Imperial College London
Department of Computing

Sentiment Analysis
-
A multimodal approach

by

Lucas Carstens

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science, Specialism Artificial Intelligence, of Imperial College London

September 2011

Abstract

Sentiment Analysis describes a Natural Language Processing problem that attempts to differentiate opinionated text from factual text and, in case of opinionated text, determine its polarity. This report presents A-SVM, a system that tackles the discrimination of opinionated text from non opinionated text through means of both Machine Learning techniques and arguments acquired via user feedback. The system has been used to investigate the potential merits of approaching Sentiment Analysis in a multi faceted manner by comparing straight forward Machine Learning techniques with this multimodal system architecture. All evaluations were executed using a corpus of annotated text, purpose built during the project, and its classification performance was compared to the performance of Support Vector Machines, which also constituted an integral building block of the system itself. The classification of a test set of approximately 12,000 words yielded an increase in classification precision of 5.6%, from 78.1% precision using Support Vector Machines to 83.7% when classifying the same test set with the system developed in this project.

My greatest thanks belong to Dr. Francesca Toni who offered supervision, guidance and encouragement whenever needed and beyond.

Also, my deep appreciation to my parents who have made it possible for me to embark upon the challenge and adventure that was Imperial College, London.

No less, I would like to thank Dr. Krysia Broda, Tobias Gierk, Ulrich Schaechtle and everyone else who has made a contribution to this project in one way or another.

Contents

1

Introduction

Language enters and profoundly shapes our lives every day, and as it does it comes in the most different guises. We lead conversations, read newspapers, listen attentively to what our lecturer, superior, or colleague has to say, and, today more than ever, we turn to electronic and interactive media to supply us with information about every fathomable topic of potential interest. The continuous evolution of modern technology has shaped the way we acquire and deal with information throughout the past decades and it has brought along with it a vast number of challenges, with ever new issues arising as these technologies develop further. Today, web connectivity has reached such a pervasive level that more than 51 Million UK citizens, over 82% of the entire population, are connected to the internet [10]. This means that we are generally able to access information about any topic, at any time and any place at our own convenience. Someone making use of this connectivity not only has access to information, but at the very same time is able to contribute and spread his or her own knowledge, wisdom, and opinions through various channels. This ever growing user-generated content within the realms of the World Wide Web offers unprecedented opportunities in dealing with information, but just the same, the immense pool of content emerging from such a collaborative effort carries significant implications when it comes to putting this content to use.

It has been the aim of this project to tackle one specific aspect of language and how it is used, and utilise this within the setting of dealing with content distributed over the internet: **Opinions**.

Expressing opinions and sentiment towards an object, an individual, an idea, or an opinion itself constitutes one of the prime purposes language fulfils and today, websites allow the user to find an opinion on virtually anything. Despite, or maybe because of this fact, it is very challenging to process these opinions in a structured manner, be it manually or computationally. This is rather intuitive for two reasons: For one, the distribution of opinions throughout the World Wide Web, as is the case with any other content, rarely occurs within truly structured settings and if it does, this structure is likely to be maintained only within the domain the opinion is published. Secondly, it would be hard to imagine how to reliably deduce an opinion and its content from text at sheer syntactic level. Going beyond syntax in text is an issue when it is done computationally, because the ability to digest semantics, i.e. the meaning (here: the opinion) of text usually requires some kind of actual understanding of its contents. A definition of understanding analogous to human understanding of language may not seem feasible when applied to a machine today, but nonetheless it is desirable to equip systems with the ability to process text in a way that would normally require such an understanding. In Sentiment Analysis, we attempt to envision and realise such systems, which encompasses a number of subproblems. These are explained in detail throughout the following chapters, but before doing so, the term Sentiment Analysis requires a precise definition, as does "opinion" itself. The WordNet [45] definition for an opinion reads as follows:

1. A personal belief or judgment that is not founded on proof or certainty

2. A message expressing a belief about something; the expression of a belief that is held with

confidence but not substantiated by positive knowledge or proof

For the remainder, though, the following definition is adopted:

3. An opinion denotes a personal, subjective belief, judgement, appraisal, or sentiment towards an entity, an idea or another opinion that can not be proven by facts. It thus constitutes the counterpart to factual information.

This is simply to explicitly include the differentiation between factual and opinionated text. Opinions are the object of investigation within the discipline of Sentiment Analysis. We introduce the following definition of Sentiment Analysis in order to cover the main aspects:

> Sentiment Analysis, or Opinion Mining, describes a Natural Language Processing problem that attempts to differentiate opinionated text from factual text and, once text is assumed to be opinionated, classify it as expressing a negative opinion, a neutral opinion, or a positive one. Another subproblem that closely relates to Sentiment Analysis is that of representing opinionated content in a comprehensive manner.

Traditionally, efforts directed at extracting content from texts have mainly focused on acquiring factual information about various domains, i.e. performing so called Information Extraction (IE) (see, for example, [25] for more on IE). When developing applications for the specific tasks, the difference that makes Sentiment Analysis generally more challenging is the fact that Information Extraction can, in most cases, rely more heavily on syntactical features of the information that is to be gathered. As explained above, due to the nature of the problem, this is generally not applicable to Sentiment Analysis. Syntactical information about the analysed text can mainly serve the purpose of supplying heuristics to enhance an employed technique, but not provide sufficient information on its own. This is explained in detail in section 2.4

The potential applications (see 2.1) of enabling a machine to discover, classify and process opinions and sentiment from text are manifold and recent years have seen a spur of interest in this field. [59] and [40] give thorough surveys of the field. Despite this, the effort directed towards this issue is still very much contained at research level and this project has aimed not at supplying a ready made solution to the problem but rather to propose a novel approach to dealing with some of the arising issues and by doing so furthering the progress in the field. This approach adds logic based arguments to probabilistic methods previously employed to solve Sentiment Analysis tasks. The intuition underlying this approach is that combining substantially different ways of dealing with written language computationally should allow an increase of general performance compared to applying probabilistic methods by themselves. During the project I have developed both an annotated text corpus that was used as training data and a system that classifies text according to its opinionatedness. The text corpus consists of approximately $13,000$ annotated n-grams that are represented as feature vectors (see chapter 5). Comparing A-SVM, the system I have developed, with a straight forward SVM classifier, I achieved an increase in classification precision of 5.6%, from 78.1% to 83.7% (see chapter 7).

1.1 Road Map

I begin by presenting an overview of what the key aspects and main challenges in Sentiment Analysis are and what they entail in chapter 2, accompanied by concrete examples. Chapters 3 and 4 cover further previous works relevant to the project. These are subdivided into Machine Learning approaches to Sentiment Analysis, which constitute the foundation for the project, and approaches to generally combining Machine Learning and Argumentation, which provide insight into some basic principles of such efforts. In addition to that, chapter 4 also spans the bridge from elucidating previous efforts in both fields described to the project itself by drawing on the ideas proposed by the developers of ABML ([47]), one of the few algorithms today that combine Machine Learning and Argumentation. Building upon the previous chapters, the efforts made within this project are laid out in the two subsequent chapters, 5 and 6. First off, the development of the text corpus is described, followed by a detailed description of the system and its building blocks. The second to last chapter (7) reports on the evaluation of the program and the results that sprung from both evaluations conducted, namely a straight forward performance analysis and a user centred evaluation. The final chapter (8) discusses these results and, drawing upon them, presents an account of future challenges that will need consideration when progressing within this field.

2

Sentiment Analysis - The challenges

Sentiment Analysis spans a number of different fields, most notably those of (Computational) Linguistics & Natural Language Processing (NLP) and Machine Learning (ML) & Pattern Recognition. Each of the fields brings with it a number of challenges that need to be addressed when working within Sentiment Analysis. This chapter presents the theoretical background Sentiment Analysis builds upon and what needs to be considered when developing systems in this area. Along with this, a number of concrete applications of the concepts described are mentioned and explained to illustrate the main issues.

2.1 Sentiment Analysis in practice

Before considering the challenges that Sentiment Analysis confronts us with and concrete techniques that have been utilised in this field, let us motivate this discourse by considering a few areas in which Sentiment Analysis has either found concrete applicability or where it may do so in the future. To name but a few, Sentiment Analysis has contributed to the following areas or may do so in the future:

- Structuring (customer) reviews
- Gathering business & governmental intelligence
- Political, or any other, discourse
- Enhancing other text processing technologies.

While the following account will limit itself to these examples, this does not imply absence of other domains and areas in which Sentiment Analysis may find applicability.

Analysing and structuring (customer) reviews has been recognised as one of the main fields to which Sentiment Analysis is able to contribute valuable insight and has thus received recognition by a number of researchers (e.g. [50], [57], [58], [78]). Reviews lend themselves to Sentiment Analysis due to their inherently opinionated nature. Accordingly, numerous research efforts have concerned themselves with the issue of determining the polarity of a review, which is intuitively plausible because generally a review will be either positive or negative rather than presenting an objective account of a product, a movie, etc. The use of performing such analysis on reviews lies in the motivation that normally goes with reading reviews. We access reviews to gather an impression of people's opinions who have watched the movie, used the product or listened to the music we are interested in when consulting reviews. We are, in this scenario, generally less interested in a product description, i.e. the facts, than we are in other individuals' personal accounts. Sentiment Analysis can simplify the process of digesting the wealth of opinions that can be found on most products we might be interested in.

It is not just individuals who consult other customers' reviews of a product to assist their decision in buying something or refraining from a purchase. Generally, most areas of business are interested in reviews and judgements of their products by customers, as well. Gathering business intelligence about customers' satisfaction and dissatisfaction can be a laborious and costly process, thus mechanisms that simplify accessing and evaluating various measures of customer sentiment can be highly valuable, i.e. cost saving, to almost any business. While businesses can be assisted in measuring customer sentiment and assessing it, governmental bodies may have similar interest in measuring various indicators of satisfaction and dissatisfaction that may be extracted from news websites, blogging sites, micro blogging sites and other web presences. Twofold benefits of systematic Sentiment Analysis for governmental purposes come to mind. Firstly, having a clear picture of public sentiment as expressed throughout the World Wide Web may allow more targeted policy making, campaigning, public relation handling etc. Secondly, effective Sentiment Analysis may contribute to finding hostile exchanges that could prove harmful to the public's interest and safety.

Closely related to the gathering of government intelligence is the issue that is of focus in this project. It is not just governmental bodies who have an interest in gathering and disseminating sentiment with regards to political developments and issues related to them. This interest is shared by individuals who follow political developments every day, accessing the most varied sources. Reading the news is a process of accessing both factual and opinionated contents. The simplest distinction that can be made here is between the content of regular reports and editorial ones. Where a news report on events is generally of a non opinionated nature, editorial contributions to news reflect an individual's sentiment towards the issue he or she writes about. Nevertheless, either of these types of news articles will generally contain both factual information and opinions. If this were not the case, an analysis of the opinionatedness of news articles would be rendered futile because it would suffice to analyse editorial articles with regards to their opinion polarity. Since such a clear cut distinction is hardly feasible, especially when performing Sentiment Analysis below document level, analysing the opinionatedness of the constituents of news articles can provide valuable information for subsequent analysis tasks.

An integration of Sentiment Analysis as a sub-component technology has proven to be another area where systems, their processes and their output can improve by analysing the sentiment of text that is processed. One of these applications can be found in traditional Information Extraction (IE). As pointed out in [57], expanding successful IE algorithms with Sentiment Analysis algorithms can further improve the performance of such systems. Considering the previous description of analysing news articles, enhancing search algorithms used by search engines with sub-components and selection criteria that draw on Sentiment Analysis may improve search procedures for certain domains, such as the ones discussed in this section.

2.2 Linguistics and Natural Language Processing

2.2.1 Linguistics

> [The field of linguistics is concerned with] the systematic study of the structure and development of language in general or of particular languages

This rather broad definition, taken from the Cambridge online dictionaries [63], allows us to establish a straight forward relationship between linguistics, NLP and this project in particular. NLP utilises computational techniques, technologies and algorithms to perform tasks of such systematic studies of structure and development of language. These tasks can concern themselves with both spoken or written language and may deal with the most varied issues, ranging from speech recognition to Part-of-Speech (POS) tagging. This project has at its focus an issue of linguistics, as well. The expression of opinions constitutes an integral part of speech, both in written and spoken form. Consequently, insights into the field of linguistics form a vital basis to the task of Sentiment Analysis and may contribute to the workings of solutions in this field in various ways.

The central benefit of linguistics we gather for this project is the knowledge about features that words exhibit. Linguistics provide the framework for a structured analysis of features such as the type of a word, its case, its tense etc. All these methodologically defined facts about constituents of speech allow us to build collections of features along which we can compare and analyse parts of speech in a systematic manner. Without the frameworks of linguistics that define language according to a set of rules, such analyses would proof difficult to realise.

The utility of linguistics with regards to Sentiment Analysis goes beyond this generic benefit which is applicable to any NLP task. Numerous insights provided by the study of linguistics can be translated into heuristics along which more efficient and more performant Sentiment Analysis systems can be designed. In [15], Ding and colleagues propose four linguistic rules that may be used to enhance the performance of a Sentiment Analysis system in one way or another:

1. *Intra-sentence conjunction rule*: A sentence expresses just one opinion orientation unless it contains a *but* which changes the opinion orientation. Thus, if two sentences are conjoined by an *and*, we assume the same opinion orientation for both sentences

2. *Pseudo intra-sentence conjunction rule*: Similar to rule one, except we try to find conjunctions that are not expressed by an *and*

3. *Inter-sentence conjunction rule*: The same principle is applied as for rule one and two, only that it is extended to analyse consecutive sentences.

4. *Synonym and antonym rule*: If a word is found to be positive in a context, its synonyms are considered to be positive, as well. Its antonyms are considered to be negative, instead.

