REAL TIME SENTIMENT ANALYSIS

Third Year Project Report

Stefan Cristian Tutuianu
BSc. (Hons.) Computer Science with Business and Management

School of Computer Science,
The University of Manchester

May, 2016

Project Supervisor: John Mcnaught
Table of Contents

Abstract 5

Acknowledgements 6

Chapter 1 - Introduction 7
  1.1 Motivation ................................................................. 7
  1.2 Aims and Objectives .................................................. 7
  1.3 Structure ............................................................... 8

Chapter 2 - Background 9
  2.1 Text Mining .............................................................. 9
  2.2 Natural Language Processing (NLP) .............................. 9
    2.2.1 Tokenization ..................................................... 10
    2.2.2 Part of Speech Tagging (POS) ............................... 10
    2.2.3 Stemming and Lemmatization ................................ 11
    2.2.4 N-grams .......................................................... 11
    2.2.5 Pareto Principle ................................................ 12
  2.3 Machine Learning Classification ................................. 12
    2.3.1 Naïve Bayes ..................................................... 13
    2.3.2 Support Vector Machine (SVM) .............................. 14

Chapter 3 - Design 15
  3.1 Why Python? ............................................................ 15
  3.2 System Architecture ................................................ 16
  3.3 Methodology .......................................................... 16
  3.4 Requirements gathering ......................................... 17
    3.4.1 Engine Requirements ......................................... 17
    3.4.2 Graphical User Interface (GUI) ............................. 19
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Tokenization example</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Supervised learning pipeline</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Bayes’ Theorem</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>SVM - two classes example</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Architecture overview</td>
<td>16</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Real time component class diagram representation</td>
<td>21</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Pipeline experiment - Naïve Bayes training phase</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Tokens distribution - Naïve Bayes model</td>
<td>26</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Unit testing (engine) - tokenizer module</td>
<td>28</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Benchmark - Metafused Limited</td>
<td>31</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1:</td>
<td>Objectives set for the project</td>
<td>7</td>
</tr>
<tr>
<td>Table 2.1:</td>
<td>Part of speech tags used throughout the project</td>
<td>10</td>
</tr>
<tr>
<td>Table 2.2:</td>
<td>Stemming rules and examples - Porter</td>
<td>11</td>
</tr>
<tr>
<td>Table 3.1:</td>
<td>Engine - functional requirements</td>
<td>18</td>
</tr>
<tr>
<td>Table 3.2:</td>
<td>Engine - non-functional requirements</td>
<td>19</td>
</tr>
<tr>
<td>Table 3.3:</td>
<td>GUI - functional requirements</td>
<td>19</td>
</tr>
<tr>
<td>Table 3.4:</td>
<td>GUI - non-functional requirements</td>
<td>20</td>
</tr>
<tr>
<td>Table 4.1:</td>
<td>Sample of the model obtained in the training phase of the classification</td>
<td>24</td>
</tr>
<tr>
<td>Table 5.1:</td>
<td>Sample of unit testing form</td>
<td>29</td>
</tr>
<tr>
<td>Table 6.1:</td>
<td>Crossfold validation results - Naive Bayes model</td>
<td>30</td>
</tr>
<tr>
<td>Table 6.2:</td>
<td>Comparison between Naive Bayes and Support Vector Machine</td>
<td>31</td>
</tr>
</tbody>
</table>
Abstract

This report demonstrates the production of a real time sentiment analysis system, with the following main objectives set:

1. Obtain a classification accuracy of over 80%,
2. Build an engine adaptable to real time sentiment classification reporting.

As a secondary objective, a graphical user interface was developed to enhance the interaction between the users and the system.

In order to produce the software artefacts presented in the report, computer science knowledge, as well as machine learning and natural language processing techniques were employed. Consequently, the concepts and techniques, which contributed to the development of the project, such as the Naïve Bayes algorithm, are explained.

Furthermore, a high level view and a low level view of the system produced are detailed in subsequent chapters. The quality is assessed in the Achievements section, where a performance benchmark and other evaluation techniques are employed.
Acknowledgements

I would like to thank my supervisor, John Mcnaught, who offered his full support throughout the project, by sharing his vast knowledge and expertise in the field of natural language processing.

Special thanks to my parents, Cristinel and Cristina, my sister, Ana, and my girlfriend, Anca, who helped me pass through many difficult moments in the past year.

I would also like to thank to Paul Maddox, for the all the advice and thoughtful insight given throughout the past year.

Lastly, but not least, I would like to thank to my dear friend, Bogdan, who was always there to with his good will and useful advice.
Chapter 1 - Introduction

1.1 Motivation

The emergence in the last decade of social media platforms such as Twitter, Facebook, and Instagram, enabled people to engage in social activities to express their opinions, thoughts, and emotions on a variety of topics. On such platforms, large amounts of data are produced (e.g: 6000 tweets per second), this representing an opportunity for companies to assess their social influence and people opinions towards their products[1]. Consequently, a computational framework is desirable to perform opinion mining and sentiment analysis which can adapt to the activity domain of the user.

1.2 Aims and Objectives

The project aims to produce real time sentiment analysis associated with a range of brands, products and topics. The project’s scope is not only to have static sentiment analysis for past data, but also sentiment classification and reporting in real time. As such, the system should automatically collect and analyse data from Twitter, the primary data source for this project.

For example, the sentence "Brand A is awesome" has positive sentiment for Brand A. More sophisticated structures can be built, for example, the sentence "Brand A is okay but Brand B is great" has neutral sentiment for Brand A, and positive sentiment for Brand B. By the end of the project the goal is to produce up to the minute sentiment values for brands and topics. As such, a system which determines the polarity of tweets (Twitter messages) by using machine learning algorithms and natural language processing techniques is proposed. Table 1.1 presents the objectives set, prioritized by the level of contribution to the scope of the project.

