loop through ~900,000 statements, this slight increase in efficiency resulted in the total required runtime of the script being reduced by several hours.

6.3 – Project versions

6.3.1 – Prototype

The purpose of the prototype was to be able to demonstrate how the system can classify a segment of text and return that classification to the user. This was intended to show how each 'layer' of the system links together as a proof of concept without having to implement too many features. The prototype did not have a user-friendly UI and only featured one classification model for subjectivity. This system was adequate for a prototype but unfortunately there was an error present during its demonstration which made it unable to return the results of the subjectivity classification.

6.3.2 – Version 1

In the next version, a model for predicting polarity was implemented. The model returned only a binary classification of polarity. When these iterations were being planned out, I thought that this binary polarity model could be improved on later to provide a polarity score but in reality, it needed to simply be replaced by the superior strength of polarity classifier.

6.3.3 – Version 2

Version 2 called for the implementation of a non-binary classifier for polarity. The development of this version represents the most disorganised and misguided I was during the entire project. After finding that the datasets I had previously though were suitable to train this model were in fact un-usual and difficulty identifying a replacement set of training data, I experimented looked into using a statistical approach rather than machine learning before coming across the SentiWord[13] dataset and training the strength of polarity model with it.

6.3.4 – Version 2

The final version of the system was the first to include the visualisation of the analysis of parliamentary data. Having discovered the Bokeh package for embedded visualisations in the web applications with python, the data analysis aspect to this final version was not so challenging. However due to time constraints and poor organization, I was not able to implement feedback to the user explaining the features of their text that influenced its classification.

6.4 – Historical Parliamentary Data Analysis Results

The results of the sentiment classification of the Hansard archives were very interesting. It was found that on average a slightly negative sentiment was expressed with a few rare outliers of periods where there were more positive opinions being expressed in debates.
The results for subjectivity indicated that the vast majority of statements processed were objective, leading to a very low average subjectivity rating. However, the results for subjectivity were still less uniform than expected.
Studies such as this which monitor trends of political discourse throughout the years are taken seriously by the parliamentary bodies themselves and a lot of effort is put into compiling these studies.[20]

6.5 – Outcome

Although much of what I set out to accomplish was achieved, the finished version of the web application could have been more thorough. As mentioned, not all of the desired features could be implemented in a timely manner and the user interface doesn’t look as appealing as it could have had more time been devoted to it.
The key feature of the system was the machine learning sentiment classification models, I feel like this was a mixed bag, the polarity classifier performs well at classifying user text while the subjectivity classifier often provides false positives for subjectivity despite the numerical accuracy score it achieves based on evaluation data.

Despite these minor disappointments, a useful tool was still produced as well as a lot of interesting results of the analysis of parliamentary data. I found the trends in levels of negativity and subjectivity to be fascinating.

7 - Conclusion

7.2 – Future Work

Automated sentiment analysis of political discourse is field where a rapid growth in relevance can be foreseen.
Sentiment is important in studies of news values, public opinion, negative campaigning or political polarization and an explosive expansion of digital textual data and fast progress in automated text analysis provide vast opportunities for innovative social science research.
I would be very interested in working on other projects related to this approach to sentiment analysis. Particularly ones involving classifying opinions in more complicated ways than just subjectivity and polarity. I think the possibility of automating the detection of advanced human emotions or political leaning is fascinating [21].

7.3 – Conclusion

The main objective of the project was to examine how an application can determine the sentiment of different types of text using machine learning. I think this project demonstrated the effectiveness of using automated machine learning techniques to tackle sentiment analysis as a classification problem.
Regardless of outcome, this project was an amazing learning experience for me. It was my first time working on a solo project with a development cycle this long. Having to come up with an idea for a project and follow through on it taught me a lot about the importance of putting in work in the research and design phases of a project.
I feel like I missed an opportunity during those first few months to gain a more solid understanding of the area as well as establish a better understanding of the scope of my own project and what it would entail.

When I take a critical look at my finished project, I can see that it is somewhat lacking in complexity. I don't blame this on a bad development methodology or not having enough time to finish my implementation but simply not laying the foundation for success from the beginning.