These rules do not give reasons on classifying certain groups of words according to features they share. They are rather a generalisation of information about words or phrases that has been obtained by some measure to other words or phrases that are in proximity. As is the case with any heuristic, these rules do not hold in every scenario. Nevertheless, one valuable way of putting heuristics to use lies in pruning the space of possible feature values that have to be considered in analysing constituents of text. More such heuristics can be devised from linguistic insights and further limit the effort needed to classify text according to whatever criteria are applied. Some other heuristics are described in section 2.3.2. One of them builds upon evidence that collocations, i.e. word combinations that appear more frequently than would be expected by chance, exhibit a higher probability of being opinionated than collocations that appear less frequently. We see that heuristics can be derived from various characteristics that text may exhibit. Syntactical features such as the type, the tense or the case of words may also prove to carry some meaningful relation to a word's or phrase's opinionatedness that could be exploited in the form of a heuristic.

Ding and colleagues define an *opinion aggregation function* and construct an algorithm that categorises the constituents of customer reviews applying both the rules explained above and the *opinion aggregation function*. This function is applied to identify all product features $F = (f_1, ..., f_n)$ and the opinion $O = (o_1, ..., o_n)$ expressed on them in a sentence s. The algorithm thus returns a score for each feature in a sentence, i.e. on the pair (f_i, s). Sentence s is first segmented using *but* words/phrases, i.e. *but, except that*, etc. For each segment s_k of sentence s, a score is calculated,

$$score(f_i, s_k) = \sum_{w_j \in s_k} \frac{w_j.SO}{d(w_j, f_i)} \tag{2.1}$$

"where w_j is an opinion word in s_k which is the sentence segment that contains the feature f_i and the opinion word w_j in s_k. $w_j.SO$ is the semantic orientation of word w_j." [15] The value assigned to $w_j.SO$ for a positive word is 1 and -1 for a negative word. The semantic orientation of a word is established using a lexicon that has been developed beforehand. The distance measure lessens the contribution of words to the score the farther they are away from f_i. If the final score (f_i, s) is positive, the opinion on the feature is as well, if the score is negative, then so is the opinion

on the feature. Those opinions that could be determined, i.e. those for which a semantic orientation value is available, are then used to iteratively establish the orientation of more opinions, applying the rules proposed. Starting out from the initial set of classified opinions, these are expanded to include feature judgements. These are added to the list of known opinions and the rules are applied again.

In this application, heuristics based on linguistic considerations are applied to widen the scope of the knowledge available. They are used to generalise the information gathered on a word's or phrase's opinionatedness to other text constituents surrounding the classified term. Along with the above mentioned pruning of the search space, this is a second potential application of heuristics.

2.2.2 Natural Language Processing

In the introduction (1) we defined Sentiment Analysis as a NLP problem. Though not sufficient on its own, this captures some of the main aspects of the field, some of which, relevant to the topic, deserve mentioning to allow a more precise understanding of Sentiment Analysis. For more details on NLP, refer to [31] or [43], from which the following points are adapted.

- Generally, Natural Language Processing can be applied to text in its raw format or in a marked up format. This means that a text corpus may either be simply in human readable format or it may be annotated with meta information about the text itself, such as the case or gender of a word. In Sentiment Analysis, such mark up may, for example, explicitly identify words as opinionated or factual. The question of whether one is dealing with sheer text or marked up text is essential when deciding on how to analyse it. As elaborated in chapter 3, techniques that are built upon corpora of annotated text generally apply Supervised Learning algorithms, while unannotated data is, in most cases, processed using Unsupervised Learning algorithms.

- NLP involves a number of low level formatting issues that need to be considered. The text that is being analysed may contain formats the machine or system used cannot deal with, such as document headers and separators, tables and diagrams, and various others. Such content may need to be filtered out or be preprocessed before carrying out the actual analysis task. Similarly, punctuation marks and other special characters may need preprocessing to be dealt with adequately. For example, one issue may be whether or not it is essential to *know* where a sentence ends. This would entail replacing punctuation marks such as full stops and question marks by an identifier specifying the end of a sentence.

- The issue of tokenisation, i.e. the process of dividing the text corpus into units which can later be analysed, needs to be considered during the early stages of any NLP task. Units usually determined during this task are words, punctuation, et cetera. For certain analysis tasks, including Sentiment Analysis, it may be an interesting option to build tokens including more than just one word, so called n-grams, which allows consideration of a word's context. The word *house* on its own would be a 1-gram, the result of adding an article to form *the house* is a 2-gram, and so on.

- Another central issue of NLP is the question of how to deal with the morphology of words, for example tense and case. Should the verbs *paint* and *painted* be considered to be the same during an analysis task or does the difference of tense carry information that needs to be taken into account? Usually, the process of collapsing grammatically different instances of the same word to its basic form is referred to as stemming.

There are far more issues that arise during NLP tasks, of which many are important to Sentiment Analysis, as well. The four chosen here are merely some of the most prominent and regularly encountered problems one is faced with when dealing with natural language computationally. Also, it should have become clear that, before actually digging into the issues that are characteristic to Sentiment Analysis, and which are covered subsequently, one needs to consider the fact that there

is a rather large gap in need to be bridged between a human reading text and a machine reading text.

NLP has produced numerous practical applications that pervade our daily interaction with text. One of the most prominent technologies has been POS tagging which is a constituent of many NLP systems. Numerous techniques of probabilistic POS tagging have been proposed, one of which is also part of the system I have developed during this project. POS tagging algorithms have been offering high accuracy in annotating text with syntactical features for about 20 years (e.g. [4], [64], [70]). I have used the TreeTagger [70] system to annotate both the text corpus developed and user input during system execution (see chapters 5 and 6). Among the models that have been applied to POS tagging we can find maximum entropy models [64], decision tree solutions [70] and Markov models [4].

2.3 Central challenges

2.3.1 Gathering text

In order to allow systems to learn, many of them need to be supplied with some sort of training input. The underlying learning processes that have been implemented in Sentiment Analysis are explained in detail in chapter 3, but to make any such process possible in the first place, for many techniques a knowledge base (KB) of examples has to be at hand (see [46] for an introduction). A decision has to be made on what kind of knowledge base (KB) should be used and how it is to be acquired. This is a non trivial issue not just because it affects the entire process following, but because the gathering of text itself entails a number of challenges and as such, building an extensive text corpus has been a substantial part of this project. The text corpus developed within this project is described in detail in chapter 5.

The construction of a text corpus can generally be an automated process or a manual one, with hybrid approaches with differing levels of emphasis on automation and manual annotation feasible, as well. Depending on the task, the size of the corpus that is built and the resources at hand, either of the two methods have their merits and demerits. Constructing corpora automatically is usually a less expensive method than constructing corpora manually. The main workload for automatic construction lies in finding an effective, and optimally efficient, algorithm that produces reliable results that have to be only marginally proof read by a human. The main efforts in constructing a corpus manually are divided into developing a practical annotation scheme that conveys all the information desired and the workload carried by the human annotators. Despite the large effort in manual annotation, such corpora are, up to this date, often preferred because they usually have an advantage over automatically constructed corpora in regards to quality and reliability of the annotations. Due to this trade off between work load and annotation quality, hybrid approaches such as the one proposed in this project can be an adequate choice when constructing a corpus.

Using manually annotated text corpora has been among the more popular approaches to the problem during past developments in Sentiment Analysis. Along with numerous annotated general purpose corpora, a number of corpora tailored to the needs and demands of Sentiment Analysis are either publicly available or available for a fee, the most widely used being the Multi-Perspective Question Answering (MPQA) corpus described in [82], which has been incorporated into my own corpus, and the TREC (Text REtrieval Conference) blog tracks ([42], [55]). While the MPQA corpus annotates new articles and is described in detail in chapter 5, the TREC blog tracks utilise blog entries which is motivated by recognising blogs as "created by their authors as a mechanism for self-expression" [56]. This interpretation of blogs implies that one should find a large amount of opinionated contents when analysing blog entries which would make blogs rather interesting to Sentiment Analysis. The original TREC blog track (2006) was comprised of 100,649 unique blog entries and has been extended subsequently. The TREC blog track entries are manually annotated by first identifying *targets* of opinions and then evaluating whether a blog entry contains opinions on one of the relevant targets. Thus, each blog entry is annotated in two steps, with the annotators only concerning themselves with the nature of the whole entry, not single words or phrases contained in it. In the

first step the annotator identifies a blog entry as either *not relevant* or *relevant*, where *relevant* entries contain an opinion about the target of concern and entries that are *not relevant* contain information about the target, but no opinion. Those blog entries that are identified as *relevant* are further annotated as either *negative opinionated*, *mixed opinionated* or *positive opinionated*.

Using corpora such as the MPQA corpus or the TREC blog tracks has the obvious upshot that little effort has to be poured into the entire subject of amassing a knowledge base because it is available in a ready-to-use format. This leaves the developer with the task of implementing a learning algorithm using this corpus according to the conventions and syntax of the corpus itself.

Using a manually annotated corpus gives rise to the potential issue that the resulting system may not be easily transferred into a real life setting. This is due to the fact that in most cases manual annotation relies on the judgements of a few participants annotating large sets of text. We can argue that this process may be prone to exhibit bias towards the annotators' own, in the case of opinionatedness, partially subjective understanding of how guidelines for annotation are to be interpreted and put to work. In addition to such problems potentially spoiling the quality of manual annotations, the process itself is rather laborious and little of the content encountered within the World Wide Web is annotated as needed for Sentiment Analysis. I have attempted to reconcile both these issues by passing the responsibility of annotation directly to the user of a system. This means that annotation is accomplished without burdening single individuals with large annotation tasks and also, assuming a large number of different users supply annotation, without relying on just a few sets of understanding of how opinionatedness manifests itself in text. More detail of these potential benefits is provided in sections 6.2 and 8.2.3.

A step away from using one monolithic corpus of fully annotated text is taken with so called Unsupervised Learning techniques. One of the Unsupervised Learning techniques applied in Sentiment Analysis takes a number of so called seed words as a starting point. A learning set is built iteratively using seed word relations to other words or concepts determined by using, for example, WordNet [45]. In [18] and [19], Esuli and Sebastiani describe building a training set from a small number of seed words by iteratively exploiting lexical relations of words such as synonymy and antonymy. In [2], Baroni and Vegnaduzzo use only a small set of manually selected seed nouns and adjectives and use the Web-based Mutual Information (WMI) method developed by [75] to build a large training set. While Unsupervised Learning techniques do not share the issues that arise when using manually annotated text, other issues arise. The predominant problem that needs consideration when annotating text in such a way is the development of a technique that is able to produce quality annotations without, or with little, human intervention. Some of the Unsupervised Learning algorithms used in Sentiment Analysis are covered in more detail in chapter 3.

2.3.2 Extracting opinionated content from text

Assuming a sufficiently documented and structured knowledge base is available a priori or one has been constructed, we are in the position to tackle the first of two major challenges Sentiment Analysis boils down to. Even if it is known that the analysed text is opinionated, generally the text will still include some factual information. Unless the analysis is performed at document level, in which case this task loses its relevance when the general nature of the text is known, the factual and opinionated information have to be disentangled to allow a closer analysis of the opinionated parts of the text. Basically, this results in a binary classification task that places parts of the text into the category *opinionated* and others amongst the category *non opinionated*. Alternative definitions may call these categories *subjective* and *objective*, but in certain cases this can be misleading because even an objective statement may implicitly carry an opinion. Take the following sentence:

> After I'd left, the agenda lost momentum. But the papers and the work are all there.

This is a quote of Tony Blair commenting on how his efforts on reforming the criminal justice system during his time of administration were handled after he had left office. Though these two

sentence simply state that these reform efforts died down after the change in government, most humans will read a clear negative connotation in these sentences, expressing a frustration with a new government abandoning policy that had been pushed forward before. Despite these sentences arguably being opinionated, one may be hard pressed to find clues for subjectivity in these two sentences. Such cases appear more or less frequently in text and for this reason I deemed a definition of opinionatedness more robust to such occurrences than one of subjectivity.

Like this project, a number of works have focussed on the issue of discerning opinionated text from non opinionated text, such as [2], [5], [28], [35], [67], and [87]. In [81], Wiebe and colleagues propose a number of clues to subjectivity, or opinionatedness, and test them one-by-one, as well as in combination. Among those clues, i.e. heuristics, are

- Low-frequency words
- Collocations
- Certain adjectives and verbs identified using distributional similarity.

According to the definition used, low-frequency words are words that only appear once in a piece of text. Wiebe and colleagues were able to show that such words exhibit a higher probability for being opinionated, i.e. words that appear just once in a text indicate that the excerpts containing them exhibit a heightened probability of being opinionated. Collocations define combinations of words that appear in a text more often than it would be expected by chance. Analogously to low-frequency words, collocations were shown to indicate a higher probability of the excerpt containing it to be opinionated than those not containing any collocations. A third indicator for opinionatedness identified by Wiebe and colleagues estimates potential subjective elements (PSE) using *distributional similarity* which judges words "to be more or less similar based on their distributional patterning in text." [81]

The use of these clues is founded on linguistic considerations which have been elucidated in more depth in 2.2. The issue of disseminating opinionated content from non opinionated content is dealt with in more detail in chapter 6 where the system developed in this project is presented in its final form and functionality.

2.3.3 Determining polarity

Once opinionated text is identified or when a text is assumed to be fully opinionated to begin with, the second primary challenge of Sentiment Analysis is the identification of an opinion's polarity. Polarity describes the nature of the sentiment, e.g., on a coarse level, whether it is negative or positive. More fine grained differentiations are easily imaginable. Pang and Lee use a five star rating scale for sentiment polarity in [58], i.e. they introduce three additional levels of polarity distinctions. A number of works on this opinion classification task deals only with the binary distinction between a positive opinion and a negative opinion ([14], [61], [76], [85]), but as Pang and Lee point out, this may not cover the actual spectrum of opinions, even in a single text, sufficiently. The two main issues arising when simple binary distinctions are applied are, for one, the inability to include any notion of strength of opinion within one orientation, and secondly, the incapacity of covering potential middle ground when an opinion is presented, but a clear leaning to the positive or negative cannot be determined. Both these issues are investigated more closely in section 2.4.1.