<table>
<thead>
<tr>
<th>No.</th>
<th>Objective</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Build two infrastructures: real time sentiment analysis and long-term</td>
<td>Highest</td>
</tr>
<tr>
<td></td>
<td>sentiment analysis, employing an engine capable to adapt to both</td>
<td></td>
</tr>
<tr>
<td></td>
<td>infrastructures</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Implement a machine learning algorithm to perform sentiment analysis.</td>
<td>Highest</td>
</tr>
<tr>
<td>3</td>
<td>Understand and implement natural language processing techniques.</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Achieve 80% or more in classification accuracy.</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>Build a web application graphical user interface for visualisation purposes</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 1.1: Objectives set for the project
1.3 Structure

The rest of the report contains six chapters. In the next chapter, background knowledge is discussed, along with the techniques and algorithms used in development.

The third chapter - Design, presents a high level view of the system produced. It also covers the design patterns applied, the requirements gathering process, both functional and non-functional, and what methodologies have been used.

Then, the Implementation Chapter demonstrates a low level view of the system, covering the implementation challenges, and what heuristics have been proposed to encounter them. In addition, the system is assessed in the Testing chapter, followed by the Evaluation and Results chapter, where the system’s performance is compared to other similar tools available, and achievements are presented.

Lastly, the conclusion is a reflective chapter, where the completion of the objectives set at the start of the project is assessed according to the timing specifications. In addition, the knowledge gained while working on the project is demonstrated, and future work is proposed.
Chapter 2 - Background

Overview: The following chapter aims to clarify the techniques used throughout the project. A broad definition will be given for the core concepts involved in the development of the artefacts: Text Mining, Natural Language Processing and Machine Learning. Furthermore, specialized terminology will be explained.

2.1 Text Mining

Text mining refers to the analysis of data contained in natural language text, (e.g. messages retrieved from twitter). It can be defined as the practice of extracting meaningful knowledge from unstructured text sources. The application domain of text mining varies from biomedical applications to marketing applications and sentiment analysis. In marketing, text mining is relevant to the analysis of the customer relationship management. This way a company can improve their predictive analytics models for customer turnover (keep track of customer opinions).

The main goal of text mining is to process data into a structured format ready for analysis, via application of natural language processing and other analytical methods.[2] Albeit, there are many aspects within the field of study of text mining, information extraction (IE) is relevant for this project. Consequently, the following material aims to explain the challenges and terminology associated with information extraction and subsequent processing.

2.2 Natural Language Processing (NLP)

For the purpose of explaining further concepts, Twitter will be used as a running example. The data retrieved from twitter presents a certain amount of structuring, in the sense that the maximum length of a tweet is 140 characters long. The advantage of the length limit is reflected in the complexity of the analysis for an individual piece of text. However, this project aims to analyse data in a continuous manner, where a large amount of data (e.g: 200 tweets per minute) will be analysed. Furthermore, there is no certainty that all the tweets will follow a formal structure, neither that they will be grammatically correct. It is also expected that abbreviations and short forms of words, as well as slang will be encountered in the text analysed. Moreover, sentences describing the same or similar ideas may have very different syntax and employ very different vocabularies [2].
Given the aforementioned textual limitation, a predefined textual format has to be produced at processing time. The techniques presented below were used in the development of the project.

2.2.1 Tokenization

The first task, that must be completed before any processing can occur, is to divide the textual data into smaller components. This is a common step in a Natural Language Processing (NLP) application, known as tokenization. At a higher level, the text is initially divided into paragraphs and sentences. As a consequence of the length limitation of 140 characters imposed by twitter, it is rarely the case that a tweet will contain more than a paragraph. In these regards, the project aim at this step is to correctly identify sentences. This can be done by interpreting the punctuation marks such as a period mark “.”, within the text analysed. The next step is to extract the words (tokens) from sentences. The challenge at this step is to handle the orthography within a sentence. Consequently, spelling errors have to be corrected, URLs and punctuation shall be excluded from the resulting set of tokens. As it can be observed in Figure 2.1, after tokenizing a tweet the returned result is an array containing a set of strings.

![Figure 2.1: Tokenization Example](image)

| text: "Want to boost Twitter followers ?! http://bit.ly/8Ua" |
| tokens: ["want", "to", "boost", "twitter", "followers"] |

2.2.2 Part of Speech Tagging (POS)

In order to understand the complete meaning of a sentence, the relationship between its words have to be established. This can be done by assigning every word a category that identifies syntactic functionality of that word. Also known as part of speech tagging (POS), this step can be seen as an auxiliary requirement for n-grams selection and lemmatization. Table 2.1 covers the part of speech notations used in the project.

| ADJ : adjective                  | PART : particle          |
| ADV : adverb                     | PRON : pronoun          |
| AUX : adjective                  | PROP : proper noun      |
| CONJ: conjunction                | PUNCT : punctuation     |
| DET: determiner                  | SYM : symbol            |
| NOUN: noun                       | VERB: verb              |
| NUM: numeral                     | X : other               |

*Table 2.1: Part of speech tags used throughout the project*
2.2.3 Stemming and Lemmatization

The goal of both stemming and lemmatization is to reduce inflectional forms and derivations of a word to a common base form[3]. For example the following words: “connection”, “connections”, “connective”, “connected”, “connecting” will have the same base, which is “connect”. Stemming, is a crude heuristic process that chops off the ends of words, so that only the base form is kept [3]. By contrast, lemmatization uses the morphological analysis of the words, returning their dictionary form (base), commonly referred as the lemma. However, for a language like English as opposed to more morphologically rich languages, this process relies on a dictionary being available. In addition, a lemmatizer can introduce ambiguity by proposing all possible lemmas for a word form, or by choosing the wrong proposal from two competing lemmas (e.g., is axes the plural of axe or of axis?).