8 - Bibliography

[1] Mullen, T. and Collier, N. (2019). *Sentiment analysis using support vector machines with diverse information sources*. [online] Aclweb.org. Available at: https://www.aclweb.org/anthology/W04-3253

[2] Rouse, M. (2019). *What is machine learning (ML)? - Definition from WhatIs.com*. [online] SearchEnterpriseAI. Available at: https://searchenterpriseai.techtarget.com/definition/machine-learning-ML

[3] MonkeyLearn. (2019). *Sentiment Analysis: nearly everything you need to know | MonkeyLearn*. [online] Available at: https://monkeylearn.com/sentiment-analysis/

[4] Delany, S., Sinnot, R. and O'Reilly, N. (2010). The Extent of Clientelism in Irish Politics: Evidence from Classifying Dáil Questions on a LocalNational Dimension. In: *21st Irish Conference on Artificial Intelligence and Cognitive Science*. [online] Galway: Dublin Institute of Technology, pp.4-7. Available at: https://arrow.dit.ie/cgi/viewcontent.cgi?article=1031&context=dmccon.

[5] Esuli, A. and Sebastiani, F. (2007). *SentiWordNet: A High-Coverage Lexical Resource for Opinion Mining*. [online] Pisa, Italy: Kluwer Academic Publishers, p.2. Available at: http://ontotext.fbk.eu/Publications/sentiWN-TR.pdf.

[6] Grijzenhout, S., Marx, M. and Jijkoun, V. (2014). Sentiment Analysis in Parliamentary Proceedings. In: *From Text to Political Positions Text analysis across disciplines*, 1st ed. [online] Amsterdam: John Benjamins Publishing, pp.9-11. Available at: http://politicalmashup.nl/new/uploads/2011/01/effekes.pdf.

[7] Voskoglou, C. (2018). *What is the best programming language for Machine Learning?* [online] Towards Data Science. Available at: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7

[8] Altexsoft.com. (2018). *Open Source Machine Learning Libraries: TensorFlow, Theano, Torch, scikit-learn, Caffe*. [online] Available at: https://www.altexsoft.com/blog/datascience/choosing-an-open-source-machine-learning-framework-tensorflow-theano-torch-scikit-learn-caffe/

[9] Statsoft.com. (2018). *Naive Bayes Classifier*. [online] Available at: http://www.statsoft.com/textbook/naive-bayes-classifier.

[10] Nigam, K., Lafferty, J. and McCallum, A. (1999). *Using Maximum Entropy for Text Classification*. [online] Pittsburgh: Carnegie Mellon University Pittsburgh, pp.1-2. Available at: http://www.kamalnigam.com/papers/maxent-ijcaiws99.pdf.

[11] Bogdanov, V. (2015). *Top 10 sites built with Django Framework*. [online] Linkedin.com. Available at: https://www.linkedin.com/pulse/top-10-sites-built-django-framework-vladimir-bogdanov

.

[12] Docs.oracle.com. (2010). *Using the XML Parser for Java*. [online] Available at: https://docs.oracle.com/cd/B19306_01/appdev.102/b14252/adx_j_parser.htm#CCHGBGIG.

[13] Cs.cornell.edu. (n.d.). *Data*. [online] Available at: http://www.cs.cornell.edu/people/pabo/movie-review-data/.

[14] O'Neil, S. (2017). *Anti-Bullying with Machine Learning Final Year Project Report*. BSc. Dublin Institute of Technology. Available at: https://ditlib.dit.ie/articles/3766210.2961/1.PDF

[15] Hanratty, E. (2017). *Detecting Bot Twitter Accounts using Machine Learning Final Year Project Report*. BSc. Dublin Institute of Technology. Available at: https://ditlib.dit.ie/articles/3766230.2972/1.PDF

[16] Chen, Y. (2017). *Euro Coin Classification Using Image Processing & Machine Learning*. BSc. Dublin Institute of Technology. Available at: https://ditlib.dit.ie/articles/3766245.2983/1.PDF

[17] Joachims, T. (2018). *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. [online] Dortmund, Germany: Universitat Dortmund, pp.2-4. Available at: http://www.cs.cornell.edu/~tj/publications/joachims_98a.pdf

[18] McCallum, A. and Nigam, K. (2001). *3*. [online] Citeseerx.ist.psu.edu. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.9324&rep=rep1&type=pdf.