To illustrate just one of the many possibly arising issues when determining text fragments' polarity, consider the following two sentence from the same editorial comment that the previous example was drawn from:

> [...] many of these people are from families that are profoundly dysfunctional, operating on completely different terms from the rest of society, either middle class or poor. This is a phenomenon of the late 20th century.

The first sentence allows us to decide that the phenomenon talked about in the following sentence is clearly a negative one. Without setting the sentence *"This is a phenomenon of the late 20th century"* in this context, though, we would not be able to decide whether *phenomenon* carries a positive or a negative connotation. The issue that arises from stripping such a sentence bear of its surroundings holds not just for a machine processing this sentence, but for a human reader, as well. The fact that even a human reader may run into trouble when classifying parts of speech (as can also be seen in the experimental results in [36]) makes it rather easy to fathom that this is an intricate task to be implemented on a machine.

To illustrate one of the many potential problems, consider a technique that is described in detail in section 3.2. [8] lets a system build its own lexicon of opinionated words starting from a small number of seed words. The synonyms of these seed words are then extracted from WordNet [45] and added to the lexicon, classified as having the same polarity as the seed word itself. One of the seed words may be *fortune* which most people would classify as unambiguously positive. According to WordNet, *phenomenon* is is related *fortune* via hyponomy, i.e. via a *type-of* relation. It may thus be incorporated in the lexicon as a positive word, depending on the inclusion criteria. This would clearly render correctly classifying the sentence above using this lexicon at least problematic. Section 8.2.1 investigates how the problem of determining the polarity of an opinion may be approached, both with regards to the system presented and in a more general manner.

2.3.4 Summarising

Irrespective of the fact that summarising content is not necessarily an issue characteristic just to Sentiment Analysis, as it is part of manifold disciplines and issues unrelated to the field, it is important enough in this context to be considered a major aspect of the field. Consider again, for example, the MPQA corpus from [82]. This corpus is comprised of 378 news articles, a total of 10,657 sentences. This is, of course, the training set of a system, but even if the amount of text analysed is considerably smaller (and it may just as well be bigger), the human reader may very likely have a hard time digesting the analysed information. These troubles would not just arise from the sheer size of analysed text but also from the, not necessarily human friendly, machine read and produced format. Consider a short excerpt of MPQA corpus annotation syntax:

766 1722,1756 string GATE_ on nested-source="w" is-implicit="" onlyfactive="yes"

The corpus annotations are made up entirely of such lines, each line describing some characteristic of an n-gram appearing in one of the annotated news articles. The meaning of this annotation is explained in detail in chapter 5 and is not of much concern at this point. Merely the issue shall be pointed out here that a human reader may be able to digest some of the information contained, such as that we are dealing with information that is, by some definition, factual. Beyond some basic comprehensibility, the value of information of the raw data is arguably rather low to the human eye. This may mean that, without proper summarisation, the efforts of Sentiment Analysis and its results may, in many cases, ultimately be rendered useless, simply by the lack of finding ways of representing content in an intuitively understandable manner.

There have been two different approaches as to how opinionated text may be summarised and represented. On the one hand, it has been suggested that an opinionated text may be represented as a shorter, also textual, version of itself. By distilling out the parts of a text that convey the central aspects of it with regards to its opinionated content and its orientation, and discarding the rest, a more comprehensive and compact representation of the text's contents may be attained. Such textual summaries may take a number of shapes. One could aim at extracting keywords or key phrases from the original text that provide an adequate summery of the text's opinionated contents. Another way might be to find a single excerpt which, according to classification results, holds the most prominent or most characteristic opinion of the text as a whole.

The second, in Sentiment Analysis prevalent, approach that has been taken is to present the results

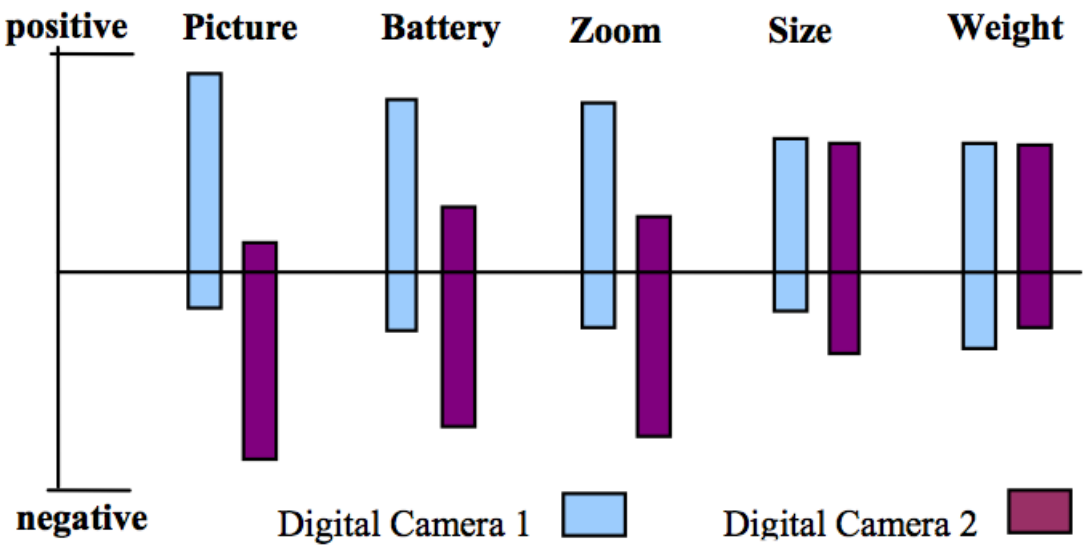

Figure 2.1: Example of graphical sentiment summarisation comparing two cameras (from [41])

of an analysis graphically, as described in [40] and [41]. Both techniques offer different qualities, their value depending very much on the domain and topic that is to be summarised. An example in [41] compares the sentiments towards two different cameras and their individual qualities taken from customer reviews. This kind of setting naturally lends itself to a concise graphical presentations of the sentiments because it allows the potential user of such information to effortlessly access the desired content. Figure 2.1 shows the graphical representation suggested in [41]. In other domains it may be more desirable to maintain a more complex, textual structure in order to retain important aspects of the analysed text. Trying to summarise an extensive commentary on current international affairs in a solely graphical manner may prove to be rather complicated or inappropriate for the issue. As is often the case, choosing a hybrid approach utilising both textual and graphical summarisation of analysis results may add to the value of a system's output by unifying individual qualities of the two different approaches to representation.

2.4 Peripheral challenges

The challenges described throughout the former sections may, as the title suggests, be viewed as the central challenges of Sentiment Analysis, but they are by no means all encompassing when it comes to the problems one faces when developing a system. There are numerous other problems that may or may not arise and those most important by today's standards are investigated below. Going further in the development of the approach and system that have been proposed by this project, and are described in detail in the subsequent chapters, would almost certainly mean tackling most of the challenges touched upon in this section. Suggestions as to how one might accomplish digging deeper into these not so imminent, but nevertheless vital, issues will be presented alongside potential gains from doing so in chapter 8. This is why, in spite of the fact that the issues presented below were generally only briefly touched upon during the project itself, elucidating upon these problems is vital to gain a full and clear understanding of the complex field of Sentiment Analysis and what is entailed by developing solutions within it.

2.4.1 Determining strength and other degrees of opinion

As discussed earlier in section 2.3.3, determining an opinionated text fragment's polarity raises the question of whether or not determining the strength of an opinion is a viable and valuable task. By simply saying that a statement is a positive one, no information is conveyed about whether the entity stating this opinion feels strongly about this particular statement or if it doesn't. Also, a

positive and a negative opinion may very well have the same strength, just situated on opposing ends of a scale. Thus, determining the strength of an opinion, in addition to its basic orientation, may offer useful information contributing to achieving a more detailed understanding of the text that is being analysed.

In [83], Wilson collects a number of different clues that suggest a high intensity of a particular text passage and states that these high levels of intensities can, in turn, be used to find opinionated content more effectively. Such clues include words and word combinations such as the following, and many more:

- grown tremendously
- so exciting
- very definitely
- gross misstatement

Accordingly, this example shows that not only may a more fine grained categorisation of the opinions encountered allow a more detailed understanding of the analysed text once it is classified accordingly, but at the same time contribute to a higher quality of the classification process, as well. Wilson also shows that inter-annotator agreement is significantly higher when comparing the annotations of such high intensity words or phrases with words or phrases of lower intensity. This suggests that analysing text constituents with respect to their intensity may allow us to employ some judgement upon ambiguities we are potentially dealing with. We may invest more trust in classification results if the text involves words or phrases of high intensity if we are confident that they are indeed less ambiguous than less intense words and phrases.

As mentioned in section 2.3.3, in [58] Pang and colleagues introduce a rating scale for a word's or phrase's sentiment that extends the basic notion of a binary classification as positive or negative. Instead of extending this binary classification scheme to encompass, for example, *weakly negative*, *stronlgy negative* etc., Pang and colleagues choose a rating scale with which one to five *stars* are assigned to a word, phrase or document to judge it with regards to its positivity or negativity. Consequently, this task is strictly speaking only marginally concerned with determining the strength of an opinion, but rather aims at supplying finer distinctions between the degree of positivity. By assigning a phrase one *star*, we supply a rather negative opinion, but give no information about whether we feel strongly about this negative judgement.

Pang and colleagues propose three different classifiers, all based on Support Vector Machines (SVM) (see section 3.1.1 for details). The first approach uses *one versus all* (OVA) SVM, a generalisation of the original SVM specification, which handles binary problems, to multi class problems such as the one at hand. "Training consists of building, for each label l, an SVM binary classifier distinguishing label l from 'not-l'." [58] The second classifier uses regression, based on the assumption that "the labels come from a discretisation of a continuous function g mapping from the feature space to a metric space." [58] Viewing the problem in metric space is justified mainly with the assumption that we can deduce some metric distance between the rating, e.g. that a rating of three *stars* is, by some distance measure, *closer* to a rating of four *stars* than it is to one *star*.

A third algorithm used to solve this problem is *metric labelling*, which, as the name suggests, also utilises the assumption of some distance measure separating the different *star* ratings. To perform metric labelling, a "special case of the *maximum a posteriori* estimation problem for Markov random fields" [58], we need to have an initial function determining label probabilities, $\pi(x, l)$. These label preferences may be determined using one of the previously described methods. With a measure d denoting a distance of labels and $nn_k(x)$ signifying the k nearest neighbours of an item x according to some similarity measure sim, we can find a mapping of instance x to label l_x that minimises

$$\sum_{x \in test} \left[-\pi(x, l_x) + \alpha \sum_{y \in nn_k(x)} f(d(l_x, l_y)) sim(x, y) \right] \tag{2.2}$$

where f is monotonically increasing and α is a trade-off and/or scaling factor. Through this, classifications are penalised that assign divergent labels to similar classes.

For the evaluation of these three methods, the data set used by Pang and colleagues was collapsed into two different versions, one of three classes, *negative*, *middling* and *positive*, and one of four classes. The three class problem was chosen to enable performance comparison of the classifiers on problems of differing complexity. A four class separation instead of the original five classes was chosen due to the fact that using five classes yielded too few training instances for some of the classes. "In the four-class case, regression performed better than OVA [...], but for the three-category task, OVA significantly outperforms regression [...]." [58] In both the three-class and four-class cases, metric labelling increased performance.

2.4.2 Determining opinion holder and target

In an application, the user may be interested in opinions on certain targets or objects, but not on all. This issue gives rise to the necessity of considering not only whether or not a piece of opinionated language is positive or negative and what its content is. Other parameters may carry significance, as well. It may be crucial to determine who the source of the concerning opinion is, or who the opinion is directed towards. Ruppenhofer and colleagues focus on this issue and highlight its importance in [69]. They use so called Automatic Semantic Role Labelling (ASRL), developed by [23], as a base for their developments. ASRL aims at identifying semantic relationships filled by constituents of a sentence within a semantic frame, which is roughly what is intended when trying to identify a holder or target of an opinion. ASRL identifies constituents of phrases according to the roles they take based on the lexicon developed during the Berkeley FrameNet project [1]. The part of the lexicon that is applied to ASRL is made up of frames, each of which which describes a family of lexical items. A frame then contains those lexical items that are member of the family and relevant to identify constituents of a sentence that take up a role. For example, the frame *TRANSPORTATION* contains the frame elements *MOVER(S), MEANS, PATH* and the scene *MOVER(S) move along PATH by MEANS*. Frames can also inherit other frames, e.g. the frame *DRIVING* inherits the frame *TRANSPORTATION*. Using this construction of frames, their elements and inheritances, ASRL identifies roles within phrases by assigning phrase constituents to frames and the according roles they take within a frame.

Ruppenhofer and colleagues identify the following four issues that are not sufficiently covered by ASRL and explain their implications:

- Attribution, i.e. the relation between beliefs and assertions expressed in text and their sources
- Referent identification, which becomes an issue when the entity or entities referred to are not overtly expressed in the text
- Inferences concerning attitudes and their sources and targets
- Targets of a less studied subjectivity type: arguing

These four issues, in combination with the underlying principles of ARSL, roughly outline what needs to be considered when attempting to identify an opinion holder or the target of an opinion. The general goal is to break down sentences or other parts of text according to the relationships that the constituents the text is made up of have to each other. By identifying such relations, one can deduce which parts of a piece of, in this scenario opinionated, text represent the entity expressing an opinion, which parts represent the receiver of an opinion and which the opinion itself.