Considering the arguments presented above, algorithm chosen for this task is Porter’s stemming algorithm. It consists of five phases, where word reductions are performed. For each phase, rules and conventions to apply them are defined [3]. Table 2.2 exemplifies the rules of the first phase of the algorithm:

<table>
<thead>
<tr>
<th>Rules</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSES</td>
<td>-&gt; SS</td>
</tr>
<tr>
<td>IES</td>
<td>-&gt; I</td>
</tr>
<tr>
<td>SS</td>
<td>-&gt; SS</td>
</tr>
<tr>
<td>S</td>
<td>-&gt; /</td>
</tr>
<tr>
<td></td>
<td>caresses -&gt; caress</td>
</tr>
<tr>
<td></td>
<td>ponies -&gt; poni</td>
</tr>
<tr>
<td></td>
<td>caress -&gt; caress</td>
</tr>
<tr>
<td></td>
<td>cats -&gt; cat</td>
</tr>
</tbody>
</table>

Table 2.2: Stemming rules and examples - Porter [3]

2.2.4 N-grams

N-grams is a common technique in text mining, where word subsets of length n within a sentence are formed. From the sentence “This is a six words sentence!” the following n-grams can be formed:

1-grams (unigrams): “this”, “is”, “a”, “six”, “words”, “sentence”
2-grams (bigrams): “this is”, “is a”, “a six”, “six words”, “words sentence”
3-grams (trigrams): “this is a”, “is a six”, “a six words”, “six words sentence”

As such, the example sentence above will produce 6 unigrams, 5 bigrams, and 4 trigrams. On a bigger data set, producing bigrams and trigrams will considerably contribute to the size of the data set, consequently, slowing down the system. An approach to solve this challenge is detailed in Chapter 4.
2.2.5 Pareto Principle

The pareto principle states that, for many phenomena, 20% of the input generates 80% of the output. It is named after “an Italian Economist Vilfredo Pareto who in 1906 noticed that 80% of the land in Italy was owned by 20% of the population” [4]. By pursuing this observation he noticed similar proportions can be described in economics. The pareto rule is rather an interesting heuristic which produced improvements in the accuracy and the efficiency when it was applied to the corpus of the project in the development stages.

2.3 Machine Learning Classification

The rest of this section will present machine learning algorithms used to classify the polarity (positive, negative, neutral) of the tweets in their normalized form.

The term machine learning refers to the “automated detection of meaningful patterns in data” [5]. Alongside with the growing size of the data produced, machine learning has become a common technique for information extraction. From spam filtering and personalized advertising, to search engines and face detection software, machine learning is applied in a wide range of domains [5]. While the variety of present algorithms depends on the learning task, specialized literature makes the distinction according to the nature of interaction between the computer and the environment. As such, the separation is made between supervised and unsupervised machine learning algorithms:

**Supervised Learning:** In a supervised machine learning algorithm the training data “comprises examples of the input vectors along with their corresponding target vectors (classes)” [6]. For example, in a supervised learning manner a computer can be thought to distinguish between pictures of cats and pictures of dogs. In the training phase, a set of labelled pictures will be processed by the algorithm. At this point, the computer ‘knows’ which pictures contain cats and which contain dogs. When presented with new unlabelled pictures, the algorithm will decide based on what it ‘saw’ before, the type of animal in the picture. Hence, the goal is to ‘learn’ a general rule that maps input to output [6].

**Unsupervised Learning:** Unsupervised machine learning algorithms have the same scope as supervised learning, which is to map input to output. However, the difference is that in the training phase the input is not labelled, consequently, the computer has to find structure in the input, without specifically being told how to classify.

As part of the project, a supervised approach (Figure 2.2) was desirable. Consequently, two algorithms were used, one implemented and the other taken from a pre-implemented library which serves as a comparison base for the evaluation process. The rest of this chapter will explain in detail the Naïve Bayes algorithm (implementation is explained in Chapter 4), and offer a brief description of the Support Vector Machine algorithm.
2.3.1 Naïve Bayes

Naïve Bayes is a supervised machine learning algorithm. It is widely recognized as one of the “most efficient and effective learning algorithms for data mining” [7]. The classifier is build using Bayes’ theorem - Figure 2.3, with independence assumptions. As such, the classifier assumes that the effect of a predictor \( x \) over a given output class \( y \) is independent of the value of other predictors.

![Bayes' Theorem Diagram](image)

\[
P(c | x) = \frac{P(x | c) P(c)}{P(x)}
\]

\[
P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \cdots \times P(x_n | c) \times P(c)
\]

*Figure 2.3: Bayes’ Theorem [9]*

- \( P(c|x) \): posterior probability of the target(c) given a predictor (x)
- \( P(x|c) \): probability of a predictor given a class(c)
- \( P(c) \): prior probability of a class
- \( P(x) \): prior probability of a predictor
Also known as class conditional independence, the aforementioned assumption is often seen as a drawback in the accuracy of the algorithm. A number of heuristics meant to handle this challenge will be presented in the Implementation chapter.

To exemplify the formula, consider the previous example with images of cats and dogs. In this case, the classifier has to decide between two classes. Classifying a new image is the equivalent of comparing between the two probabilities: \( P(\text{cats} \mid \text{new\_image}) \) and \( P(\text{dogs} \mid \text{new\_image}) \). These values will be computed based on the above formula. In the computation, the images used in the training phase of the algorithm will be used to compute the probability of the two classes, the prior probability of the predictors, and the probability of a predictor given the two classes.

2.3.2 Support Vector Machine (SVM)

Support Vector Machine is another example of supervised algorithm. While Naïve Bayes uses a probabilistic model, SVM is used to map the input to a high dimensional plane. A line representing the hyperplane will separate the classes, so that the distance between the classes and the hyperplane is maximized. When new examples are added to the model, the algorithm outputs an optimal hyperplane to categorize them. Figure 2.4 is an example of a SVM algorithm hyperplane [27].

![Figure 2.4: SVM - two classes example [10]](image)

When no such line can be established, a kernel function is used to map the data into a higher dimensional plane, where the data can be separated by its component classes. When visualised in a lower dimensional plane, the line separating the classes is curved.
Chapter 3 - Design

Overview: This chapter describes the design patterns used during the development of the project. It presents a high-level view of the system, what methodology was applied, and which third party technologies have been adopted to deliver the final product. More details about implementation will be demonstrated in Chapter 4.

3.1 Why Python?

Given the context of processing large amounts of data, memory management became prioritary. Consequently, a programming language which is capable to handle processing and storage of these amounts of data was imperative. An iterative system approach, when processing a list of items, has to firstly store it, which requires memory. In these regards, Python provides generators, which are particularly useful when processing large amounts of data, passing the source data through the processing chain, one item at a time, storing only the results of the processing chain [11].

