University of Manchester
School of Computer Science
Project Report 2012

Text Mining Twitter for Software and User Perception

Author: Tariq Patel

Supervisor: Dr. Goran Nenadic

Abstract

Text Mining Twitter for Software
and User Perception

Author: Tariq Patel

Supervisor: Dr. Goran Nenadic

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The success of a piece of software is based largely upon user opinion. Gathering such information is conventionally done through means of surveying groups of users. However, in the days of social media, people generally express their opinions on popular social networks or microblogging sites such as Facebook and Twitter. This means it is now much easier for companies to receive feedback on products they have developed by monitoring these networks.

1.1 Aim and Motivation

Twitter has been at the core of many data mining projects in recent years and this is due to the sheer amount of data produced on a daily basis. Twitter users now post in excess of 340 million tweets every day[@tw12a] and as such Twitter provides a massive corporus for opinion mining and sentiment analysis.

By text mining Twitter posts for software, users are able to discover new tools or programs they have not come across before, as well as see reviews by other users.

Thus, the aim of this project is to develop a system that text mines Twitter posts to find software or software development tools that have been mentioned by its users and to discover the general sentiment of users towards these softwares.

1.2 Challenges

There are many challenges facing Natural Language Processing(NLP)-oriented projects. Table 1.1 shows the key issues to be faced in achieving the main aim.

With the millions of tweets posted every day on Twitter, one can safely presume that many of these will have no relevance to software or any of the other desired information. As such it will be vital to ensure only the most relevant tweets are extracted from Twitter for analysis so as not to waste resources.

Another issue is the world-wide nature of the Internet and microblogging networks like Twitter. This means that several tweets will not be in English and for this reason it would be more difficult to extract features from these tweets. To counter this, it will be necessary to filter tweets not only based on key words but also on their language.

A major issue in NLP research is that of text message shorthand. In a formal document this problem becomes somewhat irrelevant due to proper usage of Standard English. However, when working with the Twitter platform, the service's 140 character limitation on tweets means

	Challenges
1	Finding relevant tweets
2	Non-English tweets
3	Text message shorthand

Table 1.1: Challenges to be faced in this project

	Task	Complexity	Priority
1	Collect and filter tweets by keyword	Simple	High
2	Feature extraction	Complex	High
3	Analyse tweet sentiment	Intermediate	Medium
4	Structure and integrate data	Intermediate	High
5	Visualise data through GUI	Intermediate	Low
6	Evaluate system	Complex	High

Table 1.2: Complexity and priority of project objectives

users are generally more likely to abbreviate their text and this allows for a lot of ambiguity in the context of each word, and variability in how users may say the same thing.

1.3 Objectives

In order to successfully complete a project of this magnitude, the task at hand must be split into smaller steps. These objectives are shown with their complexities and prorities in Table 1.2.

1.3.1 Collect and Filter Tweets by Keyword

Collecting tweets is a core task in this project as all work will be based on tweets stored in a database. Filtering through these is a relatively simple task in that it can be done using Twitter's APIs but there are some complexities in ensuring they are all relevant.

The main idea at this stage is to collect tweets from Twitter based on a set of key words and software names, games, programming languages, or company names stored in a dictionary in order to retrieve relevant, software-related tweets.

1.3.2 Feature Extraction

Feature extracting is the core functionality set out to be achieved in this project. Using rule-based text mining techniques, the aim of this task will be to retrieve up to eight features from every tweet, which are shown in Table 1.3.

These features have been selected in order to find useful information from tweets to be displayed to users. The **software name** is of course vital, in that this discovery is the main purpose of the project. The **version** of this software is important because major changes may have been made over the course of a few releases and so it is necessary to note which release people are referring to. The **company name** is not a major feature, however it may be interesting to know who developed a certain piece of software. It may also be used in a different scenario

	Feature
1	Software name
2	Software version
3	Company or developer
4	Programming language
5	Operating system
6	Price
7	Relevant URLs
8	Tweet sentiment

Table 1.3: Features to be extracted from tweets

where a user of this system wishes to find public sentiment towards a company as opposed to some specific software. The **programming language** feature ideally signifies the language or languages in which the found software was developed in. However, as with the company field, this may be used to find sentiment towards specific programming languages or practices. The **operating system** field works in a similar fashion, in that its expected use is to find the operating systems upon which the found software runs, but it can also be used to find the sentiment towards a specific operating system. **Price** and **URL** extraction are geared towards retrieving slightly for information about the product for the user. The **tweet sentiment** aims to find the general sentiment towards a piece of software, and will be used in the aggregation process in the final stages when trying to establish public perception of the software.

1.3.3 Analyse Tweet Sentiment

Sentiment analysis is another of the more important tasks in this project. This is where tweets are analysed for subjectivity, i.e. whether the tweet is positive, negative or neutral, and this will be used to show the general user perception of each piece of software.

1.3.4 Structure and Integrate Data

The data needs to be structured and aggregated to be able to provide any meaningful output for the user. Without this step, the system is producing no useful information.

1.3.5 Visualise Data Through GUI

Visualising the data is a fairly low priority task in that the system first needs to gather the information. This project centres more around the core back-end development than user experience and as such only a simple user interface is needed in its initial stages.

1.3.6 Evaluate System

The final evaluation of the produced system will be key in determining the success of this project. The system will be evaluated on the basis of the accuracy of retrieved results, the relevance and novelty of information and general usability.

1.4 Report Structure

This report documents the implementation of a text mining system that is set out to achieve the previously stated goals. The remainder of this report has been split into 6 chapters. Chapter 2 details the general background of this project and previous work in the area. Chapter 3 goes into the design of the software implementation including use case analysis, the architecture of the system and the software engineering methodologies used. Chapter 4 details the process of implementing each stage of the project and goes into details of how specific aspects such as the Twitter API integration and feature extraction work. Chapter 5 details the testing methods, results and final outcomes of the project with any meaningful information gained. Chapter 6 provides the general evaluation of the finished project, also outlining the successes and failures of the task at hand. Finally, Chapter 7 details the author's conclusions of the project, with suggestions for further work and a summary of the report.

Chapter 2

Background

This chapter provides an overview of the text mining field along with previous work in the area and all necessary background information required to understand the major tasks involved in this project. This is followed up with an overview of the different APIs provided by Twitter to work with their platform.

2.1 Text Mining

The information available in the world is growing exponentially, and the majority of this information(widely estimated at roughly 80%) is unstructured[Gri08]. This is where text mining comes in, also referred to as Knowledge Discovery from Text(KDT). "Text mining is the process of extracting interesting information and knowledge from unstructured text"[HNP05] and its applications tend to work in two steps, first using an Information Retrieval(IR) application to narrow the search space, and then they extract significant parts of the retrieved texts[Pol06]. This general process usually involves structuring a source text by means of parsing and other linguistic analysis, then finding patterns in this structured data and then interpreting this output.