2.4.3 Scope of context

An opinion is, in many cases, very much dependent on the context it is uttered in. Not only can an utterance's surroundings supply information about its content, holder or target, but often these surroundings determine whether or not a text fragment can be classified as opinionated, in the first place. Consider the following two sentence:

> Due to the most recent developments, the oil price has steadily risen. This is clearly very bad news for Britain's drivers.

A human annotator would not be troubled by the task of properly identifying the rising oil prices as something the author of the sentences appraises negatively. Without the second sentence, though, identifying this expression as negative, or even opinionated, would arguably not be valid. Only by considering the context of the statement, namely the sentence following it, can the classification be carried out reliably.

The issue of an expression that is, on its own, seemingly void of any opinion, but when placed in a certain context changes its property, frequently arises and has thus received recognition in efforts to develop Sentiment Analysis systems. [5] proposes a number of features that, in unison, are to identify opinions among factual information. One group of features, which the authors call lexical features, aim specifically at taking into account the context when analysing a word. Each word is successively considered in combination with the word occurring before and after it, and in the same manner with two words, three words, and four words prior to and after it. Breck and colleagues use two additional groups of features, one of which is based on POS tagging. All constituents of the analysed text are tagged with an appropriate type identifier. For each word, POS tag features are encoded by three values, *prev*, *cur* and *next*. These features hold the POS tag of the token preceding the constituent of concern, the tag of the constituent itself and the tag of the token following it. By encoding the POS tag features in such a manner, the context in which the tokens appear are further taken into consideration.

Applying these two feature representations to the example above, though, it is arguable whether limiting contextual considerations to the eight, or two, words closest to the term of interest always suffices. At the same time it is rather obvious that, as the scope of contextual analysis is widened further, analysing every possible combination of n-grams may become rather cumbersome. Take, for example, a text of 200 words that is to be analysed. If only 1-grams, i.e. single words, are considered, $f_1(200) = 200$ 1-grams need to be analysed. If each word is analysed by itself and also, in turn, together with the word preceding it and the word following it, the number of fragments, f_2, in need of analysis rises to $f_2(200) = 200 + (2 * 1 + 198 * 2) = 598$. Analysing $f_3(200)$ would mean analysing $f_3 = 200 + (2 * 1 + 198 * 2) + (4 * 1 + 196 * 2) = 994$ combinations. Generally, following this pattern of analysis, whenever considering more than just 1-grams, we need to analyse

$$f_n(N) = N + \sum_{i=2}^{n} 2N - 2(i-1) \tag{2.3}$$

word combinations will have to be analysed, where N is the total number of analysed words and n is the size of the largest n-gram that is considered. Issues may arise because, generally, significantly larger text corpora may be analysed and the analysis of each n-gram is a process potentially involving a number of subtasks in itself.

In [57], Pang and Lee propose *cut-based* classification, a technique that classifies words according to two scores, *individual* scores and *association* scores. "Suppose we have n items $x_1, ..., x_n$ to divide into two classes C_1 and C_2" ([57]). Individual scores, $ind_j(x_i)$, are non-negative estimates of each x_i's probability of belonging to one class or the other just based on x_i's value. Association scores, $assoc(x_i, x_k)$ estimate the importance of the two classes having the same class label. It is then the aim of a classifier to both maximise an item's individual score while penalising putting closely related items in different classes. From these two objectives a partition cost is defined that is to be minimised. By introducing the association score, Pang and Lee establish a relation between words appearing in a text that is not based on local proximity within the text, but rather on proximity of meaning and thus propose an alternative to the approach of partitioning text into n-grams.

3

Machine Learning in Sentiment Analysis

Having outlined the predominant challenges that arise when developing Sentiment Analysis systems in chapter 2, this chapter offers an overview of the Machine Learning techniques that have been deployed in this area. Generally speaking, the Machine Learning techniques that have been applied within the field of Sentiment Analysis may be placed among one of three categories:

1. Supervised Learning (3.1)
2. Unsupervised Learning (3.2)
3. Reinforcement Learning (3.3)

Out of these three categories, Supervised Learning algorithms have been the most popular in Sentiment Analysis and Reinforcement Learning techniques have only been considered in recent years. Argumentation, which constitutes a significant part of this project, has not been used in any Sentiment Analysis developments as of today and is not covered in this chapter, but rather separately in chapter 4. Refer to [40] and [59] for additional examples and explanations of Machine learning algorithms that have been applied to Sentiment Analysis but are not mentioned in the following sections.

3.1 Supervised Learning

Some of the most predominant Supervised Learning techniques in Sentiment Analysis have been SVM, Naive Bayesian Classifiers, and other, mostly binary, classifiers. The rise of interest in Supervised Learning is largely owed to an increase in availability of labelled text corpora such as the MPQA corpus ([80]) or the TREC blog tracks ([55], [42]), which are a basic prerequisite for using Supervised Learning techniques. Before giving a detailed account of SVM, the algorithm of choice for this project, and a less extensive overview of some other Supervised Learning algorithms, consider the binary classifier described below that is do decide whether a text fragment is opinionated or non opinionated. Considering the information provided throughout the previous chapters, it will become obvious rather quickly that a classifier such as this, even in a case where it performs very well, will only be able to solve the issue partially. Nonetheless, the issues that arise in designing this classifier apply equally, or at least similarly, to other classification problems and this example thus offers a reasonable starting point.

In [35], Kim and colleagues describe a simple method to distinguish opinion bearing sentences from non-opinion bearing sentences using a Supervised Learning approach. As elaborated, having access to labelled data is crucial in this setting, and Kim and colleagues test their learning algorithm on four different text corpora:

1. WordNet ([45])
2. Editorial entries in Wall Street Journal (WSJ) ([35], [42])

3. Columbia WordList ([85])

4. A combination of the above

Kim and colleagues propose two simple training algorithms. One classifies every sentence that, according to the used data set, contains a number of opinion bearing words as an opinion bearing sentence. The second approach is to search for what they call strong opinions and classify a sentence as opinion bearing whenever it contains one of these strong opinion words. To determine whether an opinionated word is also *strongly* opinionated they define a cutoff threshold λ which is determined by means of comparison with human annotated text in regard to opinion strength.

A number of other Supervised learning algorithms have been proposed to tackle different issues of Sentiment Analysis, some of which Pang and colleagues investigate in [61]. They use three different classifiers for the task of classifying movie reviews at document level and compare the performance of the following Machine Learning techniques:

- Naive Bayes classifier
- Maximum Entropy classifier
- Support Vector Machines (SVM)

The results presented by Pang and colleagues show that, for the particular task of classifying movie reviews at document level, the best performance is achieved using SVM while the worst performing method is the Naive Bayes classifier. At the same time, none of the three methods achieves classification accuracy comparable to performance in classifying the reviews topics instead of their sentiment.

Starting with SVM, all three algorithms are explained below.

3.1.1 Support Vector Machines

SVM, pioneered by Corinna Cortes and Vladimir Vapnik ([12]), have been one of the more popular Supervised Learning algorithms not just in Sentiment Analysis, but various other fields, as well, such as spam filtering (e.g. [16], [68]) or medical research (e.g. [6], [22], [27]). The system developed in this project also integrates SVM and, accordingly, its performance is measured up against an SVM classifier. SVM were originally developed as binary classifiers, which is the function they fulfil both in the project and in most other applications in which they are used, but there have also been efforts to extend the use of SVM beyond binary classification problems. The following account of SVM and their works, which will limit itself to explaining binary classification with SVM, has been adapted from [3], which may also be referred to for details on classification with SVM beyond binary decision problems.

Consider a linear model describing a two class classification problem such as the one of determining opinionatedness. We can describe such a problem with linear model of the form

$$y(x) = \mathbf{w}^T \phi(x) + b \tag{3.1}$$

where $\phi(x)$ is a feature-space transformation of the data x where $\mathbf{w}^T$ is a weight vector and b is a bias. Transforming data into feature space can yield linear separability of data that is not linear separable in the original data space. Figure 3.1 shows a simple example of how adding features may achieve separability of data that is not linear separable in data space. A training data set consists of N input vectors $x_1, ..., x_N$ all of which have a class label $C \in \{1, -1\}$ and new data is classified according to the sign of $y(x)$. If the training data is linearly separable in feature space, we can perfectly split the data according to their class with one hyperplane in feature space. When this is not the case, so called slack variables may be introduced which allow some of the training instances to be misclassified. SVM find the hyperplane that not only separates the data but that maximises

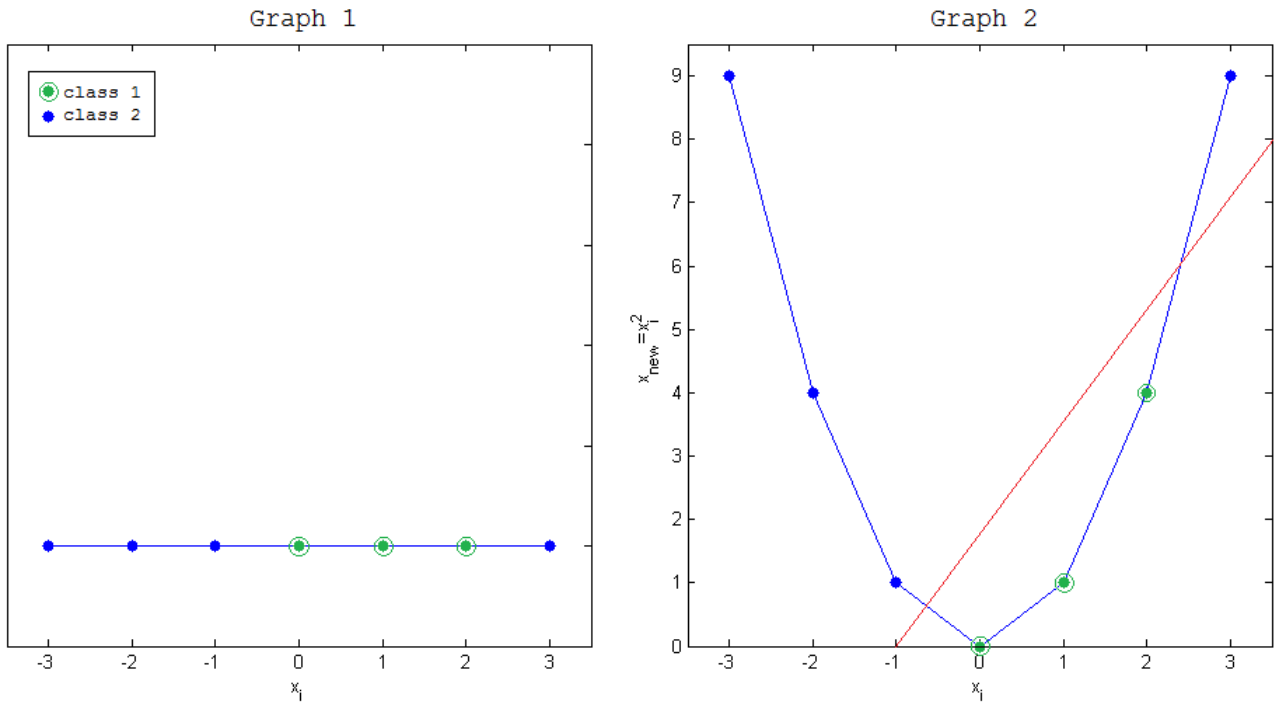

Figure 3.1: Example showing how linear separability of data can be achieved by adding features. Graph 1 shows a two class data set with one feature that is not linearly separable. Graph 2 shows the same data set with a second feature added, making the data linearly separable, as shown by the red line.

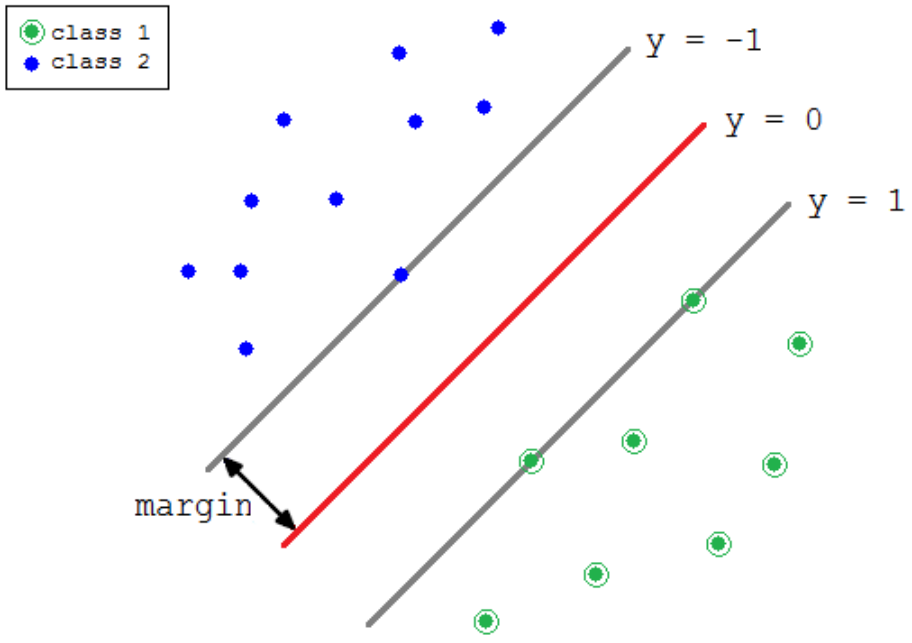

Figure 3.2: "The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points" [3]. The support vectors are those data points that lie on the decision boundaries.