Considering the above argument, Python proves capable of efficiently managing memory, a task crucial for the ‘real time’ component of the project. A drawback of Python is that it is an interpreted language, which by contrast, is slower than compiled languages (such as C, Java, etc.). The developers community considered the disadvantage, and proposed different ways to improve Python’s speed. As such, projects like Numba and PyPy are viable solutions. The creator of Python, Guido van Rossum recognizes the improvements added to Python and states that PyPy is the best way to obtain high-performance systems while using Python [12].

Furthermore, advanced libraries for data processing such as NumPy and SciPy were developed by the scientific community and domain experts [29]. Such tools proved to be helpful during the development stage, and reinforced the reasoning of choosing Python.

3.2 System Architecture

In order to perform sentiment analysis, data manipulation is required via a processing chain. On this matter, in the early stage of the project, a support module was developed. The support module, referred from now on as the pipeline, had to be capable of integrating and testing the following components:
Together, the pipeline and the aforementioned components form the engine of the system. One of the early objectives of the project was developing a working model of the engine. As a consequence, the user interface was produced in later development stages. The architectural overview is illustrated in Figure 3.1. The following subsections will cover in detail the design and the requirements gathering for each component of the system.

![Figure 3.1: Architecture overview](image)

### 3.3 Methodology

The project’s design and development stages were completed in small iterations. This is a core part in the agile methodology which was extensively applied throughout the project. The time available for development was divided into smaller chunks resembling the iterations, each having a deadline and a set of tasks to be completed [15]. A mini-plan covering a summary for each iteration can be found in Appendix A.

Furthermore, GitHub was used as a version control platform. At the end of each iteration a new branch containing the tasks done was created. If the code passed the tests, the current branch was merged with the master branch. If the code did not pass the tests, the code remained unmerged until errors / conflicts were resolved. To help with the progress tracking,
JIRA, a third party software offering an implementation of the “Tasks Board” agile practice, was used.

3.4 Requirements gathering

Requirements, both functional and non-functional, were gathered at different stages during the development process, contrary to a classical up front approach. However, defining a base set of requirements and expected behavior was necessary to begin development. As such, the engine had to be scalable, allowing integration of new modules without affecting its performance. In addition, a standard structure of the processed data was defined, using JSON (JavaScript Object Notation) formatting, where data objects were stored using attribute-value pairs.

A non-functional requirement prior to the development stage, was researching the state of the art in sentiment analysis. From this perspective the accuracy of the algorithms used for classification, the flexibility of implementing heuristics on such algorithms, and the time required for implementation and testing were assessed. For example, a system using Neural Networks requires more time for implementation compared to one where Naïve Bayes is used. In addition, Neural Network algorithms require in depth understanding in order to perform heuristics, while a probabilistic model like the Naïve Bayes algorithm is more flexible [13]. More details on how supervised machine learning algorithms are compared to each other in specialized literature will be presented in Chapter 5.

3.4.1 Engine Requirements

Most of the requirements for the engine are of a functional nature. Hence, heuristics on improving the machine learning algorithm were required, as well as the use of third party software. An initial list of functional requirements ranked by priority and number of hours required for completion is presented in Table 3.1.

In order to achieve the real time behavior desired, a reliable Twitter stream software had to be used in the development of the engine, as Twitter does not provide a native Python API. In these regards, a “pure Python wrapper for Twitter API” [14] - Twython was employed. As there are a number of Python wrappers available for Twitter’s API (e.g: Tweepy, Twitter Search, Birdy, etc.), the feature which made Twython most appealing was the ability of supporting a multi-threaded implementation. This way, the application supports more than one user at a time, and also to manage multiple requests from the same user without having to run multiple instances of the engine [14].
<table>
<thead>
<tr>
<th>No.</th>
<th>Functional Requirements</th>
<th>Priority</th>
<th>Hours</th>
<th>Development Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Train the main classification algorithm</td>
<td>Very High</td>
<td>15-20</td>
<td>Early</td>
</tr>
<tr>
<td>2</td>
<td>Output and store the model after the training phase of the classification algorithm</td>
<td>Very High</td>
<td>5-10</td>
<td>Early</td>
</tr>
<tr>
<td>3</td>
<td>Test the classification algorithm proposed</td>
<td>Very High</td>
<td>15-20</td>
<td>Early</td>
</tr>
<tr>
<td>4</td>
<td>Clean the noise from tweets retrieved and address orthography issues</td>
<td>High</td>
<td>10-15</td>
<td>Middle</td>
</tr>
<tr>
<td>5</td>
<td>Retrieve a stream of tweets (domain specific for specified topics) for the long-term</td>
<td>Very High</td>
<td>10-15</td>
<td>Middle</td>
</tr>
<tr>
<td></td>
<td>components within intervals of: 10 minutes, 2 hours, 2 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Store the acquired data depending on the component in which the engine is used</td>
<td>High</td>
<td>10-15</td>
<td>Middle to Late</td>
</tr>
<tr>
<td>7</td>
<td>Train an additional machine learning algorithm for comparison with the main algorithm</td>
<td>Low</td>
<td>12-15</td>
<td>Late</td>
</tr>
<tr>
<td>8</td>
<td>Based on the selected topic, retrieve a stream of tweets for an undetermined period</td>
<td>High</td>
<td>5-10</td>
<td>Late</td>
</tr>
<tr>
<td></td>
<td>of time (until user exits the system)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Engine - functional requirements

Another important design problem was data storage. Here, two database architectures were considered: a lightweight database - Redis - using a file system stored in RAM, and a relational database - PostgreSQL.

Furthermore, the engine has to support both the real time component, and the long-term sentiment analysis component. Consequently, Redis was the first choice, as it proved to be an efficient solution for both temporary and permanent storage, being particularly useful at caching operations in the long-term component.

A more detailed list of non-functional requirements is outlined in Table 3.2.
Table 3.2: Engine - non-functional requirements

<table>
<thead>
<tr>
<th>No.</th>
<th>Non-functional requirements</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Extensibility - the engine should be able to implement new modules without performance issues</td>
<td>Very High</td>
</tr>
<tr>
<td>2</td>
<td>Efficiency - the engine should be able to support large intakes of data without affecting its performance</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Speed - The engine should perform sentiment analysis in seconds from the time the request was made</td>
<td>Very High</td>
</tr>
<tr>
<td>4</td>
<td>Robustness - the system should be able to run multiple instances of engine</td>
<td>Medium</td>
</tr>
<tr>
<td>5</td>
<td>Scalability - the engine should scale the output size based on the size of the input</td>
<td>Very High</td>
</tr>
</tbody>
</table>

3.4.2 Graphical User Interface (GUI)

As one of the main objectives of the project is performing highly accurate sentiment analysis, the graphical user interface was not within the initial scope. However, a graphical user interface turned out to be imperative, such that users can interact with the engine in a more intuitive manner. As such, a set of functional requirements was developed in the GUI as evidenced in Table 3.3.