Text mining is fundamentally different from standard web searching in that web searches rely primarily on information that is already known. However, the goal of text mining is to discover interesting, previously unknown information[GL09]. There is however one key issue introduced by text mining; natural language is used by humans for communication and recording information, while computers are incapable of interpreting natural language. Humans are naturally able to find linguistic patterns in text and understand the semantics of what is being said. Computers, on the other hand, face difficulties in interpreting variations in written text through spelling, colloquialism and also the general context of the text. Nonetheless, computers have what humans do not, that is, computers are much more capable of processing large datasets at very high speeds, particularly in comparison to the human being. Thus, the objective of text mining is to combine the best of these both by creating an application that can retrieve relevant documents and then apply linguistic patterns which may be rule-based, using human-defined rules, or taught by means of machine learning techniques. This project takes the rule-based approach and as such only these techniques will be discussed.

An example of the text mining process can be seen in Figure 2.1.

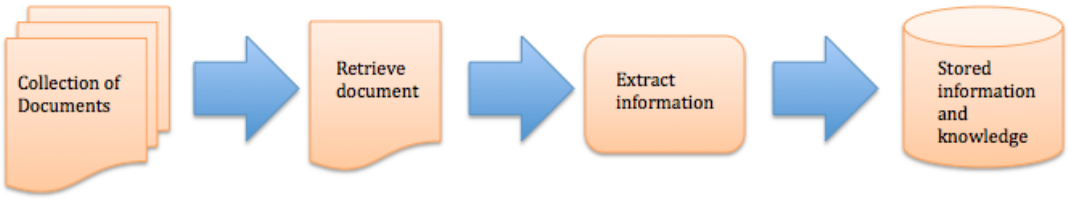

Figure 2.1: The text mining process[GL09]

2.1.1 Information Retrieval(IR)

IR is the process of retrieving textual documents which may contain the answers to questions but do not answer these themselves[Hea99]. Information retrieval is fundamentally a web search working off user queries representing an information need. The process works by searching a collection of documents, and then retrieving those matching a user query depending on relevance. The approach to calculating relevance is dependent upon the actual IR engine itself, generally working on the frequency of specific key terms in each of these documents, and usually assigns a relevance rank to each document. This allows a sorting amongst the results and gives improved results, especially when given a limited number of results.

The IR tasks in this project will mainly be carried out on Twitter's systems, and as such, besides the core concept of IR, its internal specifics are not in the scope of this report.

2.1.2 Natural Language Processing(NLP)

NLP, in the scope of this project, is the process of extracting information from natural language [WH98], that is, any language written or spoken by humans. This involves parsing and processing unstructured text to be able to gain meaningful knowledge from it. Nowadays most natural language processing is done using machine learning techniques, however, in the past implementations were based on large sets of coded 'rules'. These rules are used to define certain linguistic features in the text in order to understand the semantics behind it. NLP is a major field of research at present and also has applications in both information retrieval and information extraction.

There are many methods involved in NLP tasks and some of these will now be further explored.

Tokenisation

Tokenisation is the process of splitting a stream of text into singular words or phrases, otherwise known as tokens. These tokens usually form the basis of further NLP work. While it can be a straightforward process when using Standard English, the definition of a word, from the tokeniser's point of view, can be somewhat ambiguous. This is particularly true when considering the use of apostraphes. Figure 2.2 shows an example of the different ways of tokenising the word *don't*.

These variations can be problematic in terms of the results being output for certain user queries. For example, in the case of these differing tokenisations of *don't*, a user search for the word *don* would return true twice, but should be false in its actual context. The importance of

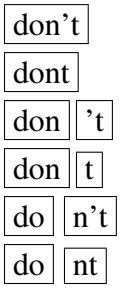

Figure 2.2: The different ways of tokenising the word *don't*

normalisation is highlighted when tokenising tweets because a lot of users do not use apostrophes, either due to ease when typing, or in order to reduce the number of characters being used. Thus, varying spellings of the terms should not be tokenised differently.

Normalisation

Once text has been tokenised, these words may need normalising. Normalisation accounts for the several variations in spelling. For example, if you want to search for *Mozilla Firefox* you would want an IR engine to return not only documents containing the exact query but also those containing terms such as *Firefox*, *firefox* or *mozilla firefox*. Not doing this would obviously yield fewer results, or in the case of information extraction, it may suggest that *Mozilla Firefox* and *mozilla firefox* are two different things. Thus, normalisation is required to successfully map equivalent classes of terms.

Stop Word Filtering

Stop words are very commonly used words like *a*, *and* or *the*. By creating a list of these terms, a *stop list*, a natural language parser can remove these terms from the source text as they hold little or no value in matching queries to documents. In modern systems, however, stop lists are not widely used as they provide little gain in terms of efficiency[MRS08].

Part of Speech(POS) Tagging

POS tagging is a process carried out after tokenisation. Its task is to assign tags to words, for their corresponding grammatical parts of speech, based on both the word's definition, and its context. This is essentially identifying words as nouns, verbs, adjectives, etc.

However, the process is more complicated than it may first seem. The main issue is that most words do not just have one part of speech, they can have many. For example, *can* could be a noun or a verb, depending on the context it is being used in. Thus, when tagging words, it is important to analyse a whole phrase or sentence. Analysing a word out of context could have significant repercussions. In the context of this project, taking the Microsoft software *Paint* as an example, if someone were to tweet:

> Microsoft Paint has seen some major improvements in its latest release!

This would differ from, say,

> Paint something now.

where *paint* is being used as an imperative verb. In the first example, seeing that *Paint* is followed by *seen*, a verb, suggests it is unlikely *Paint* is being used as verb. This would be the difference in this project between discovering a piece of software and totally missing it, and thus highlights the importance of context and semantics when analysing text.

Stemming and Lemmatisation

Documents contain many different derivations of words, such as *normalise* and *normalisation*, and differing forms of the same word due to its usage or tense, for example, *walked* or *walking*. An information extraction tool should ideally see these as somewhat equivalent terms; this is where stemming and lemmatisation come in. The following example, taken from [MRS08], shows how these techniques can map text:

> am, are, is $\Rightarrow$ be
> car, cars, car's, cars' $\Rightarrow$ car
>
> the boy's cars are different colors $\Rightarrow$
> the boy car be differ color

Stemming is a heuristic process hoping to achieve this goal by simply building basic forms of words by removing affixes like a plural 's' from nouns or the 'ing' from verbs[HNP05]. However, these are not always correct terms. Lemmatisation on the other hand utilises a more sophisticated approach in that it uses a vocabulary and morphological analysis of words, in order to return to the true base form of a word that may be found in a dictionary. This process, however, is much more complex and time-consuming than the former.

2.1.3 Information Extraction(IE)

The goal of IE is to extract specific data from a given corpus[1]. IE can be defined as the task of automatically extracting this structured information from unstructured or semi-structured machine-readable documents, generally done through the use of NLP techniques.

In structured texts, information extraction can be fairly straightforward, as labels or tags may delimit strings that need to be extracted[Sod99]. However, in unstructured texts, information is not as clearly understandable by computers and so IE requires techniques from other fields such as machine learning, statistical analysis or those previously discussed from natural language processing.