[19] Sentistrength.wlv.ac.uk. (2018). *SentiStrength - sentiment strength detection in short texts - sentiment analysis, opinion mining*. [online] Available at: http://sentistrength.wlv.ac.uk/

[19] Theyworkforyou.com. (2019). *Index of /pwdata/scrapedxml/debates*. [online] Available at: https://www.theyworkforyou.com/pwdata/scrapedxml/debates/

[20] Coleman, C. (2013). *House of Commons Trends*. [online] London: House of Commons Library. Available at: https://researchbriefings.parliament.uk/ResearchBriefing/Summary/RP13-48#fullreport

[21] Falck, F., Marstaller, J., Stoehr, N., Maucher, S., Ren, J., Thalhammer, A., Rettinger, A. and Studer, R. (2019). *Sentiment Political Compass: A Data-driven Analysis of Online Newspapers regarding Political Orientation*. [online] Blogs.oii.ox.ac.uk. Available at: **http://blogs.oii.ox.ac.uk/policy/wp-content/uploads/sites/77/2018/08/IPP2018_Falck.pdf**

9 - Appendix

9.1 – Installation Guide

In order to run the machine learning algorithms contained in web application, you'll first need to import the MySQL database. This is done using the sentimentDB.sql file in the project folder. This is done by:

1. Open phpMyAdmin
2. Navigate to an empty database
3. Click the Import tab.
4. Select the sentimentDB.sql file representing the database for this application
5. Click Go.

After completing these steps, you should have the working database up and ready to provide training data to the machine learning models.

To run the web application, you'll need to set up a Python development environment, which means installing including Python, pip, and virtualenv.
It will also be necessary to install some key python libraries used in the system such as sklearn, nltk and bokeh.
To install the necessary dependencies on windows, you'll need to run the commands

```
virtualenv env
env\scripts\activate
pip install -r requirements.txt
```

Then run the migrations in the directory containing manage.py

```
python manage.py makemigrations
python manage.py makemigrations sentiment
python manage.py migrate
```

Finally run the server and access the application at *http://localhost:8000*

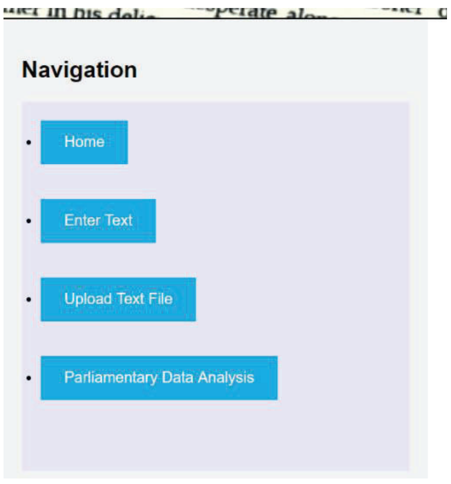
```
Python manage.py runserver
```

9.2 – Senti-Text User Guide

- **Home Screen**

  Upon starting the Senti-Text application you will be greeted by a welcome screen containing some background text.
  You can navigate the site by using the links in the side bar to take you to a different page at any time.

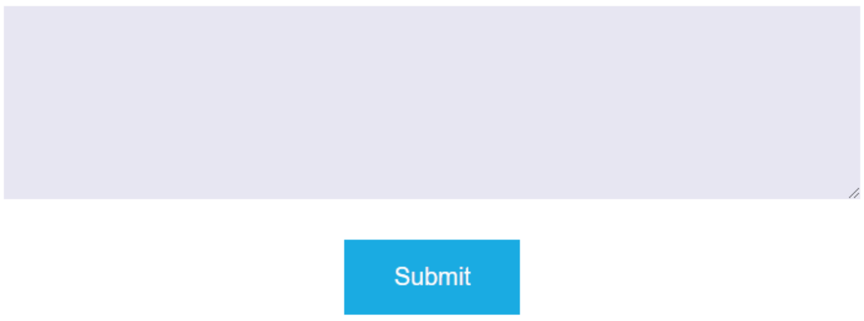
  Figure 17 – Navigation Bar



- **Enter Text Screen**

  To enter your own text and view the classification of its sentiment, select the 'Enter Text' option in the sidebar. You'll be taken to a screen where you can use a text input box to enter the text