Table 3.3: GUI - functional requirements

<table>
<thead>
<tr>
<th>No.</th>
<th>Functional Requirements</th>
<th>Priority</th>
<th>Hours</th>
<th>Development Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Include textual input field on which sentiment analysis is performed</td>
<td>Very High</td>
<td>5-10</td>
<td>Late</td>
</tr>
<tr>
<td>2</td>
<td>Output stream chart in real time</td>
<td>Very High</td>
<td>20-25</td>
<td>Late</td>
</tr>
<tr>
<td>3</td>
<td>Display last text received and average sentiment score in panel nearby chart</td>
<td>Very High</td>
<td>15-20</td>
<td>Late</td>
</tr>
</tbody>
</table>

The non-functional requirements for the graphical user interface were set following three principles: responsiveness, readability, and usability. As such, Table 3.4 presents the non-functional requirements developed for the graphical user interface.
<table>
<thead>
<tr>
<th>No.</th>
<th>Non-functional requirements</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The graphical user interface should have a quick response time for user interactions</td>
<td>Responsiveness</td>
</tr>
<tr>
<td>2</td>
<td>The graphical user interface should present an intuitive design.</td>
<td>Readability</td>
</tr>
<tr>
<td>3</td>
<td>The graphical user interface should output the text body of the tweets analysed</td>
<td>Usability</td>
</tr>
<tr>
<td>4</td>
<td>The graphical user interface should update in real time the aggregate sentiment score of the tweets analysed</td>
<td>Usability</td>
</tr>
</tbody>
</table>

*Table 3.4: GUI - non-functional requirements*

An important challenge was to make the interface robust, so that a user can follow the sentiment evolution of two or more topics at the same time. This was accomplished using Flask - a third party micro framework in Python, which offers a build-in development server with restful request dispatching [20]. As such, every time the user places a request for a topic, a new URL which contains the name of the topic is generated. For example, if the topics of interest are “Brand A” and “Brand B”, two charts will be displayed in the same window, resembling real time sentiment analysis for those, with the following URL address: “http://localhost:8000/Brand_A+Brand_B”
Chapter 4 - Implementation

Overview: This chapter follows the approach described in the Design section, making a clear distinction between the back-end (the engine of the system) and front-end (the graphical user interface). More in depth technical details will be presented for both of the components, as well as how third party software was employed in development.

4.1 Engine

The engine was built using “the pipeline” (introduced in Chapter 3) and it had to adapt for two purposes. One is to serve the real time interface (described in future subsections). The second purpose is to be run continuously in a pre-established interval of time in order to perform in depth sentiment analysis for a spectrum of brands and topics. This is reflected in the separation in two modules of the data gathering component.

![Class diagram representation](image)

*Figure 4.1: Real time component class diagram representation*
The engine uses the pipeline to add a series of functionalities to the processing chain. The following components build up the final version of the engine:

1. Data Gathering Modules
2. Data Filtering Modules
3. Sentiment Classification Module
4. Association Rules Modules

The rest of this subsection will be focused on explaining the implementation steps and technicalities of both, the pipeline and the aforementioned modules. A simplified class diagram of the real time system is presented in Figure 4.1.

4.1.1 The Pipeline

The pipeline had to provide an environment ready to process data, while integrating the other modules. Furthermore, the pipeline had to be flexible so that it could easily adapt data streams. One common issue when processing data streams with auxiliary use of data structures, is that the random access memory space available might be exceeded. This can lead to system failures which are least desirable. Relevant to this issue, was a built-in feature of Python: functions generators. Instead of returning a set of values stored in a data structure, a function can be written so that it generates the values as they are needed by other components in the processing chain [30].

While using generators presents a set of advantages, there is one drawback to be mentioned. Once data was generated for usage it can only be called once. Not as bad as it sounds, this obliged the code to be written in a careful manner by avoiding redundancies, auxiliary variables and data structures, in order to preserve the efficient memory use.

Another important feature of the pipeline is to ensure the suitability of an object for the purpose it was initially designed for. From this perspective, the pipeline had to serve data structures from one module to another, while ensuring that those objects preserved their properties.

The end result of the pipeline is a class which once instantiated serves as support for the other modules in the processing chain. Figure 4.2 is an example of the pipeline usage and how modules are added to it. The first step is to create an instance of the pipeline class. Once done, this instance can incorporate other modules (e.g.: line 7 to line 12 in the code snippet). After the modules are added using the add_step method, the run method is called on the pipeline in order to start the processing chain.
# Coverage Selection Parameters

```python
min_cov = float(sys.argv[1])
max_cov = float(sys.argv[2])
```

# Training

```python
pipeline = MFPipeline()
engine = pipeline.add_step(MFTwitterSentiment140Loader('training_data'))
engine = pipeline.add_step(MFTokenize5())
engine = pipeline.add_step(MFTokenAddBigrams())
engine = pipeline.add_step(MFNegationHandling())
engine = pipeline.add_step(MFCoverageSelectionTool(min_cov, max_cov))
engine = pipeline.add_step(MFSentimentTrainModel('trained_model.csv'))
```

# Run

```python
pipeline.run()
```

*Figure 4.2: Pipeline experiment - Naïve Bayes training phase*

## 4.1.2 Data Gathering

The main data source used throughout the project is the online social networking service, Twitter. Two data gathering modules were implemented using Twython (introduced in Chapter 3). One component was designed to be used for the real time interface. Once a request is placed, it will further send the request to Twython, which in turn will reply with a continuous stream of tweets.