Typical IE tasks include the following:

Named Entity Recognition(NER)

The aim of NER is to annotate a source text with markup tags in order to classify strings representing predefined categories such as names, companies, locations, dates and times. For example,

> Cook named new Apple CEO.

would yield the following annotations.

[1] A collection of documents

<ENAMEX TYPE="PERSON">Cook</ENAMEX> named new
<ENAMEX TYPE="ORGANIZATION">Apple</ENAMEX> CEO.

This example is using the *ENAMEX* tags defined at the Message Understanding Conference(MUC) in the 1990s[GS96]. From the source text it can be seen that *Cook* has been identified as a person and *Apple* as an organisation and structures this text in doing so.

Relationship Extraction

This works with entity extraction in that it works to identify relations between entities. Using the previous example, the relationship extraction process should be able to identify that,

PERSON named new ORGANISATION CEO.

2.2 Sentiment Analysis and Opinion Mining

Sentiment analysis, also known as opinion mining, refers to the NLP application of extracting subjective information in source texts. There are two use cases for sentiment analysis. The first of these is determining whether a text is subjective or objective, that is, whether the statement is factual or opinionated. This scenario is not currently in the scope of the project and leads to the second use case; sentiment analysis also aims to classify the *polarity* of a given text[PL08]. This, to the basic level, involves determining whether an expressed opinion is positive, negative or neutral, but can also be extended to more complex emotions such as happy or sad.

Opinion mining can be done using a weighting system. This method assigns a positive or a negative weighting on a given scale such as -3 to +3. These are applied for each word in the text that relates to the core entity. The text is then given a total score which determines its polarity, and also the strength of the sentiment. In simpler systems the scale may only be from -1 to +1, essentially opting to ignore the strength of the sentiment and just asking for the general sentiment of the text.

2.3 Twitter Mining

There has been several previous works on text mining Twitter posts, however, the bulk of these have focussed primarily on biomedicine and the financial sector. These works have shown the potential in Twitter has for providing valuable knowledge and information it is felt that software is a new area of interest where Twitter has not previously been used to analyse public opinion. Twitter's low character limit means users have to express their opinions explicitly and encourages the use of emoticons, such as :) or :(, which have been shown to work very well in sentiment analysis[Rea05].

2.4 Twitter API

Twitter provides three public APIs for developers to access their massive corpus of data. These are the REST, Search and Streaming APIs, and shall now be further explored.

2.4.1 Search API

The Search API is the simplest tool provided by Twitter. This API is designed to allow users to query for Twitter content and works very much like the search bar found on the Twitter website itself. This content may include a set of tweets with specific keywords or tweets to, from or mentioning a specific user. A simple search would yield up to 1500 of the latest tweets in the last 7 days, which have been cached over a 60 second period. There are, however, restrictions on the rate at which programs can utilise this API[@tw12b].

2.4.2 REST API

The REST API enables programs to access more of the core Twitter functions. This API retrieves not only the information taken from the Search API but also allows building timelines and retrieves more specific user information such as the user's name, profile avatar, tweet count and the number of followers and friends they have. The REST API also allows programs to post on Twitter and carry out other functions like retweeting or favouriting tweets. These extra functions, however, are not required in this project.

2.4.3 Streaming API

Twitter's Streaming API is a real-time sample of all public tweets posted on the sample. It allows filtering in various ways such as user id, keywords or even random sampling, and is regarded as the default option for data mining operations. This is because the Streaming API allows a long-lived HTTP connection unlike the other APIs and as such, programs can constantly remain connected so as to retrieve a running stream of tweets, as the name itself suggests. This removes the overheads associated with reconnecting every time you want to make a query and the API also removes all rate limitations so there is no worry of exceeding your quota. Unlike the other APIs, programs must be authenticated to use the Streaming API.

Chapter 3

Design

This chapter details the overall design of the system to be developed in this project. The software engineering methodology to be used will first be discussed, along with use cases, requirements and the architecture of the system. This will be followed up with notes on the class and database design diagrams. The decisions made with regards to some design choices will also be discussed in more detail.

3.1 Methodology

The software engineering methodology used in developing an application can have many effects on its final outcome. The development of this system will be carried out using principles taken from continuous integration and agile methods such as feature-driven development. There is always a working code repository available for deployment, and all new features to be implemented are to be worked on in clones of said repository. Upon completion of these minor implementations, they are tested to ensure everything is working correctly and assuming there are no issues, the changes are merged into the base repository. This methodology assures developers that if any major issues arise due to recent changes, they will be able to discard all changes and restart if they feel debugging would be a longer process. This ultimately allows for a faster development cycle and provides rigorous testing throughout the implementation process. This development methodology also allows for frequently changing requirements which is to be expected in any development task.

3.2 Use Cases

There are two main use cases for the proposed system. The following use case definitions apply only to the first stage of the system, i.e. retrieving and storing posts from Twitter. In both cases the system requirements converge, and will be explored below.

3.2.1 UC1 - Streaming Twitter

The first use case for the system requires a tool capable of continuously monitoring public tweets and storing these in a database. These tweets should be filtered by language and relevance, that is, tweets should be related to software. These tweets need to be filtered by language

to counter any issues faced in the feature extraction stage due to the complexities involved in NLP. This use case will thus be referred to as streaming Twitter as that is its principle aim.

3.2.2 UC2 - Searching Twitter

The second use case for the proposed system requires the ability to search Twitter for tweets concerning user-specified key terms, and as such will henceforth be referred to as searching Twitter. These key terms should also be related to software. The returned tweets will also be filtered by language as in the streaming use case to counter the NLP complexities. An extra function required here should be to see which software tools have been mentioned most often, and these should be displayed to the users with the option to search again, or see the full analysis of these tools.

3.2.3 Merging the Use Cases

Upon fulfilling these core requirements, the systems should store these tweets in a database for work in the remaining stages. The first of these is extracting the previously stated features from these tweets. These features must again be stored separately from the initial tweet data, as some tweets may contain information regarding more than one piece of software. Finally, all of this information must be aggregated and shown to users in the form of charts displaying sentiment, and all relevant information found alongside it.

3.3 General Architecture

As previously explained, the system design follows a 3-stage approach, these being tweet retrieval, feature extraction and visualisation for users. These stages are shown in the general archicture model displayed in Figure 3.1. These will be further explored in Sections 3.3.1, 3.3.2 and 3.3.3 respectively.