  Figure 18 – Text Input Area

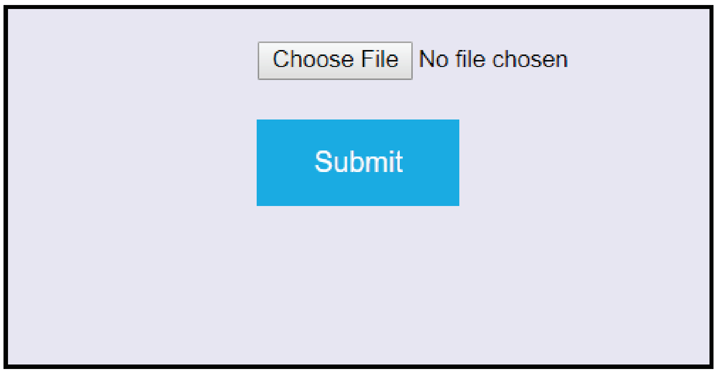


  After you've written some text, use the 'Submit' button below to be taken to a results page where you can view your text's classification.

- **Upload File Screen**

To upload your own text file, select the 'Upload Text File' option on the sidebar. In this screen, instead of typing text input an input area, you'll be prompted to select a file from your local files to upload and pass to the sentiment classifier.

Figure 19 – File Upload Area

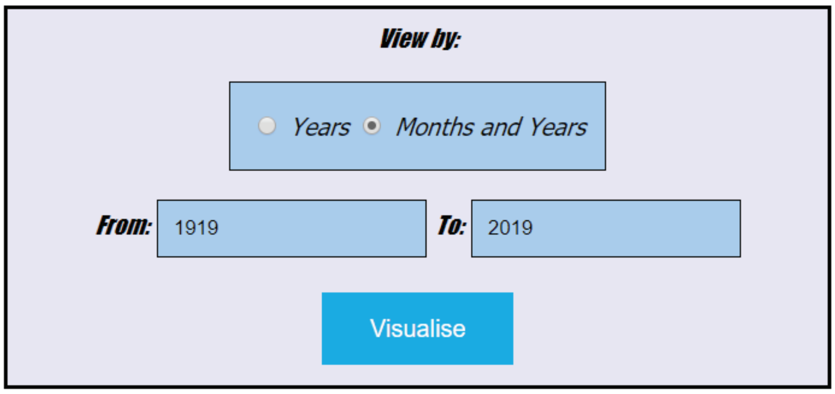


Only .txt file uploads will be accepted.

- **View Historical Data**

  To view the results of the analysis of historical parliamentary data, select 'Parliamentary Data Analysis' from the sidebar. You'll be taken to a screen where you'll be prompted to specify the range of results you'd like to see as well as which sentiment attribute you'd like to see the data categorized as. There are two separate forms, one for subjectivity and one for polarity.

Figure 20 – Parliamentary Data Specification Screen


View by:

Years [✓ Months and Years]

From: 1919 To: 2019

After selecting the date information, click the 'Visualise' button to be taken to a screen displaying the data in the form of a line graph. The X-axis represents time while the Y-axis represents level of either subjectivity or polarity.

9.2 – Table of Figures

Figure 1 - System Overview
Figure 2 - Software Architecture Diagram
Figure 3 - Machine Learning Diagram
Figure 4 - Chose Technologies Table
Figure 5 - Use Case Diagram
Figure 6 - Senti-Text Mock-up
Figure 7 - Initial Proposed Architecture 1
Figure 8 - Initial Proposed Architecture 2
Figure 9 - System Architectural Diagram
Figure 10 - Subjectivity Database
Figure 11 - Polarity Database
Figure 12 - Parliamentary Data Visualisation
Figure 13 - Source Code Layout
Figure 14 - Test Plan
Figure 15 - Text Input Screen
Figure 16 - Results Screen
Figure 17 - Navigation Bar
Figure 18 - Text Input Area
Figure 19 - File Upload Area
Figure 20 - File Upload Area