the margin, as well. This means, the separating hyperplane is placed in such a manner that the distance between the separating plane and the nearest points of both classes is maximised. This is illustrated in figure 3.2. The margin, i.e. a distance from point x_n to the hyperplane is given by

$$\frac{t_n y(x_n)}{||\mathbf{w}||} = \frac{t_n(\mathbf{w}^T\phi(x_n) + b)}{||\mathbf{w}||}. \tag{3.2}$$

We wish to maximise the margin, i.e. the perpendicular distance, between the separating plane and the point(s) x_n closest to by optimising the parameters $\mathbf{w}$ and b. This maximum margin solution is found by solving

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{||\mathbf{w}||}\min_n[t_n(\mathbf{w}^T\phi(x_n) + b)]\right\} \tag{3.3}$$

where $1/||\mathbf{w}||$ is taken outside the optimisation over n because $\mathbf{w}$ does not depend on n. Attaining a direct solution of this problem would be very complex and thus some conversions of the problem are needed. We can set

$$t_n(\mathbf{w}^T\phi(x_n) + b) = 1 \tag{3.4}$$

for the point that is closest to the separating plane and with this, all data points will satisfy the constraints

$$t_n(\mathbf{w}^T\phi(x_n) + b) \geq 1, \qquad n = 1, ..., N. \tag{3.5}$$

This is the so called canonical representation of the separating plane which leaves us to maximise $||\mathbf{w}||^{-1}$ which is equivalent to minimising $||\mathbf{w}||^2$. Thus, we optimise

$$\arg\min_{\mathbf{w},b}\frac{1}{2}||\mathbf{w}||^2 \tag{3.6}$$

subject to the constraints given by equation (3.5). We introduce Lagrange multipliers (see appendix B.1) $a_n \geq 0$ with one multiplier a_n for each constraint to solve this optimisation problem, which yields

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n\left\{t_n(\mathbf{w}^T\phi(x_n) + b) - 1\right\} \tag{3.7}$$

where $\mathbf{a} = (a_1, ..., a_N)^T$. Setting the derivatives of $L(\mathbf{w}, b, \mathbf{a})$ with respect to $\mathbf{w}$ and b equal to zero yields two conditions,

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(x_n) \tag{3.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{3.9}$$

Using these two conditions to eliminate $\mathbf{w}$ and b from $L(\mathbf{w}, b, \mathbf{a})$ yields the *dual representation* of the maximum margin problem we are trying to solve. We are thus maximising

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \tag{3.10}$$

with respect to under the constraints

$$a_n \geq 0, \qquad n = 1, ..., N, \tag{3.11}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{3.12}$$

kernel type	function
linear	$K(x, y) = x^T y$
polynomial	$K(x, y) = (\gamma * x^T + coef)^p$
RBF	$K(x, y) = \exp(-\gamma * \|x - y\|^2)$
sigmoid	$K(x, y) = \tanh(\gamma * x^T * y + coef)$

Table 3.1: Kernel functions provided by libSVM

$k(x, x')$ represents the kernel function that is used in training the SVM. A number of kernels applicable to SVM exist, libSVM ([7]), the SVM implementation used for the project, offers linear kernels, polynomial kernels, radial basis function (RBF) kernels and sigmoid kernels (see table 3.1 for details on these kernels).

Using the trained SVM model as described above, we classify new data points by evaluating the sign of $y(\mathbf{x})$ in equation 3.5. We express equation 3.5 in terms of the paramters $\{a_n\}$ and the kernel function by substituting for $\mathbf{w}$, using 3.8

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{3.13}$$

This constrained optimisation problem satisifes the *Karush-Kuhn-Tucker* (KKT) conditions (see appendix B.2), which ins this case requires that the following properties hold:

$$a_n \geq 0 \tag{3.14}$$
$$t_n y(\mathbf{x}_n) - 1 \geq 0 \tag{3.15}$$
$$a_n\{t_n y(\mathbf{x}_n - 1\} = 0 \tag{3.16}$$

This means that for every data point we have either $a_n = 0$ or $t_n y(\mathbf{x}_n = 1)$. All data points for which $a_n = 0$ will not appear in the sum in equation 3.13. Thus they play no role in making predictions for new data points. The data points that do appear are called *support vectors*, and they satisfy $t_n y(\mathbf{x}_n) = 1$, i.e. they correspond to points that lie on the maximum margin separating plane in feature space (see figure 3.2). This fact poses one of the crucial benefits of SVM because, once the model is trained, we only retain the support vectors for subsequent classification and discard of all other data points.

3.1.2 Other Supervised Learning techniques

The Naive Bayes' Classifier assigns a given document d the class $c^* = argmax_c P(c|d)$ where, according to Bayes' rule

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)} \tag{3.17}$$

in which $P(d)$ is simply a normalising factor. The probability of a document belonging to one of the possible classes, i.e. opinionated or non opinionated, is decided upon on the basis of a set of m possibly occurring features, $\{f_1, f_2, ..., f_m\}$. $n_i(d)$ then denotes the number of times f_i occurs in document d. Assuming conditional independence between all f_i, equation 3.1 may be rewritten as

$$P_{NB}(c|d) := \frac{P(c)(\prod_{i=1}^{m} P(f_i|c)^{n_i(d)})}{P(d)} \tag{3.18}$$

Pang and colleagues' training method consists of relative-frequency estimation of $P(c)$ and $P(f_i|c)$. This technique, despite being rather simple, often produces acceptable results.

Maximum Entropy Classifiers serve the basic idea that we should prefer the most uniform models that at the same time satisfy any given constraints. A model for classification must be found

that satisfies any constraints imposed by the data that is used and at the same time makes as few assumptions about this data as possible to allow a good degree of generalisation onto new data. Probability estimates for assigning data to one of the possible classes, e.g. opinionated and non opinionated, take the following form:

$$P_{ME}(c|d) := \frac{1}{Z(d)} exp\Big(\sum_i \lambda_{i,c} F_{i,c}(d,c)\Big) \tag{3.19}$$

where $\lambda_{i,c}$ are weights, $Z(d)$ is a normalisation function, $F_{i,c}$ is a *feature/class function* for feature f_i and class c_i defined as:

$$F_{i,c}(d,c') := \begin{cases} 1, & if \;\; n_i(d) > 0 \, and \, c' = c \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

The parameters are then set to maximise the entropy of the induced distribution. The explanations of Naive Bayes' Classifiers and Maximum Entropy Classifiers were adopted from [61].

3.2 Unsupervised Learning

All approaches previously described build upon a set of fully annotated data, which is used to train a classifier, with one technique or another. This classifier is then used to classify novel incoming text. The obvious downside is that, despite more and more annotated text being available, often this reliance on annotated text is going to restrict the general applicability of a system.

Unsupervised Learning techniques are applied in Sentiment Analysis to tackle this issue. Here, one of the most popular approaches, which is explained in detail below, uses so called seed words to automatically build a lexicon, or knowledge base, which is subsequently used to classify newly incoming text. Building a lexicon from seed words is, for example, proposed in [2], [8], [17], [24], [79], [85], [87], and also mentioned in [40], [59] and [81]. This technique based on seed words is, by some authors, also referred to as a semi-supervised learning approach. This additional distinction is employed because building a lexicon from seed words relies on the availability of a small initial set of manually annotated data and is thus not strictly unsupervised.

Kim and Hovy [34] present a system for determining sentiment polarity which uses the concept of seed words to construct a classification model. A small amount of seed words is collected which are either unambiguously positive or negative and annotated accordingly. This list is then iteratively expanded using synonymy and antonymy relations in WordNet. Kim and Hovy assume that synonyms of positive words should be positive, as well, and that antonyms of positive words should be negative, with analogous relationships for synonymy and antonymy of negative words. Those words found by this process in an iteration are added to the collection of words with their according label and the process is repeated. Throughout, Kim and Hovy treat nouns, verbs and adjectives separately. Starting from a seed list of 23 positive and 21 negative verbs and 15 positive and 19 negative adjectives, a large collection of classified verbs, 5880 positive and 6233 negative, and adjectives, 2840 positive and 3239 negative, was constructed, with nouns added further along the process. From this collection a number of words had to be removed because they were classified wrongly. Also, some words appeared multiple times as both negative and positive. Due to this fact, Kim and Hovy developed a measure of strength of sentiment polarity, trying to determine how strongly a word is both negative and positive. Assigning each word a score allows dealing with ambiguities resulting from such problems as multiple occurrences of the same word.

Using the constructed list of seed words and utilising the concept of a score that is calculated according to the frequency and class distribution of words, new unseen words can be classified. Given a new word, a collection of synonyms is collected from WordNet and the relation between this new, unclassified, list and the list collected during training is evaluated by

$$\arg\max_c P(c|w) \tilde{=} \arg\max_c P(c|syn_1, syn_2, ..., syn_n) \tag{3.21}$$

where c is a sentiment category (positive or negative), w is the unseen word and syn_n is the set of synonyms of w as determined by WordNet. Two methods were used and compared to compute this probability measure, the first being

$$\begin{aligned} &\arg\max_c P(c|w) = \arg\max_c P(c)P(w|c) \\ &= \arg\max_c P(c)P(syn_1, syn_2, ..., syn_n|c) \\ &= \arg\max_c P(c) \prod_{k=1}^{m} P(f_k|c)^{count(f_k, synset(w))} \end{aligned} \tag{3.22}$$

"where f_k is the k^{th} feature (list word) of sentiment class c which is also a member of the synonym set of w, and $count(f_k, synset(w))$ is the total number of occurrences of f_k in the synonym set of w. $P(c)$ is the number of words in class c divided by the total number of words considered." [34] This method is rather simple to use because it does not require manually annotated data for training and instead utilises the synonymy and antonymy relations of WordNet, which are already in place. The second model to construct a classification starting from the seed words is shown below:

$$\begin{aligned} &\arg\max_c P(c|w) = \arg\max_c P(c)P(w|c) \\ &= \arg\max_c P(c) \frac{\sum_{i=1}^{n} count(syn_i, c)}{count(c)} \end{aligned} \tag{3.23}$$

The probability $P(w|c)$ of word w given a class c the occurrences of synonyms of w in the list of c are counted.

Either of the two methods yield scores of negativity and positivity for single words. Kim and Hovy use these scoring methods for single words to construct a *sentence* sentiment classifier. The structure they are interested in on sentence level is finding sentiment of a *holder* about a *claim*. Based on results of manual analysis of text, the assumption is made that sentiments are found most likely in close vicinity to the *holder*. As a simplification, Kim and Hovy assume the *topic* as given, i.e. identify it through direct matching. The holder is identified using a Named Entity Recognition (NER) system and a region around this *holder* is considered for sentiment extraction. Different regions are defined and tested, e.g. full sentences or the words between *holder* and *topic*. The sentiment is then derived by analysing the single words appearing in the region according to one of the methods described above and joining the results into a single classification, either by counting the signs in the region or by calculating harmonic or geometric mean. The best performance is achieved when a window encompassing all words in a sentence following the *holder* is considered and the signs in this region are counted.

There have been numerous systems proposed that fall into the category of Unsupervised Learning that use other techniques than seed words. Some classifiers use heuristics, others apply bootstrapping, which is similar to the technique described above. In [60], Pang and colleagues base their classifier on the simple heuristic that words that frequently appear in, for example, a customer review, have a tendency to be factual, and words that rarely appear, and especially words that appear just once, exhibit a high likelihood to be opinionated. This heuristic is based on a previous empirical analysis of text. In [66], Riloff and Wiebe present a bootstrapping approach, which employs a technique similar to, but more complex than in [8] that extracts patterns from training text. These patterns may look as follows:

- <subj> passive-verb
- noun prep <np>
- infinitive prep <np>

- et cetera

Initially, two *high-precision classifiers* are applied to unannotated text, one subjectivity classifier and one objectivity classifier. Based on a collection of words and n-grams that have previously been shown to be good subjectivity clues, sentences in the unannotated text are classified as either subjective or objective, given that a classification can be made with confidence. For subjectivity classifications this is the case whenever a sentence contains at least strongly subjective clues that can be identified by the described collection. Clues are strongly subjective if the collection of words and n-grams they are extracted from does not contain ambiguous classification results of this clue. Sentences are classified as objective whenever strong subjectivity clues are entirely absent from a sentence. Sentences to which neither of the two criteria apply remain unclassified at this stage. Using this technique, classification accuracy of the subjectivity classifier lies at 91.5% with a recall of 31.5% and the objectivity classifier yields accuracy of 82.6% at a recall rate of 16.4%. For the subsequent learning process of extraction patterns from those sentences that were classified, a learning algorithm based on AutoSlog-TS [65] is employed. The extraction distinguishes between *relevant* extraction pattern, i.e. those of subjective phrases, and *irrelevant* extraction patterns, i.e. those of objective phrases. The learning process consists of two steps. The first step extracts every possible instantiation of all extraction pattern considered that appear in the text that is analysed. For example, an instantiation of the extraction pattern *<subj> passive verb* may be *<subj> was satisfied*. In a second step statistics are gathered for how often each of the learned extraction patterns occurs in subjective versus objective sentences. The extraction patterns found and the frequencies they occurred with are ranked using a conditional probability measure, i.e. the probability that a sentence is subjective given that a specific extraction pattern appears in it:

$$Pr(subjective|pattern_i) = \frac{subj\ freq(pattern_i)}{freq(pattern_i)} \tag{3.24}$$

where $subj\ freq(pattern_i)$ is the frequency of $pattern_i$ in training sentences that are subjective and $freq(pattern_i)$ is the frequency of $pattern_i$ in all training sentences. As a last step, only those extraction pattern are chosen to represent subjectivity that are larger than some threshold ϕ.

The extracted patterns are then added to the initial set of n-grams and the procedure is repeated, using the now extended set of annotated training data. In this way, a collection of subjective and objective patterns is iteratively built up to eventually form a lexicon comprised of n-grams of varying length.