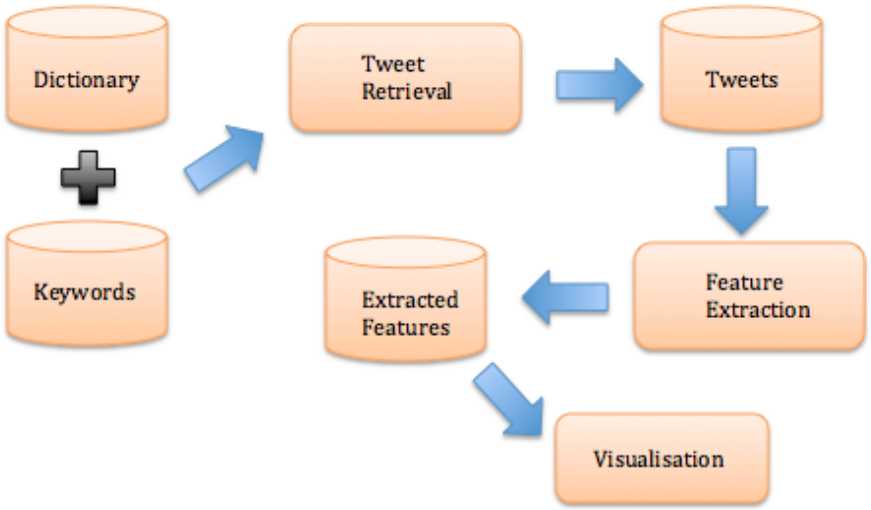

Figure 3.1: General architecture of the system

3.3.1 Retrieving Tweets

The first stage involves retrieving tweets from Twitter. The design for this stage follows the same concepts for each of the use cases defined. This can then be split further as seen in Figure 3.2. The program should retrieve a set of search terms from a dictionary along with some keywords that may be associated with software. These are to be used to form a request for tweets from Twitter. Twitter will respond with the corresponding tweets and data, which are to be checked for language, to ensure they are in English. The remaining set of English tweets are then to be further parsed to extract the required information for storing these tweets in the database.

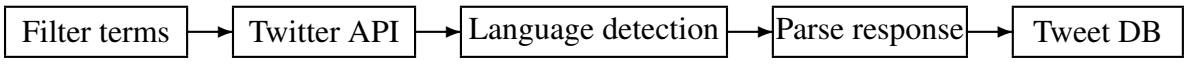

Figure 3.2: Design for tweet retrieval

This returned information will be stored in a relational database and its design is shown below in Figure 3.3. Tweets will not be stored alone but also with simple user information to allow for future user profiling for a more targetted approach to tweet retrieval. The actual tweet information to be stored is its id on the Twitter platform - allowing for cases where users delete their tweets - its text content, time of creation, user id and the keyword used to find it, where one was used. There will also be fields for sentiment, i.e. positive, negative or neutral, and sentiment strength, where weightings have been used. These fields should default to `NULL` as their values will be computed at a later time. Finally, there is a *tagged* boolean flag which signifies the given tweet has been processed, and the feature extraction process has been carried out on it. This is to be implemented in MySQL due to its simplicity, compatibility with the design and also because it is readily available on the university computers where data can be easily accessed both locally and remotely.

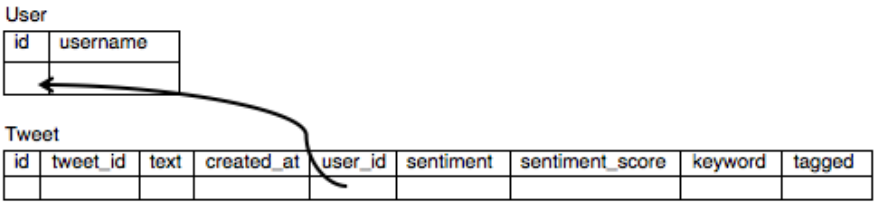

Figure 3.3: Database design for storing tweets

3.3.2 Feature Extraction

The feature extraction stage will execute the task of extracting information from tweets. This will involve the stages described in 2.1 and its general design can be seen in Figure 3.4. The main aim of this stage is to take the tweets previously stored in the database and find the softwares mentioned in them.

Due to the volatile nature of the information being extracted from these tweets, this data should be stored in a NoSQL-type database, that is, a schema-less design that diverges from

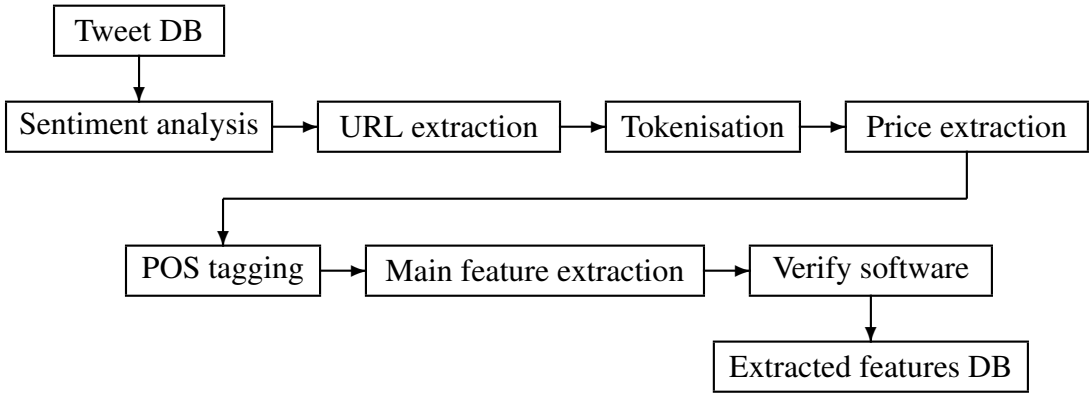

Figure 3.4: Design for feature extraction

the traditional relational database model. As a result, there is no formal design for this database structure, however, it is required of the system to at this stage store any features it has managed to extract along with the key information associated with the tweet that had been stored in the tweet retrieval stage, such as its unique id and text content.

3.3.3 Visualisation

The visualisation stage has the task of displaying all the gained information and knowledge to the user. Ultimately, it must also provide a graphical interface for users to interact with the system in order to perform their own searches.

Chapter 4

Implementation

This chapter outlines the main stages in implementating the designed system in code.

The system to be developed, as initially described in Figure 3.1, is fairly representative of a Question-Answering(QA) system. QAs are data mining systems which use Information Retrieval(IR) and Information Extraction(IE) techniques to answer user queries. As such, this project was implemented in 3 stages, each corresponding to these subsystems in QA systems. The first of these was retrieving tweets, the IR phase of this project. Upon successful retrieval, information, the required features in this case, must be extracted and finally, these results must be displayed to the user in a simple, straightforward fashion. These stages will be further explored as follows:

4.1 Tweet Retrieval

Without tweets, there is no work to be done, and so retrieving tweets can be regarded as the most important part of this project. The main objective of this stage is to retrieve as many relevant tweets from Twitter as possible. To do this, the system will interact with the set of public APIs Twitter provides in order to fulfill the requirements of each use case stated in Section 3.2. The system is designed to use all of these to fulfill the requirements of each use case. This subsystem in the project is implemented in Java. This is because of its strong object oriented nature and platform independency. Of course, there are other options such as Python, however Java is a simple programming language with relatively straightforward multithreading capabilities.

4.1.1 Streaming Twitter

The Streaming API allows the system to fulfill the requirements of having a fully automated system that constantly monitors Twitter for software-related posts.