The other component collects data within a pre-established interval of time. As marketing is concerned with development and implementation of a promotional strategy, the time factor had to be carefully considered when performing analysis. Ten minutes worth of data on a campaign might not justify the success rate of it. Considering the amount of time available for the development of the project, waiting an entire week to perform analysis over a set of data was not feasible either. In order to meet the requirements, while having enough time to test and adjust the sentiment analysis process, the second module was used to store data within the following intervals: ten minutes, one hour, and two days. This way, adjustments of other components is possible in an efficient amount of time, while the amount of data analysed will deliver an accurate success estimation of a campaign.

Furthermore, an auxiliary module was developed to format the labelled data obtained from a third party source - “Sentiment140” [16].

23
4.1.3 Data Filtering I

For data filtering one module was built, which performs tokenization splitting textual input in tokens (words). The tokenization module implements auxiliary methods to detect and discard tweets which are not entirely made out of English characters (e.g.: Japanese, Hiragana, Katakana, etc.). However, tweets containing other symbols (e.g.: mentions marked with the symbol “@” and hashtags, marked with the symbol “#”) are filtered in a separate JSON field.

4.2 Classification - Naïve Bayes Algorithm

Once data was acquired and filtered, the next step was the implementation of the classification algorithm, the main module of the engine. As outlined in Section 3, the module implemented the Naïve Bayes algorithm, which consists of two phases: training and testing. In the training phase, labelled data acquired from Sentiment140 was used. As such, 1.2 million tweets out of the 1.6 million size of the corpus was used in training. The output is a model consisting of token-value pairs, where the values are: number of occurrences over the entire dataset, number of occurrences in positive labelled tweets, and number of occurrences in negative labelled tweets. This is exemplified in Table 4.1.

<table>
<thead>
<tr>
<th>Token</th>
<th>Positive counts</th>
<th>Negative counts</th>
<th>Total counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>more</td>
<td>11426</td>
<td>11320</td>
<td>22746</td>
</tr>
<tr>
<td>fleet</td>
<td>16</td>
<td>61</td>
<td>77</td>
</tr>
<tr>
<td>whole</td>
<td>17</td>
<td>12</td>
<td>29</td>
</tr>
<tr>
<td>woods</td>
<td>97</td>
<td>75</td>
<td>172</td>
</tr>
<tr>
<td>spiders</td>
<td>33</td>
<td>88</td>
<td>121</td>
</tr>
<tr>
<td>like</td>
<td>1087</td>
<td>542</td>
<td>1629</td>
</tr>
<tr>
<td>woody</td>
<td>26</td>
<td>15</td>
<td>41</td>
</tr>
<tr>
<td>loving</td>
<td>931</td>
<td>12</td>
<td>943</td>
</tr>
<tr>
<td>sigh</td>
<td>8</td>
<td>105</td>
<td>113</td>
</tr>
</tbody>
</table>

*Table 4.1: Sample of the model obtained in the training phase of the classification*

In the testing phase, unseen labelled data and the probabilistic model produced in the training phase were used to measure the accuracy of the algorithm. Consequently, the algorithm produced a classification accuracy of 71%. For the development stage when it was implemented, such accuracy was expected, yet further improvements were required.
As the data set used in training had labels only for positive and negative tweets, yet the goal is to distinguish between the three: positive, negative, neutral, further heuristic was required. In addition, a sentiment evaluation scale: -1 (negative) to 1 (positive) was introduced, where tweets of which sentiment score was in the range of: [-0.05, 0.05] were considered neutral. The limits of the range were established after a manual inspection of the tweets analysed.

The following subsection describes the implementation of heuristics to improve the performance and the classification accuracy of the algorithm.

### 4.2.1 Data filtering II

Later in the development, as a solution to improve the accuracy of the classifier, another filtering module was build to be used in the training stage of the classification. The problem identified was the large number of tokens produced by the tokenization module, which equaled almost 1 million. This made the model to be computationally expensive, reducing the speed performance of the algorithm. In addition, two approaches were considered:

1. Using Porter’s Stemming algorithm to perform word reductions.
2. Excluding tokens with low sentiment value.

While Porter’s algorithm reduced the corpus size to 60% of its initial size after tokenization, this was not sufficient, as a significant number of the tokens left, held no particular sentiment value. Verifying manually which tokens were prevalent in positive or negative contexts was not a solution, due to the high amount of manual work. An interesting new heuristics using the pareto principle (explained in Chapter 2) was applied. Looking at the tokens distribution of the training model - Figure 4.3, it can be noticed that words such as “I” and “the” have the highest occurrence throughout the whole input data. At the opposite end, are words which albeit, they might present some sort of sentiment value, the number of total occurrences in the corpus is low, consequently the impact over the module is almost irrelevant.

However, when separated from their context, those words do not hold any sentiment value. By contrast, words situated on the curve’s slope in the figure, appeared to have a strong sentiment value (e.g.: “love”, “hate”, “amazing” etc.). Following the pareto principle, the corpus was cut to 20% of its size, so that only the tokens with a high sentiment value were kept. A manual inspection was required to determine where tokens with low sentiment value start to appear in the distribution, in order to determine the cut points. As a result, the final size of the model was reduced down to 40,000 tokens. Consequently, the accuracy was improved by 7% as a result of noise (unwanted data) removal.
4.2.2 Association Rules

Another common improvement in computational linguistics is the use of n-grams (introduced in Chapter 2) for which a module was created. However, implementing n-grams in the form of bigrams and trigrams, without considering only relevant associations, increases the size of the corpus and also adds noise to it. As such, associations between nouns, adjectives and verbs were done using a third party software, polyglot, which offered a part of speech tagging tool [18].

As a result of adding bigrams the overall accuracy of the classifier grew by 2.9% up to 80.9%.