3.3 Reinforcement Learning and Conditional Random Fields

Up to this date, Reinforcement learning has sparsely been introduced to Sentiment Analysis. Among more established Supervised learning and Unsupervised Learning algorithms, few scientists have looked to Reinforcement Learning to provide algorithms that might solve problems in Sentiment Analysis. Among a few others, a notable exception is marked by Conditional Random Fields (CRF), which have been applied to the field by a number of scientists (e.g. [5], [9], [84], [88]). To provide an intuition as to how Reinforcement Learning may be utilised within Sentiment Analysis, a short account is presented of a CRF application developed by Yejin Choi and colleagues [9].

The following explanations of CRF are adapted from [38], [73] and [77]. CRF represent a conditional distribution $p(y|x)$ as undirected graphical models for building probabilistic models to split and label sequence data.

Generally, when dealing with Markov models, we need to assume the Markov property to avoid having to define the complete probability distribution, i.e. define probabilities for all combinations of past and future events in a Markov chain, i.e.

$$Pr\{s_{t+1} = s', r_{t+1} = r|s_t, a_t, r_t, s_{t-1}, ..., r_1, s_0, a_0\} \tag{3.25}$$

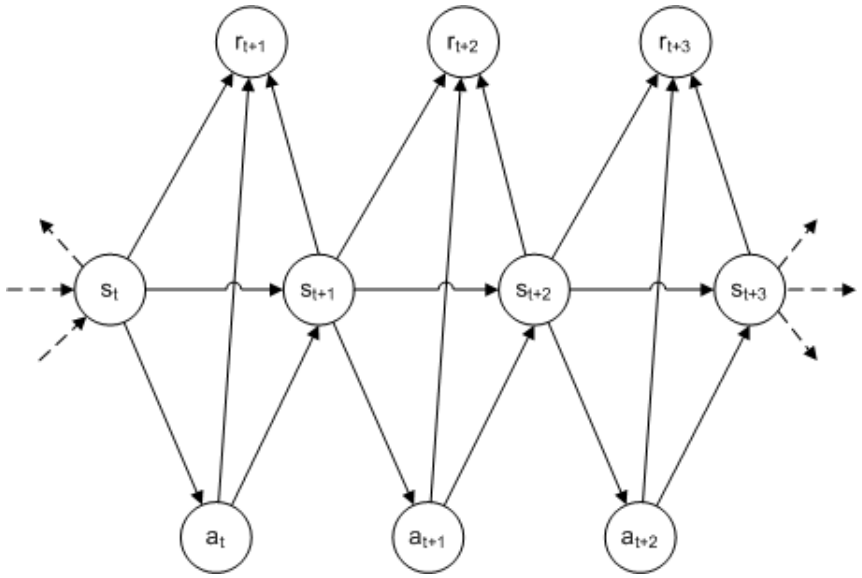

Figure 3.3: Graphical representation of Markov decision process, where s=state, a=action, r=reward.

where s is the current state, a are the actions available when in state s and r is the reward for taking action a when in state s. In the context of text analysis such as Sentiment Analysis, we are not concerned with time, but states are rather words at certain positions in text along which we move as we would along states in a temporal sequence.

The Markov property states that an environment's response at $t+1$ depends only on the state s_t and the action a_t:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{3.26}$$

We thus assume that vertices, i.e. variables, not directly connected are conditionally independent. In terms of the graph in figure 3.3 this means, we only consider maximally connected cliques at each one time and thus we obtain one potential function for each maximally connected clique contained in the graph. This ensures that problems remain tractable, but also limits the model in not allowing long range dependencies of entities. This, however, is desired in tasks such as Sentiment Analysis, since an assumption that a word's sentiment depends only on its successor's characteristics is arguably not valid.

CRF offer reconciliation to this issue because, unlike other models, such as Hidden Markov models (HMM), they do not take a generative approach but are rather conditional models that specify the probabilities of possible label sequences given an observation sequence. Therefore, the model does not expend effort on modelling the observations, which are fixed at test time anyway.

Lafferty and colleagues define the probability of a label sequence y, e.g. a succession of *opinionated* and *non opinionated* tags, given an observation sequence x, e.g. a collection of phrases, as a normalised product of potential functions of the form

$$exp(\sum_j \lambda_j t_j(y_i - 1, y_i, \mathbf{x}, i) + \sum_k \mu_k s_k(y_i, \mathbf{x}, i)) \tag{3.27}$$

"where $t_j(y_i - 1, y_i, \mathbf{x}, i)$ is a transition function of the whole observation sequence and the labels at positions i and $i-1$ in the label sequence; $s_k(y_i, \mathbf{x}, i)$ is a state feature function of the label at position i and the observation sequence." ([77]) The adjustable parameters λ_j and μ_k are estimated from training data.

In [5], Breck and colleagues use CRF to distinguish opinionated text from non opinionated con-

tent based on the MPQA corpus (see section 5.1.1) which has been used in this project for the construction of the corpus. They also collect a number of features describing the text, among which are syntactical features determined by a POS tagging system, and have a CRF algorithm subsequently classify expressions according to those features.

Choi and colleagues [9] apply CRF not to identify opinions but rather to find sources of opinions. Using various features that determine syntactic, semantic, and orthographic lexical characteristics of text, as well as dependency parse features and opinion recognition features they train a CRF to identify both sources of direct and indirect opinions. The features that were used include

- Capitalisation features: two boolean features to represent capitalisation, *all-capital* and *initial-capital*
- POS features: Representing lexical categories, acquired using the GATE POS tagger ([13])
- Opinion lexicon features: A boolean value representing whether a word is included in a predefined set of opinionated words
- Syntactic chunking: groups of words, i.e. chunks of the text, are identified and assigned grammatical roles, e.g. *subject*, *object*, etc.
- Opinion word propagation: Identified chunks that contain a word that is member of the set of predefined opinionated words are defined as opinionated altogether
- Dependency tree features: Encodes structural information between words and checks for grammatical relations to other opinionated words which is devised from the previous two items of information
- Semantic class feature: Assigns words to one of seven classes identifying their semantic properties with regards to what sort of source the word represents. The classes are *authority*, *government*, *human*, *media*, *organisation_or_company*, *proper_name* and *other*, where other includes 13 semantic classes that cannot be sources, such as *vehicle* and *time*.

These features are conjoined by the CRF to generate a set of labels, $X_1, ...X_n$ with possible values *'S'*, *'T'* and *'-'* for a sequence of tokens, $Y_1, ...Y_n$. *'S'* denotes the first token of a source, *'T'* is assigned to non-initial tokens of a source and *'-'* is assigned to tokens that are not part of any source.

For a comprehensive introduction to Reinforcement Learning, see [73], for more on Conditional Random Fields, refer to [38] or [72].

4

Argumentation in Machine Learning and Sentiment Analysis

Argumentation has, to this date, not been used within the setting of Sentiment Analysis and has to my knowledge neither, with few exceptions, been applied in unison with Machine Learning techniques in other disciplines. The noted exceptions are [47], [48], [49], and [86]. The research described in these publications has focused upon developing a technique named Argument Based Machine Learning (ABML), in which classical Supervised Learning algorithms such as the CN2 algorithm (see [11] for details) are combined with defeasible argumentation to enhance the performance of the used algorithms. Before turning to a description of the ABML algorithm in the following, originally described in [47], a brief introduction to defeasible argumentation is presented, which is described in detail in [62].

4.1 Defeasible Argumentation

Prakken and Vreesvijk define defeasible argumentation as follows:

> "Logic investigates patterns of correct reasoning. ['Defeasible argumentation' describes] logics for a particular group of reasoning patterns, viz. those where arguments for and against a certain claim are produced and evaluated, to test the tenability of the claim." [62]

Prakken and Vreesvijk present a broad conceptual sketch of logics systems for defeasible argumentation followed by a more detailed description of features of such systems, as well as a number of examples. This section will restrict itself to a summarised reiteration of what is presented on a conceptual level in [62], followed by an example of just one of many possible syntactical representations.

Logics systems for defeasible argumentation are generally made up of five elements, not all of which may be explicitly specified:

1. An underlying logical language
2. Definitions of an argument
3. Definitions of conflicts between arguments
4. Definitions of defeat among arguments
5. A definition of the assessment of arguments

The logical language underlying the system may be partially or fully unspecified, rather turning the system into a framework than an actual system. An argument corresponds to a proof, or the existence of a proof, in the underlying logic and may be in one of three forms:

1. Arguments may be defined as a tree of inferences grounded in the premises

2. Arguments may be defined as a sequence of inferences as described in the previous point

3. Systems may define an argument as a premise-conclusion pair, implicitly handing over the validation of a proof of the conclusion from the premise to the underlying logic

Conflicts, or, alternatively, attacks or counterarguments, in defeasible argumentation usually come in one of two types:

1. Rebutting

2. Undercutting

Rebutting describes arguments that present opposing conclusions, for example "Tweety flies because it is a bird" versus "Tweety does not fly because it is a penguin", thus describes a symmetric conflict. Undercutting, on the other hand, presents asymmetric conflicts, which may take one of two possible forms. In one case, a counterargument undercuts an argument by proving a statement that is claimed not to be provable by the attacked argument. For example, the argument "Tweety flies because it is a bird, and it is not provable that Tweety is a penguin" is undercut by the argument "Tweety is a penguin". Another way of undercutting an argument is to attack the link between premises and the associated conclusion. This is only possible if the attacked argument is non-deductive. For example, the inductive argument "Raven 101 is black since I observed that ravens raven1, raven2, . . ., raven100 were black" is undercut by the argument "I saw raven102, which was white".

How an argumentation process is evaluated is captured by the definitions of defeat among arguments and a definition of the assessment of arguments. Defeat may, for example, be defined via a notion of comparing the strength of arguments attacking each other, i.e. a counterargument defeats the attacked argument if it is, by some definition, stronger. The intensity of a defeat itself may also be subdivided according to how definitive the defeat is. Finally, the assessment of an argument does not exclusively depend on it being defeated once, because it may be reinstated at a later stage of an argumentation process. Argument B, "Tweety does not fly because it is penguin" may defeat argument A stating that Tweety is able to fly. But if argument C then defeats argument B by disproving the argument that Tweety is a penguin, argument A is reinstated by argument C. Due to such and other cases, possibly arising during an argumentation process, the final assessment of the arguments may not be carried out until all available arguments have been assessed.

One representation commonly used to construct rules from which arguments are deduced is the following:

IF *Complex* THEN *Class*

where *Complex* is a conjunction of simple conditions, so called selectors. These selectors, in turn, specify attribute values. Picking up the previous example about Tweety, an rule may take the following form:

IF *IsBird* = *true* AND *Species* = *penguin* THEN *CanFly* = *false*

As we will see below, ABML creates arguments from rules that exhibit the syntax that is shown above.

4.2 ABML

As explained above, Machine Learning and Argumentation have traditionally been rather disparate fields of research and development. Not until recently have the potential merits of combining both techniques been explored, when Argument Based Machine Learning (ABML) was introduced in [47]. This algorithm unifies Supervised Learning with argumentation as laid out above by constructing arguments devised from rules of the form

IF *Complex* THEN *Class*

and influencing the classification procedure with those arguments. The syntax of the resulting arguments is elucidated below. ABML uses the CN2 algorithm (see [11]), which is extended to form the ABCN2 learning algorithm. Usually, Supervised Learning techniques take a preferably large number of training examples and, using these, try to find a theory, or hypothesis, that adequately explains the training examples and then correctly classifies new cases. Within the framework of ABML, some of the training examples have associated with them an argument explaining the reasoning behind why an example is classified the way it is. Consider the following example of a bank approving or denying credit, $C(CreditApproved) \in (Yes, No)$, used by Mozina and colleagues. Table 4.1 shows some training examples without the added arguments.

The CN2 algorithm takes such examples as input in the form of a pair $E = (A, C)$, where A is an attribute-value vector, e.g. (*Name = Mrs Brown, PaysRegularly = No, Rich = Yes, HairColour = Blond*) and C is the class the example belongs to, e.g. (*CreditApproved = Yes*). The ABCN2 algorithm accepts such examples, as well, but in addition is also able to process examples of the form $AE = (A, C, Arguments)$, where $Arguments$ is a set of arguments $Arg_1, Arg_2, ..., Arg_n$. Arg_i can take one of two forms, either

C because *Reasons*

or

C despite *Reasons*

where *Reasons* may be a conjunction of reasons:

$$Reasons = r_1 \wedge r_2 \wedge ... \wedge r_n. \tag{4.1}$$

Each of the reasons r_i may take one of five forms, given r_i is part of a positive argument (the explanations for negative arguments are exactly opposite):

1. $X = x_i$: Value x_i of attribute X is the reason why the example is a class as given. This is the only form allowed for discrete attributes and will thus be the only of concern in the following.
2. $X > x_i (X >= x_i)$: As the value of attribute X of the example is higher than (or equal to) x_i, this is the reason for the class value.
3. $X < x_i (X <= x_i)$: The opposite to form 2.
4. $X > (X >=)$: X is *high*; similar to form 2 with the difference being the lack of knowledge about a threshold value which thus has to be found by the ABCN2 algorithm.
5. $X < (X <=)$: X is *low*; the opposite to form 4.

For the ABCN2 algorithm, the previous example may have the following form:

((PaysRegularly = no,Rich = yes, HairColor = blond),
CreditApproved = yes, {CreditApproved = yes because
Rich = yes ∧ CreditApproved = yes despite PaysRegularly = no})

Name	PaysRegularly	Rich	HairColour	CreditApproved
Mrs Brown	No	Yes	Blond	Yes
Mr Grey	No	No	Grey	No
Miss White	Yes	No	Blond	Yes

Table 4.1: Training Examples for ABCN2 learning algorithm

The example specifies a positive and a negative argument, e.g. *because* and *despite*. As seen, *Reasons* is a conjunction of reasons, $Reasons = r_1 \wedge r_2 \wedge ... \wedge r_n$, where each r_i may take varying forms, depending on the attributes of the examples that a reason refers to (see [47] for details).