The implementation at this stage utilises Twitter's filtering URL at `https://stream.twitter.com/1/statuses/filter.json` and passes it a set of dictionary terms and keywords to filter tweets by. This dictionary consists of a list of software, companies, operating systems and programming languages. The set of keywords contains words like *release* or *version*, which may be associated with software and are likely to be mentioned in software-related tweets. The full list of these dictionary items and keywords can be seen in Appendix A.

This implementation could have been done using the *Twitter4J*[1] Java library for Twitter integration, however most of the functions appear unnecessary and excessive in the scope of this project. For this reason, the Twitter Streaming API integration was implemented from scratch.

Upon retrieving these filter terms from the database, the application formats this list into a string after which it creates three new Thread objects, a *DatabaseThread* which will carry out all database operations, a *StreamParseThread* which parses the stream of responses sent back from the Twitter server, and a *ScannerThread*, which monitors the running state of the program, so as to allow a clean exit when the user wishes to quit. This scanner thread simply monitors the console input for users to type the exit command, upon which all connections are dropped and final parsing and database operations are carried out before closing the application. This high level control flow can be seen in Figure 4.1.

On initialising these threads, the application attempts to set up a secure connection to Twitter using the HTTPS protocol. It uses the POST method to write the string of filter terms to the server in order to being receiving tweets. Once this connection is fully set up, Twitter will return JavaScript Object Notation(JSON) strings for each tweet, and so a JSON parser is set up using the Google-Gson Java library[SLW11]. The aforementioned threads are then started as the actual streaming process now begins.

For every tweet returned by the API, the application adds this JSON response, as a *JsonObject*, to a queue in the *StreamParseThread* class, using the following simple method:

```
private final List<JsonObject> parseList = new ArrayList<JsonObject>();

public boolean addTask(final JsonObject object) {
    synchronized (parseList) {
        return parseList.add(object);
    } // synchronized
} // addTask(JsonObject)
```

Listing 4.1: Adding tweets to a parse queue

The parser thread now assumes control of the processing to be done, while the main thread continues to add to this *parseList* queue. The parser thread has the sole task of parsing the information in this JSON object into a more meaningful *Tweet* object. This class' properties can be seen in Listing 4.2. To do this, the JSON object first needs to be checked if it represents a tweet delete entity, that is, a object containing the "delete" key signifying a user has deleted their tweet. In such a case, Twitter requests that applications honour the user's requests and delete this tweet. If otherwise the JSON object is actually a tweet, the program extracts the Twitter user's details to check their locale. In cases where this is not English, a `null` value is returned and the tweet is ignored. If this test passes, all the remaining properties described in the *Tweet* and *User* classes are extracted and returned as a single Tweet object.

This Tweet object is encapsulated in an *InsertKeywordTask* object. This class is an implementation of the *DatabaseTask* interface, which is used to perform the different database operations when used in conjunction with the different *DatabaseConnector* classes. The hierarchy of these classes can be seen in Figures 4.2 and 4.3 respectively. To further clarify, the *DatabaseThread* constructor takes a *DatabaseConnector* object as an argument. This allows for a more extensible system as different types of database management systems can be used with the application. It must be noted that in the current implementation, the *DatabaseThread*

[1] `http://twitter4j.org/`

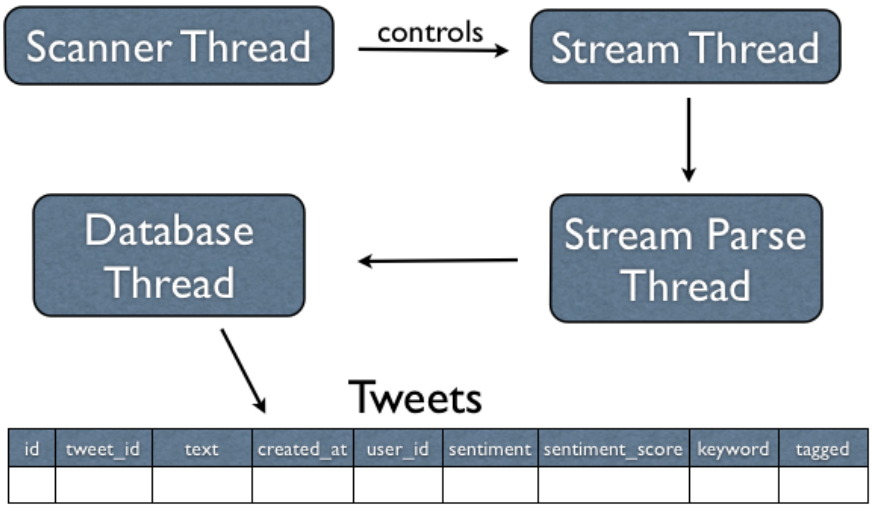

Figure 4.1: Control flow for tweet retrieval subsystem

class has been implemented at a similarly high level of abstraction, and as such an *SQLThread* extends this class for use with a MySQL database. The *SQLThread* initialises with a *TweetSQL-Connector* object, as it only operates in classes related to tweet retrieval. The DatabaseThread class maintains its own queue of *DatabaseTask* objects as the parser thread, previously seen in Listing 4.1.

```
public class Tweet {
    private final long tweetId;
    private final String tweet;
    private final String createdAt;
    private final User user;
    private String keyword = null; // Only used when filtering
    ...
} // class Tweet

public final class User {
    private final long id;
    private final String username;
    ...
} // class User
```

Listing 4.2: Tweet and User class properties

Class design aside, once this *InsertKeywordTask* object has been created, it is added to the queue in the *DatabaseThread*. The respective implementations of the *doTask()* method in each of the DatabaseTask classes will be performed as this queue is emptied. In the case of InsertKeywordTask, this is simply `db.insertTweet(t)`, where `db` is the TweetDatabaseConnector passed to it in the `doTask()` method, and `t` is the Tweet object it was initialised with.

This process is completed when the *TweetDatabaseConnector* inserts the tweet into the database, in the current implementation using Java Database Connectivity(JDBC) to manipulate the MySQL server.

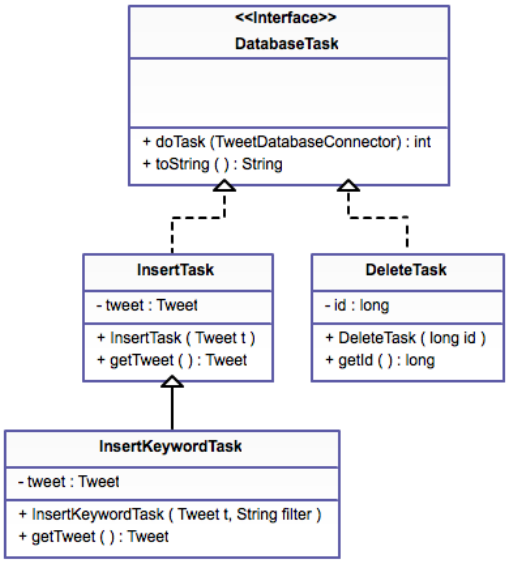

Figure 4.2: DatabaseTask class diagram

4.1.2 Searching Twitter

The Search API will now be used to handle the alternative use case of the designed system. With the Search API not operating in real time, the realisation of this use case can afford to be a much simpler system. The levels of multithreading displayed in streaming Twitter will not be required, as interaction with the API is more of a serial communication as can be seen in Figure 4.4.