In addition to n-grams, a separate module was build to handle negation. A classical approach in specialized literature is to add artificial words: “if a word x is preceded by a negation word (e.g: not, don’t, no), then rather than considering this as an occurrence of the feature x, a new feature (token) NOT x is created” [4]. For example after performing negation handling the sentence: “I do not like this new brand” will result in the following representation: “I do not NOT_like, NOT_this, NOT_new, NOT_brand”. The advantage of this feature is that the plain occurrence and the negated occurrence of the word are now distinguishable in the corpus. In addition, the accuracy of the classifier was improved by 1.8% up to 82.71% overall accuracy (more details are presented in Chapter 5).
4.3 Graphical User Interface

As described in Chapter 3, the graphical user interface was implemented for the real time component. In order to add interactivity to the system, the language of choice to build the front-end was JavaScript. Consequently, a third party library - D3JS, was used to build interactive charts. Albeit there is a variety of templates already available within the library, an implementation from scratch was necessary to meet the requirements of the system. As such, the visual component was challenging due to the lack of previous experience in JavaScript (more details in the Conclusion chapter). Captures of the graphical user interface are presented in Appendix B.
Chapter 5 - Testing

Overview: This chapter presents the testing methodologies that were applied during development, under the test driven development recommendations [26]. As such, unit testing was exhaustively used for each component module of the engine and the graphical user interface. In addition, end to end testing was performed on the two major parts of the system: the real time component and the long-term component.

5.1 Unit Testing

For the engine, unit testing was done in a systematic manner, each module being tested during or immediately after it was developed. In these regards, “pytest” - a third party library for Python was used. Figure 5.1 is the example code comprising of one test case for unit testing the tokenizer module. An expected behaviour is defined and tested against the input. Consequently, all the modules developed were tested using the same format [28].

```
# define test - ignore URLs:
# test_expected_output = ["i", "wish", "i", "go", "to", "mars", "spacelife"]

def tokenizer_test():
    assert f(test_input) == test_expected_output

if tokenizer_test():
    print "tokenizer test pass "
else:
    print "tokenizer test fail"
```

Figure 5.1: Unit testing (engine) - tokenizer module

This approach was useful in discovering errors which affected the engine. In addition, Table 5.1 presents a set of malfunctions were found for the modules tested. The ordering in table was done based on the error prone of each module. As such, the stream module and the graphical user interface required a significant amount of time for debugging compared to the other modules. While the GUI seemed challenging due to the lack of previous knowledge in JavaScript, the stream module was affected by the use of the Twython API which for the purpose of the project, required a multi-threaded implementation.
<table>
<thead>
<tr>
<th>No.</th>
<th>Module</th>
<th>Source</th>
<th>Error-Behavior</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream</td>
<td>Twython</td>
<td>Module crashed after retrieving the first tweet. Solution: multi-threaded implementation of the module, allowing a</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Data Filtering</td>
<td>Negation Handling</td>
<td>Unexpected behavior reflected in the double negations found in the corpus. Solution: merge negation handling and bigrams module in one.</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>GUI</td>
<td>Flask</td>
<td>Failed to load two requests from the same user, due to a typo in the routing function.</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>GUI</td>
<td>D3JS</td>
<td>Chart crashed after a few minutes of displaying live content from twitter. The cause was the excessive number of SVG elements displayed, due to the lack of experience in JavaScript. Solution: reduced the number of SVG elements.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1: Sample of unit testing form

5.2 System Testing

While most of the erroneous behavior was discovered and solved using unit testing, end to end system testing was necessary to establish the reliability and the performance of the system as a whole, so that interaction between modules is as expected. In addition, end to end testing was required to ensure that the tests employed cover a significant amount of the code produced [29].

For the engine, the pipeline’s debug functionality indicating which module has caused the system to crash, proved to be very useful. As such, the filtering module caused the stream module to crash and as a result, the whole system stopped functioning. This was a time consuming error, caused by polyglot, which mistakenly labelled tweets, by applying a built-in language detection feature over the one Twython performs. In addition, the error text displayed by polyglot was ambiguous, which required manual inspection of different features employed by polyglot, in order to detect the source of the problem.
Chapter 6 - Evaluation and Results

Overview: In this chapter a detailed evaluation of the engine is provided, as well as an accuracy comparison is between the main classification algorithms implemented - Naïve Bayes, and the SVM algorithm. In addition, the achievements recorded after the engine was deployed to serve the marketing industry are presented.

6.1 Evaluation

In order to evaluate the sentiment analysis accuracy of the engine produced, k-fold cross validation was used. As such, the technique requires splitting the dataset used in the training and the testing phases, in k equally sized subsets. Each one of the k subsets is then used in the testing phase, while the other k-1 are used in the training phase of the algorithm. The process employs a total of k such iterations. Then, the average accuracy of the k steps is calculated [21]. Table 6.1 presents the evaluation’s results of the engine using k-fold cross validation. The dataset used contains 1.6 million labelled tweets, and the k’s value was set to 10, a value commonly used in specialized literature [22]. Consequently eight subsets, each containing 160,000 tweets, were produced.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Training subsets</th>
<th>Testing subset</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1, s2, s3, s4, s5, s6, s7, s8, s9</td>
<td>s10</td>
<td>83.67</td>
</tr>
<tr>
<td>2</td>
<td>s1, s2, s3, s4, s5, s6, s7, s8, s10</td>
<td>s9</td>
<td>82.54</td>
</tr>
<tr>
<td>3</td>
<td>s1, s2, s3, s4, s5, s6, s7, s9, s10</td>
<td>s8</td>
<td>84.79</td>
</tr>
<tr>
<td>4</td>
<td>s1, s2, s3, s4, s5, s6, s8, s9, s10</td>
<td>s7</td>
<td>80.44</td>
</tr>
<tr>
<td>5</td>
<td>s1, s2, s3, s4, s5, s7, s8, s9, s10</td>
<td>s6</td>
<td>84.65</td>
</tr>
<tr>
<td>6</td>
<td>s1, s2, s3, s4, s6, s7, s8, s9, s10</td>
<td>s5</td>
<td>79.1</td>
</tr>
<tr>
<td>7</td>
<td>s1, s2, s3, s5, s6, s7, s8, s9, s10</td>
<td>s4</td>
<td>84.01</td>
</tr>
<tr>
<td>8</td>
<td>s1, s2, s4, s5, s6, s7, s8, s9, s10</td>
<td>s3</td>
<td>81.56</td>
</tr>
<tr>
<td>9</td>
<td>s1, s3, s4, s5, s6, s7, s8, s9, s10</td>
<td>s2</td>
<td>83.22</td>
</tr>
<tr>
<td>10</td>
<td>s2, s3, s4, s5, s6, s7, s8, s9, s10</td>
<td>s1</td>
<td>82.18</td>
</tr>
<tr>
<td>Average:</td>
<td></td>
<td></td>
<td><strong>82.71</strong></td>
</tr>
</tbody>
</table>

Table 6.1: Crossfold validation results - Naïve Bayes model

6.2 Classifiers Comparison

In the initial stages of the project, research in the field of machine learning was imperative to decide which classification algorithm was best suited for implementation [24][25]. Consequently, in the final version of the engine, the Naïve Bayes algorithm (NB) was
implemented, whilst the second choice remained the Support Vector Machine algorithm (SVM). *Table 6.2* shows an accuracy comparison between the two algorithms, based on the features employed.