The arguments, i.e. *Reasons*, described above are acquired via experts labelling the original training data of the form $E = (A, C)$ to form training data of which a subset has the form of $AE = (A, C, Arguments)$. This allows the development of reliable arguments which do not have to be applicable to the entire domain, but rather just single training examples.

This is helpful because introducing arguments or rules that have to cover the entire training set may be considerably harder, and a generalisation to newly incoming data may entail problems, as well. Take, from the table above, Miss White: We could argue that she is approved credit because she pays regularly i.e.

((PaysRegularly = yes,Rich = no, HairColor = blond), CreditApproved = yes, {CreditApproved = yes because PaysRegularly = yes})

The rule covers this particular example, but applied to Mrs Brown, this rule would fail because there are other reasons for granting Mrs Brown credit, namely the fact that she is rich. Associating arguments with just single training examples thus allows for covering reasons for classification in a quite detailed manner. From this new collection of training examples, a classifier is learned in a similar manner as by the CN2 algorithm. Roughly, in a first step rules are applied that cover, i.e. explain, as many examples as possible.

The definitions of a rule covering training examples is where the ABCN2 algorithm differs from the CN2 algorithm it is based upon. With the CN2 algorithm, "a rule covers an example if the condition part of the rule is true for this example." [47] For the ABCN2 algorithm we say that a rule has to *AB-cover* an example and it does so if three conditions are fulfilled:

1. All conditions in rule R are true for argument E. This is the the condition equivalent to the CN2 algorithm.

2. Rule R is consistent with at least one positive argument of example E.

3. Rule R is *not* consistent with any of the negative arguments of example E.

Accordingly, the main difference between the CN2 and the ABCN2 algorithm is that the covering process is more complex for the ABCN2 algorithm, since rules not only have to cover the features of an example, but also have to agree with the arguments attached to the example. Whenever examples are covered by a rule, these examples are removed from the training set. This process is repeated until all examples are covered. This means that with each iteration a new rule is introduced that covers as many of the examples as possible, argumented or not. Accordingly, the more diverse the data set proves to be, the more rules are needed to cover the entire set of training examples while a more homogenous training set needs fewer rules to be fully covered, i.e. explained.

One field Mozina and colleagues have applied their ABCN2 algorithm to is that of legal discourse. In [48], ABCN2 is trained on a data set that is concerned with a fictional welfare benefits issue. [48] defines the problem to be solved by the system as follows:

> The benefit is payable if six conditions are satisfied. These conditions were chosen to represent different kinds of condition that are found in the legal domain, so that we can see whether the different form of conditions affects their discoverability. [...] These conditions represent a range of typical condition types: [Two] are Boolean necessary conditions [..., one] is a threshold on a continuous variable representing a necessary condition, and [one] relates five Boolean variable, only four of which need to be true. [The remaining two conditions] relate the relevance of one variable to the value of another.

2400 example cases were randomly split into a training set containing 70% of the cases and a test set containing the remaining 30%. Half of the cases satisfied all conditions and the other half did not. All cases were comprised of both data relevant to a decision on the conditions and irrelevant data. From the training set, a first set of rules was generated using the CN2 algorithm, i.e. at this point no arguments were involved in the generation process. In a second step, *problematic* examples, i.e. outliers were identified and rules were added to them. These rules were then induced on the training set using the ABCN2 algorithm. After this second iteration, new rules were added to more outliers and the procedure was repeated. Thus each iteration following the first pass through the training data left the features of the examples untouched, but introduced rules that provided a framework for classifying a number of *problematic* cases correctly. Since the arguments were attached to training examples that the CN2 algorithm was not able to classify correctly, the system's performance was increased by diversifying the sources of information about the training data upon which its classification was based. After seven iterations of adding arguments and training, the accuracy on the test set amounted to 99.8%.

4.3 From ABML to A-SVM

Though ABML and A-SVM differ significantly from one another, as the subsequent chapters will show, some important parallels can be found and deserve mentioning to set the scene for the details on A-SVM that follow. The basic training set of ABML consists of a large number of classified entities that exhibit a certain combination of features. Recall the following example:

$$((PaysRegularly = yes,\ Rich = no,\ HairColor = blond),\ CreditApproved = yes)$$

Here, we have three feature variables and a class label, each of the four assigned with a value. As will become apparent below, this representation is basically equivalent to the representation chosen for the data processed by A-SVM. A-SVM is trained on a large number of feature vectors, i.e. a collection of instances that have feature values and a class label associated with them. Both in ABML and A-SVM, a human annotator reasons upon these combinations of feature values. The annotator chooses one or more features that either rectify the class this instance is assigned to or that indicate an opposite classification. In ABML the annotator chooses features one by one, as many as he or she deems appropriate. In A-SVM, the annotator instead chooses a single combination of features, i.e. a succession of words appearing in the n-gram the user reasons upon. This process is nevertheless very similar.

The main difference between the two algorithms can be found in the way of utilising the arguments supplied by the user. For ABML, the arguments are attached to the training instances. Subsequently, by means of the covering algorithm, these arguments are integrated into the training procedure to construct classification rules. Whenever a rule is proposed during training, it is checked how many training instances this particular rule covers. This includes both the feature values and the argument attached to the training instance. By these means, the training procedure relies on whatever the human annotator has attached to the training instances. For A-SVM, on the other hand, training the classifier and putting the arguments supplied by the annotator to use are two separate processes. As explained in detail in chapter 6, A-SVM yields two classification results, one after SVM classification and before argumentation and one after both processes have been executed. Thus, instead of influencing the training of the classifier, or even the classification

procedure itself, the arguments in A-SVM fulfil more of a correcting function of SVM classification results. The user provides reasoning upon a combination of words by classifying it and indicating the part of the n-gram that he or she deems responsible for the chosen classification. In ABML, the classification is given to the annotator. Accordingly, the annotator has no choice over the class but rather performs a matching process between the predetermined class and the features that may explain this classification.

5

Building a text corpus

In this project I have developed a system that employs different techniques in unison with the goal of enhancing performance in comparison to a similar unimodal approach to Sentiment Analysis. The classifier developed uses SVM, explained in section 3.1 , which are, as highlighted before, a Supervised Learning technique. Any such technique in Machine Learning relies on a training set being available. Upon this set the classifier is then trained according to the algorithm that is applied and, once this training is completed, the classifier categorises new input following the structures found in the training set. In accordance with this concept, the first and foremost task of the project was to develop a training set that would allow training a classifier to an extent where it could subsequently perform quality classification on new and unannotated text. This chapter describes the development of an extensive text corpus of roughly 13,000 semi-automatically annotated n-grams which was then used to train the SVM that classifies new text within the final system. The corpus was constructed using a number of sources, each of which contributed either text, features annotating this text or both. As explained, the resulting corpus is made up of n-grams, each of which is represented by a vector of features. The classifier is trained on this collection of feature vectors.

The maximum length of n-grams in the corpus is five words. This value was chosen based on the intuition that limiting the possible size of the n-grams too far would hamper the ability to grasp the role that the context in which a word appears plays. At the same time, allowing n-grams that are too large may have posed computational problems because the larger the n-grams are the larger are the feature vectors and the more extensive is the subsequent classification procedure. Thus, the maximum size of five words was chosen as a compromise, respecting the trade off between infeasible computational demands that large n-grams may pose and hampering the corpus quality by only allowing very short n-grams. Whether a maximum length of five words is an optimal choice of parameter will have to be investigated further, along with a number of other adjustable parameters, as is explained in section 8.2.5. Each n-gram is annotated with the following features:

- The size of the n-gram
- Three scores, one for positivity, one for neutrality and one for negativity
- All types of words that appear within an n-gram

Accordingly, each feature vector is comprised of up to nine features, one determining the size of the n-gram, three representing scores and between one and five features representing the word types that appear in an n-gram. The word types categorise all words within an n-gram according to their lexical types, as discussed in section 5.1.3. In addition to the features, each line contains a class label, $c \in \{1, -1\}$, for the particular n-gram, identifying it as either *opinionated* or *non opinionated*.

5.1 Sources

The development of the corpus was a task of taking a number of sources offering text and valuable information about text and merging the content of these sources into a single corpus on which a

classifier could be trained. The following three sources of information and content were used to build the corpus:

1. MPQA corpus [82]
2. SentiWordNet [20]
3. TreeTagger [70]

The rest of this chapter is devoted to first elaborating upon these three systems separately and subsequently giving an account of how these sources have been brought together to form a large collection of classified vectors containing different combinations of features.

5.1.1 The MPQA corpus

The MPQA corpus [82] is a collection of 378 news articles, comprising around 10,000 sentences, each of which has been manually annotated with tags describing a number of subjectivity measures, as well as sources and objects of opinions within the text. The annotation scheme, proposed by Wiebe and colleagues, used to develop the MPQA corpus annotates the texts at word and phrase level. It thus applies relatively fine grained information about the articles annotated. Wiebe et al. use the term *private state* to describe expressions of opinion, which is a term common to the field. For their annotation scheme, they propose a *private state frame* which assigns to each private state annotated in the corpus a number of parameter values. These include the source of the private state, i.e. who expresses the state, the target of this private state, i.e. who the sentiment is directed towards, and "various properties involving intensity, significance, and type of attitude" [82]. The MPQA corpus offers rather diverse annotations which is owed to the way the annotations were contrived. All annotations were done manually and the annotators were given annotation guidelines that were rather loose and left the annotators which large room of choice in their annotation. The resulting annotations are highly valuable in regards to capturing many aspects of the subjectivity expressed and especially paying tribute to the context. However, this intricacy brings with it a high demand in processing and working with the corpus.

As the main focus of this project lay on investigating the merits of performing Sentiment Analysis using a combination of input channels rather than finding ways of disseminating highly complex annotations, as a first step, contents were extracted from the MPQA corpus into a more rudimentary format, only retaining some of the annotations. The prime information that was provided by the MPQA corpus were the n-grams themselves, each annotated with a label classifying it as either opinionated or non opinionated. To extract the content needed from the MPQA corpus, it was necessary to extract content from 756 different files, one file for each newspaper article and one file for each of the articles' annotations. The files containing the articles themselves were in plain text, while the annotation files were comprised of a number of annotation lines, each of which supplying information about an g-ram contained in the article. To illustrate the structure of the MPQA corpus and how it was used in building a new corpus, take to following sentence, taken from one of the news articles that the MPQA corpus is comprised of:

While President Chen Shui-bian has mellowed his stance on independence, he has refused to bow to the "one-China policy", a precondition for dialogue.

A sentence such as this may have numerous annotations associated with it, i.e. the annotation of this one sentence may span multiple lines in the annotation file. In the case of this sentence, one of the annotations associated with it looks as follows:

1265 1258,1276 string GATE_ on on-strength="high" nested-source="w, bian" overall-strength="high" onlyfactive="no"

Annotation type	Explanation
GATE_on	The annotated n-gram is nestled within a direct private state
GATE_agent	N-gram refers to sources of private states & speech events
GATE_expressive-subjectivity	N-gram expresses private states indirectly
GATE_direct-subjective	N-gram directly mentions private state
GATE_objective speech event	N-gram does not express any private state
GATE_attitude	N-gram contains attitude
GATE_target	N-gram marks the target of an attitude
GATE_split	Marks the end of an excerpt

Table 5.1: Annotation types found in the MPQA corpus

This annotation provides information about the part of the sentence which reads *"...has refused to bow..."*, i.e. an n-gram of length four. Let us consider all contents of this annotation one by one, starting with the first four digits, *1265*. This number is the ID of the annotation with which it can be unambiguously identified. This ID, though, is only unique within a single annotation file and thus looses its relevance once multiple annotation files are processed as one. The following digits, *1258,1276*, are the span of the annotation. The digits preceding the comma signify the starting byte of the n-gram within the original text, i.e. the example n-gram starts at byte *1258* in the text file, and it ends at byte *1276*, the digits following the comma. The item *"string"* describes the data type of the annotated content, which is the same for all n-grams used within the newly constructed corpus. *"GATE_on"* determines the annotation type, which can generally have one of eight types, listed and explained in table 5.1. Each of the annotation types can have a number of attributes, which make up the remainder of an annotation line. For a detailed description of all attributes that each annotation type can have, refer to [21].

As mentioned, not all of the parameters contained within the annotations were used in constructing the new corpus. To construct the basis of the corpus I focused on the digits identifying the position of the annotated n-gram in the original text as well as the annotation type and attribute values. The latter two allowed me to determine the class of the n-gram while the position identifiers made it possible to extract the n-gram itself from the original text file. Accordingly, processing each annotation file, together with the file containing the appropriate news article, yielded a number of n-grams annotated with a class label. Applying this to the example, the result would be as shown below:

1 has refused to bow

This tells us that the n-gram *"has refused to bow"* is opinionated, which is all the information that we need to extract from the MPQA corpus. The features, which make up the feature vectors that are processed by the system, are added step by step using other sources, which are explained below.

5.1.2 SentiWordNet

SentiWordNet, developed by Esuli and Sebastiani [20] is a lexical resource that has been specifically designed for the purpose of aiding Sentiment Analysis. It is a lexicon based on and similar to WordNet [45], but extended with lexical information about the sentiment of each synset contained in WordNet. Synsets are groups of words that are synonymous to each other, thus express the same idea, describe the same entity or carry the same meaning in another respect. The additional information that is provided by SentiWordNet, but not present in WordNet, comes in the form of three different values, positivity, objectivity and negativity, which sum to one and describe the orientation of sentiment. Through these three measure, each synset in the lexicon is associated with a fine grained categorisation of the nature of its sentiment. One of the resources provided by the developers of SentiWordNet is a single text file containing the entire lexicon, including all its annotations. For the project, this file was split into 26 smaller files, one for each letter in the

alphabet. Each file contains all words starting with the letter designated to this very file. Splitting the lexicon in this way allowed faster search through the lexicon at later stages.