The implementation of the Twitter search use case utilises the Twitter Search API URL at `http://search.twitter.com/search.json`. The API also offers eXtensible Markup Language(XML) format responses, however, working with JSON allows consistency within the system. As with the implementation of the Twitter stream use case, the application begins with the initialisation of a *DatabaseThread* and is initially given a single keyword which will be searched for. This keyword is chosen by the end user. A simple HTTP GET request is then made to the above URL and Twitter then returns up to 1500 tweets from the last seven days corresponding to the search term.

The Search API returns a different JSON response to that of the Streaming API. Each JSON string contains an `iso_language_code`, and this will be used to filter tweets by language. Once the desired information has been extracted, i.e. the properties of the *Tweet* class, the remaining operations are carried out just as they are in the realisation of the Twitter streaming use case, that is, the tweet is encapsulated in an *InsertKeywordTask* object and this is added to a queue in the *DatabaseThread* for the insert task to be carried out.

4.2 Feature Extraction

Once tweets have been retrieved from Twitter and stored in the MySQL database, they are now available for feature extaction, which can be regarded as the core stage in implementing the system. This subsystem involves using NLP techniques to extract the information shown in Table 1.3 from each tweet. This subsystem is implemented in Python 2.7 due to the raw power

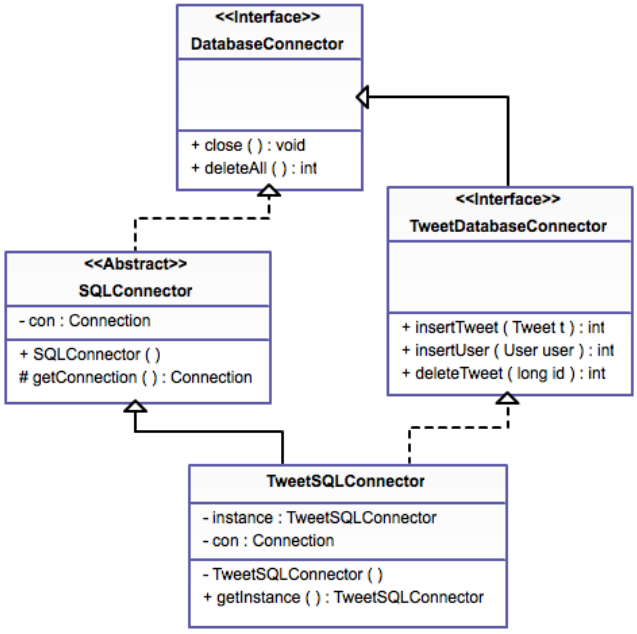

Figure 4.3: DatabaseConnector class diagram

it possesses and also due to the decision to use the Natural Language ToolKit(NLTK)[BLK09]. NLTK is an active, well documented Python toolkit project. Alternatives to NLTK include GATE[2] and Minor Third[3]. GATE is equally well documented, however it appears to be a bulky library, and many of its features are not required in the scope of this project. Minor Third on the other hand, is not as well documented, at the time of development at least, and as such would be more difficult to integrate. These alternatives are also Java implementations and it was felt that Python's speed and text manipulation would allow for a better implementation of the system.

There are many steps involved in implementing the feature extraction. These are explored in order of execution.

4.2.1 Sentiment Analysis

Sentiment analysis was recognised as one of the key features to be extracted from the initial design stages. It has been implemented using the Twitter Sentiment Bulk Classification Service API. This was chosen ahead of others such as AlchemyAPI[Alc11] and the CLiPS Pattern modules. AlchemyAPI results were accurate, however with the massive number of tweets being streamed from the service it was not deemed feasible to continuously make calls to a web service to analyse them for sentiment, as the service only analyses individual tweets. As well as this, there is a limit of 10,000 tweets per day and with the large numbers of tweets posted on Twitter on the daily basis, this was also an issue.

While Pattern is an offline system, it states 72% accuracy for movie reviews[CLi12], while the Twitter Sentiment API is optimised for tweets and boasts 83% accuracy[GBH09]. As well as this, the bulk classification service allows mass analysis with requests consisting of up to

[2] http://gate.ac.uk/

[3] http://sourceforge.net/apps/trac/minorthird/wiki

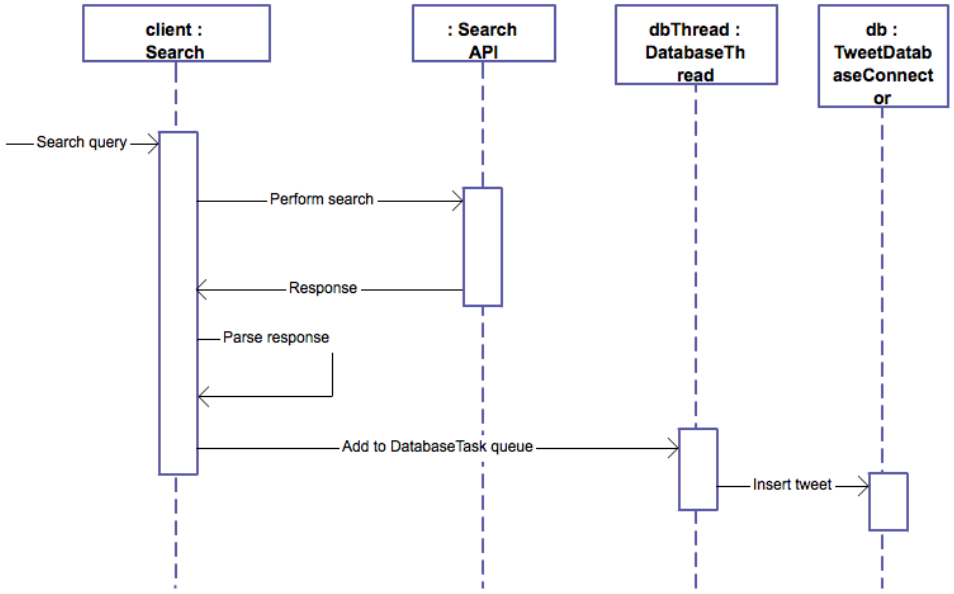

Figure 4.4: Searching Twitter sequence diagram

10,000 tweets.

The sentiment analysis of tweets is carried out before any of the other feature extraction work. As tweets have been retrieved and stored in the MySQL database, this part of the system selects 100 of the latest tweets, retrieving just the id and text, that have yet to be analysed and packs them into a JSON string object of the format:

```
{ "data": [ { "id": "1234", "text": "Google Chrome is awesome!" },
            { "id": "1235", "text": "Safari 5.0.2 is out now" },
            { "id": "1236", "text": "I really hate the new Firefox" } ] }
```

This JSON string is then posted to the Twitter Sentiment API where classifications into the positive, negative and neutral classes are carried out by a Maximum Entropy classifier trained with tweets containing emoticons. The internal specifics of a Maximum Entropy classifier, however, is not in the scope of this project.