<table>
<thead>
<tr>
<th>Naïve Bayes Accuracy (%)</th>
<th>SVM Accuracy (%)</th>
<th>Unigrams</th>
<th>Bigrams</th>
<th>Negation Handling</th>
<th>Coverage Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>70.2</td>
<td>69.9</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>76.67</td>
<td><strong>78.34</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>80.9</td>
<td>79.7</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>82.71</td>
<td>81.4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Table 6.2: Comparison between Naïve Bayes and Support Vector Machine*

### 6.3 Achievements

Albeit, the evaluation’s result of the engine is at a high standard, considering a wide number of topics employed in the dataset used for evaluation, this is not the only feature indicating the overall achievements of the project. As such, the engine was adopted by a Manchester based company - Metafused Limited, where the long-term component of the system, with time intervals set for 10 minutes, was used to perform sentiment analysis on four major topics: sports, music, television, and fashion. One noticeable achievement, alongside the accuracy of the engine, relies on its efficiency and speed. This is reflected in the engine’s benchmark with similar software, used by other companies to collect and classify the latest news for sports related topics (e.g. football, basketball, rugby). *Figure 6.1* presents the aforementioned benchmark performed by Metafused (BrandFlo platform) in December 2015, when Chelsea Football Club dismissed their manager, fact which stirred interest on social media platforms [23].

*Figure 6.1: Benchmark - Metafused Limited [23]*
Chapter 7 - Conclusion

Overview: This is a reflective chapter, where the completion of the objectives set at the start of the project is assessed. In addition, future work is proposed and the knowledge gained while working on the project is demonstrated.

7.1 Objectives and Timing

The main objective set for the project was to develop a system which can perform real time sentiment analysis on social media sources. As evidenced in the Evaluation chapter, the engine employed in the system was used in the marketing industry, to monitor and report sentiment analysis on a series of topics. From a timing perspective the development of the engine in December 2015, which is in line with the timeline set (presented in Appendix A).

Furthermore, the real time component of the system was produced and natural language processing techniques (tokenization, stemming, part of speech tagging, n-grams) were employed to improve the accuracy of the engine. Consequently, the evaluation placed the classification accuracy at 82.71%, which surpassed the initial objective.

Albeit, the development of the graphical user interface for the real time component was set as a medium priority objective, it was considered to be imperative for the user’s experience. As such, users can interact with the engine in a more intuitive manner, whilst their understanding is augmented with visual representations of the data analysed. This objective was met as part of the last milestone in the project.

7.2 Future Work

Future work to enhance the capabilities of the engine produced includes the development of a more complex graphical interface. By doing so, when the user inputs a topic, the interface will also display the sentiment classification performed on past data, as a second metric.

Another future improvement relies on the implementation of a module to retrieve the content of the URLs discovered in the tweets analysed. While analysing the corpus, a significant number of URLs which point to Instagram - a social media platform - were found. Besides Instagram, the engine will include modules to extract information from other social media platforms: Facebook, which provides a comprehensive Python API, and YouTube. Additionally, a more complex spectrum of sentiment values will be introduced (e.g., joy, anger, hope, etc.)
7.3 Reflection and Knowledge Gained

By designing and developing this project, my overall knowledge and skills in computer science were significantly improved. As such, a better understanding of the machine learning field was gained, while implementing the classification algorithm and the subsequent heuristics to improve its accuracy. In addition, fundamental knowledge in the field of natural language processing was acquired.

Furthermore, an addition to the set of programming languages I came across is JavaScript. I am confident now, that I can further enhance interaction between users and future systems developed, by relying not only on the intrinsic value of the artefacts produced, but also on the versatility of JavaScript and the native visual frameworks available. Within the same note, the previously acquired Python skills were sharpened, by mastering advanced concepts such as generators and decorators [30].

From the perspective of the application domain - online marketing industry - a number of lessons were drawn. Deploying the software in industry, has shown that a sentiment analysis tool is fundamental for the success of a company’s marketing campaign, yet a more specific, topic oriented, product is desirable.

Consequently, if I were to start the project again, I would consider a more exploratory approach, in order to establish what are the prioritary topics on which sentiment analysis will be performed. However, the deliverables would include similar functionalities. From an implementation perspective, the benefit of developing the classification algorithm from scratch was reflected by the full control over the engine’s capabilities. The pipeline produced was helpful to keep an organized manner of writing code and an useful debugging tool. However, a second iteration of the project will employ a more in depth research of the algorithms available for classification, deprioritizing the pipeline for later stages of the development.

7.4 Conclusion

Considering the above reflection on the project’s planning, it can be concluded that the scope of the project was met within the timeline set. As such, the desired classification accuracy of 80% or more is reached. The reporting is performed in real time and the graphical user interface offers an intuitive, yet comprehensive user interaction. Overall, the project was a good opportunity to further hone my programming skills and expand my knowledge in the fields of natural language processing and machine learning.
REFERENCES


APPENDIX A: Timeline chart - initial plan

Figure A.1: Timeline chart - initial plan
APPENDIX B - Graphical User Interface

In this appendix screen captures of the graphical user developed are presented. Each screen capture contains a short description.

**Figure B.1:** Graphical User Interface - The engine performs sentiment analysis on the topic selected - ‘Nike’. The chart contains two lines:
- orange indicating the sentiment value of the currently analysed tweet
- blue indicating the average sentiment for all the tweets analysed since the request was made
Figure B.2: Graphical User Interface - The engine outputs, at the top of the website, a negative sentiment value for the input: ‘I don’t like swimming’.