Currently only 100 tweets are analysed at a time due to time constraints when users wish to run the program in real time. By using a small number, the data needing to be transferred is minimal and allows for a more interactive user experience.

Using the previous example, the data is returned by the server in the following format, with a polarity field added to each analysed tweet with values 0, 2 and 4 corresponding to negative, neutral and positive respectively.

```
{ "data": [
  { "id": "1234", "text": "Google Chrome is awesome!", "polarity": 4},
  { "id": "1235", "text": "Safari 5.0.2 is out now", "polarity": 2 },
  { "id": "1236", "text": "I really hate the new Firefox", "polarity": 0 }
] }
```

Upon receipt of this response, the JSON formatted string is parsed and the corresponding record for the tweet previously stored in the MySQL database is updated with new values for sentiment score and the actual sentiment, using polarity and its semantic meaning respectively.

4.2.2 URL Extraction

Before extracting context and semantics from tweets, any URLs mentioned are found and removed. Assuming the tweet is software-related, these URLs are quite likely to be links to the software, or further reviews. This task is done using NLTK's `regexp_tokenize()` function with `http://[^ ]+` passed as the regular expression that finds URLs. If the tweet is later found not to contain any software, these URLs are discarded. Potential issues with this implementation could be that a user may post a URL without the preceding `http://` protocol prefix and these would not be found by this regular expression. However, Twitter automatically converts URLs to their `http://t.co/` domain and so this is resolved on the Twitter server side.

4.2.3 Tokenisation

After URLs have been extracted and removed from the source text, the tweet is tokenised to produce an array of all the terms in the tweet. The tokenisation process is also done using NLTK's `regexp_tokenize()` function, passing it the regular expression - `\w+([.,]\w+)*|\S+`. This expression returned superior results to alternation tokenisation functions provided by NLTK, such as `wordpunct_tokenize()` as it was capable of finding numbers and currencies without splitting them. Using the above example,

> I really hate the new Firefox

this would be tokenised to the following:

> ['I', 'really', 'hate', 'the', 'new', 'Firefox']

The following example shows a more complicated tokenisation process.

> Norton Anti-Virus released for $50 #ripoff
>
> ⇒ ['Norton', 'Anti', '-Virus', 'released', 'for', '$50', '#ripoff']

4.2.4 Price Extraction

Continuing on from this tokenisation of the original source text, the current subsystem attempts to find prices in the array of terms. This is done using Python's built-in regular expression module, `re`. A number of regular expressions are used to define patterns denoting numbers, currencies and quantifiers like 'hundred' and 'thousand'. As the form of prices vary, for example in the case of mobile apps you might find '£0.59', '59p' or even '59 pence', these combinations of tokens may be split across two tokens in the array returned from the tokenisation process. For this reason, it is necessary to iterate over all items in the list of tokens while remembering the previous one. This obviously means a less efficient system, however, it has produced the best results in such variable conditions.

4.2.5 Part-of-speech (POS) Tagging

The POS tagger used by this system is taken from the NLTK modules and uses the `pos_tag()` function which takes a tokenised sentence as its only argument. Continuing from the first example, this process tags as follows:

['I', 'really', 'hate', 'the', 'new', 'Firefox']

⇒ [('I', 'PRP'), ('really', 'RB'), ('hate', 'JJ'), ('the', 'DT'), ('new', 'JJ'), ('Firefox', 'NNP')]

PRP	Pronoun
RB	Adverb
JJ	Adjective
DT	Determiner
NNP	Proper Noun

4.2.6 N-Grams

N-grams are sequences of n tokens from a given source text. The implementation of creating n-grams in this project is done using the `nltk.util.ngrams()` function. This process starts by creating a five-gram of the tweet tokens. This means a sequence of five tokens will be created from the array of tokens. The system utilises a five-gram sequence due to potentially long software names, basing this on the naïve assumption that these names will not exceed five words. This will allow for improved extraction of software names in the next stage. Using the Firefox tweet as a running example, the outcome of this five-gram modelling process can be seen below.

['I', 'really', 'hate', 'the', 'new', 'Firefox']

⇒

[(('I', 'PRP'), ('really', 'RB'), ('hate', 'JJ'), ('the', 'DT'), ('new', 'JJ')),

(('really', 'RB'), ('hate', 'JJ'), ('the', 'DT'), ('new', 'JJ'), ('Firefox', 'NNP'))]

4.2.7 Main Feature Extraction

This tagging process consists of the core functions of the proposed system. Its purpose is to extract all the features that have yet to be extracted, that is, software names and versions, companies, programming languages and operating systems. It also attempts to find any prices that may previously have been missed, and also has the task of discovering software that is not already in the dictionary.

Having created a set of five-gram sequences from the tweet, the application may now iterate through each of these in an attempt to find any information that has not yet been found. For each of these sequences, the program iterates through each POS-tagged token in the sequence. The tagging process then proceeds as follows:

if token is tagged as a noun **then**
  if token is in dictionary of software, companies, os, programming languages **then**
    if previous token not tagged as determiner or preposition **then**
      Feature has been found
    end if
  end if
end if

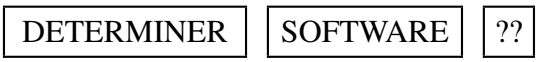

Figure 4.5: A linguistic filter

This rule filters out linguistics of the form shown in Figure 4.5. If however, these conditions fail, usually in the case where none of the tokens are in the dictionary of keywords used to retrieve these tweets, a regular expression is used to find clues to the presence of new software.

```
^download$|^get$
```

The above regular expression matches on the words *download* and *get*. This works on the basis that many tweets about software are usually posted to promote said software. This is generally done by urging others to download it, and that too by means of application stores like the App Store, or Google Play. This then allows the next token to be analysed to check if it is in fact a piece of software. This is done by checking that token's part of speech tag, adn if it is a noun, the following tokens are also assessed in case the name of the software is longer than one word. This possible software name is then noted and kept aside for verification by web search as discussed in Section 4.2.8. This regular expression can be applied to the five-gram in conjunction with others in order to maximise information gain. The following expression could be used to find an operating system.

```
^on$|^for$
```

By applying these together in the form displayed in Figure 4.6, the system may be able to determine the platform upon which a piece of software runs.

Figure 4.6: A linguistic rule to find software and the operating system it may run on

- `^is$|^for$`
- `^free$`

```
{
  'programming_language_id' : '317',
  'programming_language_name' : 'python',
  'software_id' : '159',
  'software_name' : 'moodle',
  'tweet' : 'Best Affordable Drupal, Python, Moodle Hosting:
             Top 3 Drupal, Python, Moodle Web Hosts http://t.co/0WHPTsns',
  'tweet_db_id' : '440094',
  'url' : 'http://t.co/0WHPTsns'
}
```

Listing 4.3: Example of some extracted features

4.2.8 Software Verification

The feature extraction subsystem may discover new software, and as such needs to verify these are actually pieces of software and not something else. To do this the program utilises the Microsoft Bing API which returns web search queries. As the main tagging process checks the dictionary for matching software names, and the tweet retrieval engine uses both the dictionary and a set of keywords, there will be some pieces of software mentioned in the tweets that are not in the dictionary. As a result, these will be flagged as possible software names, and then queried on the Microsoft Bing search engine with the keywords "movie", "music", and "software game". These keywords were selected on the basis that the initial search key terms retrieved many tweets referring to music and films.

```
function bing_search(bing, term){
    music = bing.search(term, music)
    movie = bing.search(term, movie)
    software = bing.search(term, software game)

    if size(software) greater than size(movie) and size(music)
        if references to software in title and description
            return True
    return False
}
```

If the number of results for software associated with the searched term is greater than corresponding results for films and music, the results are checked for identifiers of software in their headings. Therefore if any of the results suggests the searched term is a piece of software, that is assumed true.

4.3 Storing the Extracted Information

As the information being extracted is temporarily stored in a Python dictionary variable, it is essentially in the form of a JSON string. The database design for storing this information is also in the form of a NoSQL database. For this reason, a document-based database system seems to be the best approach. MongoDB[4] is a document-oriented NoSQL database, which stores JSON-style documents. By using MongoDB, it is easy to store the extracted information, as it is as straightforward as directly storing the string representation of this variable as a record in the database.

4.4 Visualisation/Graphical User Interface(GUI)

The final stage of the project is to aggregate and present the results to the user in a GUI. Aggregating the results is the process of bringing together all the different data sources for data on a single output entity such as a piece of software. This aggregated data can then be used easily by the GUI to display understandable information to the user. The GUI of choice is a web application as opposed to a desktop application, as it allows for a more centralised system

[4] http://www.mongodb.org/

that users can easily connect to. It is also a more scalable solution as updates would not need to be pushed out to all users.

4.4.1 Aggregation

4.4.2 Web Application

The web application available to users is a Python application running the CherryPy web framework.

Chapter 5

Testing and Results

This chapter details the testing methods use over the course of this project along with results and the final findings of the system.

5.1 Testing

5.2 Results

5.2.1 GUI

5.2.2 Discussion

Chapter 6

Evaluation

With implementation and analysis complete, one can now identify and evaluate the key successes and shortcomings of this project. These evaluations have been performed on the basis of the following questions:

- Does the system work?
- Is the information found novel and interesting?
- Is the system easy to use?

This process has been carried out by means of independent user evaluations by 5 users of the software.

Chapter 7

Conclusions

This chapter discusses the author's reflections on the success and conclusions of the project. Suggestions for further work are made and concludes with a summary of the report.

7.1 Reflections

In terms of the overall progress made over the course of the year I feel this project has to some extent been a success. It has been a steep learning curve and on that basis some good results have been achieved. However, in absolute terms, I think I have made many mistakes in the way I went about working on the project over the year. This is down to a few key issues.

Time management can be seen as one of the overriding causes of the shortfalls of this project. There were times when progress had completely stalled due to minor issues with implementation.

I also feel as though my **preparation** may not have been sufficient, as I had overlooked a few key concepts when developing parts of the system. For example, I did not create a training set of data that would ultimately allow me to use a machine learning approach in my system and I think this may have resulted in a slightly lesser performing program, in terms of its accuracy.

However, as previously stated, there have been major strides over the year. I feel I have a much greater grounding in the core object oriented design and programming principles. As well as this, I have developed skills in many new technologies and concepts. This was the first time I have had to develop a multithreaded desktop environment, and I had never previously worked with Python, JSON, MongoDB or the HTTPS protocol. As such I am happy with the general progress made in this project.

7.2 Future Work

Bibliography

[Alc11] AlchemyAPI. Sentiment analysis. http://www.alchemyapi.com/api/sentiment/, October 21, 2011.

[BLK09] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[CCMS10] Courtney D Corley, Diane J Cook, Armin R Mikler, and Karan P Singh. Text and structural data mining of influenza mentions in web and social media. *International Journal of Environmental Research and Public Health*, 7(2):596–615, 2010.

[Cha97] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18:33–44, 1997.

[CLi12] CLiPS. Pattern.en sentiment. http://www.clips.ua.ac.be/pages/pattern-en#sentiment, April 6, 2012.

[Cul10] Aron Culotta. Detecting influenza outbreaks by analyzing twitter messages. *CoRR*, abs/1007.4748, 2010.

[FD95] R. Feldman and I. Dagan. Kdt - knowledge discovery in texts. In *Proc. of the First Int. Conf. on Knowledge Discovery (KDD)*, pages 112–117, 1995.

[GBH09] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *Processing*, pages 1–6, 2009.

[GL09] Vishal Gupta and Gurpreet S Lehal. A survey of text mining techniques and applications. *Journal of Emerging Technologies in Web Intelligence*, 1(1):60–76, 2009.

[Gri08] Seth Grimes. Unstructured data and the 80 percent rule. Carabridge Bridgepoints, 2008.

[GS96] Ralph Grishman and Beth Sundheim. Message understanding conference - 6: A brief history. In *Proceedings of the International Conference on Computational Linguistics*, 1996.

[Hea99] Marti A. Hearst. Untangling text data mining. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 3–10, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

[HNP05] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 20(1):19–62, May 2005.

[Hun07] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, May-Jun 2007.

[KV05] M. Krallinger and A. Valencia. Text-mining and information-retrieval services for molecular biology. *Genome Biol*, 6(7), 2005.

[LMRS07] Witold Litwin, Riad Mokadem, Philippe Rigaux, and Thomas Schwarz. Fast ngram-based string search over data encoded using algebraic signatures. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 207–218. VLDB Endowment, 2007.

[MRS08] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[PD11] Michael J. Paul and Mark Dredze. You are what you tweet: Analyzing twitter for public health. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[PL08] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, 2008.

[Pol06] Tamara Polajnar. Survey of text mining of biomedical corpora. June 2006.

[PP10] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010. European Language Resources Association (ELRA).

[Rea05] Jonathon Read. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL Student Research Workshop*, ACLstudent '05, pages 43–48, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[Sar08] Sunita Sarawagi. Information extraction. *Found. Trends databases*, 1(3):261–377, March 2008.

[Ski08] Diane J. Skiba. Nursing education 2.0: Twitter & tweets. *Nursing Education Perspectives*, 29(2):p110 – 112, 2008.

[SLW11] Inderjeet Singh, Joel Leitch, and Jesse Wilson. Gson user guide. `https://sites.google.com/site/gson/gson-user-guide`, October 15, 2011.

[Sod99] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34:233–272, 1999. 10.1023/A:1007562322031.

[@tw12a] @twitter. Twitter turns six. `http://blog.twitter.com/2012/03/twitter-turns-six.html`, March 21, 2012.

[@tw12b] @twitterapi. Getting started. https://dev.twitter.com/start, April 2, 2012.

[WH98] Mark Warschauer and Deborah Healey. Computers and language learning: an overview. *Language Teaching*, 31(02):57–71, 1998.

Appendix A

Dictionary of Software and Keywords