The developers of SentiWordNet chose the annotation of synsets over terms based on the assumption that "different senses of the same term may have different opinion-related properties." [20] This means that the same word may come up in the lexicon multiple times, specifically as often as it is part of a synset. The process of building SentiWordNet was comprised of training a number or ternary classifiers, i.e. classifiers that determine each training instance as belonging to one of three classes, in this case positivity, neutrality and negativity. All classifiers used yielded similar performance, i.e. classification accuracy, but exhibited different characteristics and qualities in their classification. "Each ternary classifier differs from the other in the training set used to train it and in the learning device used to train it, thus producing different classification results of the WordNet synsets" [20] they were used to classify. The three scores that are associated with each synset, and used for the project corpus, result from the normalised proportion of classifiers that have assigned the according label to it. For example, if half of all classifiers have assigned a synset to the category *negative* and the other half have assigned it to the category *neutral*, the scores would be the following:

- Positivity: 0.0
- Neutrality: 0.5
- Negativity: 0.5

Each classifier is generated using semi-supervised learning techniques, meaning that a data set is automatically built from a small set of manually annotated examples. The initial set of manually annotated words is iteratively expanded by identifying the synsets of the words contained in the current training set with the same class label as the word it was found through. In addition to that all synsets that are connected to one of the synsets in the current training data via an antonymy relation are added to the training data carrying the opposite class label. This procedure is used to identify both positive and negative terms. All synsets found are then added to the training set and the procedure is repeated. The procedure for finding objective, or non opinionated terms, differs from the one described above. The set of objective terms is constructed from synsets that do not belong to either of the other two categories, and to make this classification more robust, are also "not marked as either positive or negative in the General Inquirer lexicon [71]." [20]

The training data sets acquired using this technique are then used to train standard supervised learning algorithms. For each set of training data, two classifiers are trained, one distinguishing between terms being *positive* and being ***not*** *positive* and the other classifier classifying terms as either *negative* or ***not*** *negative*. The results of the two classifiers are combined to place each term in one of three categories:

- Positive: both classifiers have classified the term as positive
- Negative: bot classifiers have classified the term as negative
- Objective: One classifier has classified the term as positive, the other as negative

The classifications that each classifier produces through this procedure are then accumulated and normalised for all three values to form the final SWN score as it is represented in the SentiWordNet lexicon. In addition to the simple text file that is described above and used for the purposes of the project, the developers of SentiWordNet also offer a lexicon application that visualises the SWN scores as it is shown in figure 5.1. We can see a triangle of which the outer corners denote the extremes of SWN scores, i.e. placing the blue dot in the top left corner for a word sense would indicate that this particular word, used in a particular sense, is entirely positive, i.e. has the following SWN scores:

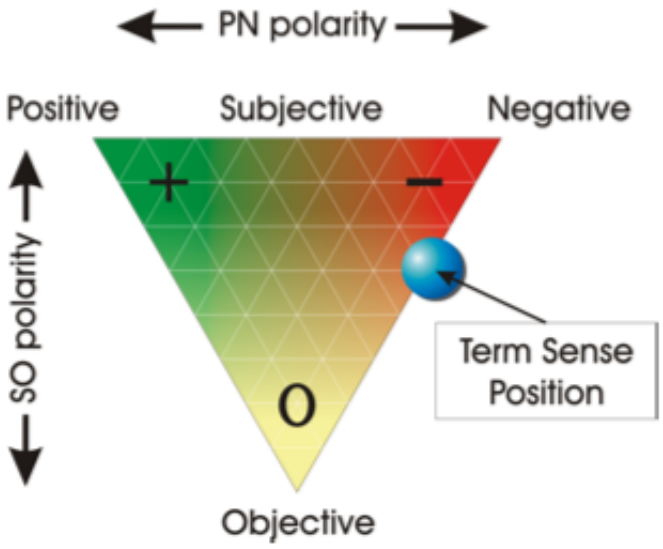

Figure 5.1: Graphical representation of SWN scores used in the SentiWordNet lexicon application (from [20])

- Positivity: 1.0
- Neutrality: 0.0
- Negativity: 0.0

Upon searching the lexicon for all instances of a word, the user is presented with all senses of the word, meaning every score of every synset that the searched word is a member of.

5.1.3 TreeTagger

The TreeTagger system [70] is a publicly available Part-Of-Speech (POS) tagging system that uses decision trees for probabilistically selecting POS annotations. The aim of using decision trees in tagging parts of speech is to allow taking the context into account in which the word that is tagged appears. The leaf nodes of the decision tree mark the actual decision on how to tag the concerning word or phrase while the higher nodes give information about the surrounding words. An example of such a decision tree, adapted from [70], is shown in figure 5.2. We can see in this example that at each level within the tree a decision is made whether or not the word of concern, its position identified in relation to the word that is analysed by the subscripts of *tag*, is of a certain type. The decision tree is constructed during a training phase and then, come time of tagging new text, passed through to attain the appropriate tags for the new text at hand.

The TreeTagger offers a number of parameters text can be annotated with and for this project the facilities of tagging words with their type, as explained in section 5.2, as well as their basic form were utilised.

TreeTagger has been developed to overcome shortcomings of many techniques that apply Markov models for POS tagging. "Because of the large number of parameters (particularly in the case of trigrams), these methods have difficulties in estimating small probabilities accurately from limited amounts of training data." [70] Both Markov models and TreeTagger estimate the probability of a certain combination of words as

$$p(w_1, ..., w_n, t_1, ..., t_n) := p(t_n|t_{n-2}t_{n-1})p(w_n|t_n)p(w_1, ..., w_{n-1}, t_1, ..., t_{n-1}) \tag{5.1}$$

but with TreeTagger, instead of estimating the transition probabilities via maximum likelihood estimation (MLE), the transition probabilities are estimated according to a decision tree.

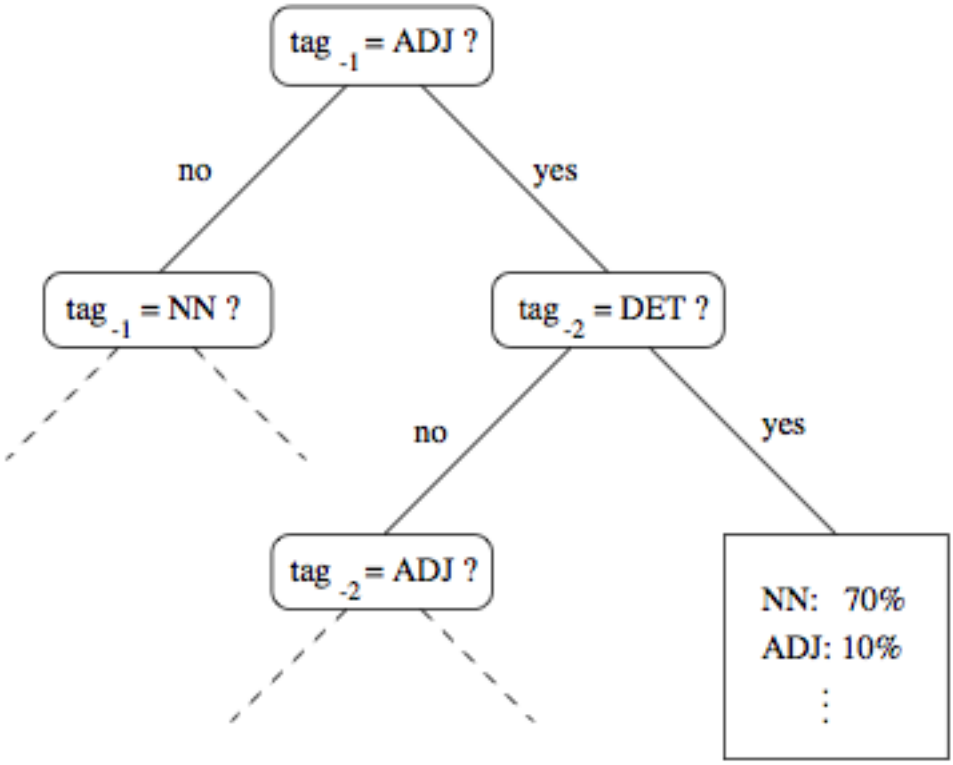

Figure 5.2: An example instance of a decision tree as used by TreeTagger

TreeTagger estimates probabilities of trigrams occurring in text and these probabilities are determined by following the appropriate path through a decision tree whose leave nodes hold probabilities of the word of concern within the trigram being of one type or another.

The decision tree along which the system decides how to tag words is built recursively during a training phase using a modified version of the ID3 algorithm (for details on ID3, see [46]). "In each recursion step, a test is created which divides the set of trigram samples in two subsets with maximal distinctness regarding the probability distribution of the third (predicted) tag. The test examines one of the two preceding tags and checks whether it is identical to a tag t. A test has the following form:" [70]

$$tag_{-i} = t; \quad i \in \{1, 2\}; \quad t \in T \tag{5.2}$$

where T is the tagset. At each recursion step, the node yielding the highest information gain for all possible tests on the training data is expanded. The criterion determining the information gain when comparing tests q is "the amount of information that is gained about the third tag by making each test. Maximising the information gain is equivalent to minimising the average amount of information I_q that is still needed to identify the third tag after the result of the test q is known:" [70]

$$I_q = -p(C_+|C) \sum_{t \in T} p(t|C_+) log_2 p(t|C_+) - p(C_-|C) \sum_{t \in T} p(t|C_-) log_2 p(t|C_-) \tag{5.3}$$

"Here, C is the context which corresponds to the current node and C_+ (C_-) is equal to C plus the condition that test q succeeds (fails). $p(C_+|C)$ ($p(C_-|C)$) is the probability that test q succeeds (fails) and $p(t|C_+)$ ($p(t|C_-)$) is the probability of the third tag t if the test succeeded (failed). These probabilities are estimated from frequencies with MLE." [70] The resulting decision tree is pruned to attain the final training result. When trained on trigrams, the TreeTagger achieves classification accuracy of 96.06%.

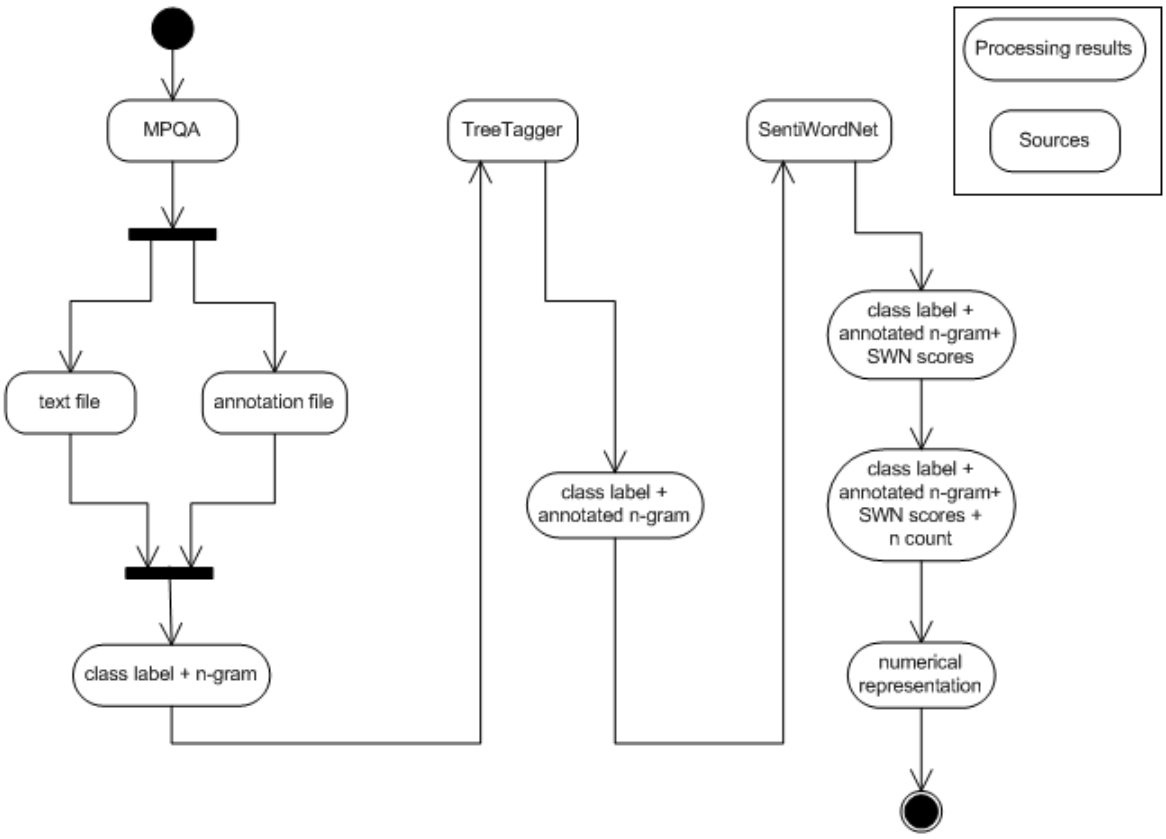

Figure 5.3: Schematic view of the corpus construction, including the sources from which n-grams and annotations are attained and the intermediate conversion results

5.2 Merging the sources into one

Each of the sources and systems described above contributed information to the construction of feature vectors describing n-grams, which were constructed in six steps, in each step adding information to an n-gram and finally attaining the final feature vector conforming to the syntax that is needed to process the data with libSVM [7], the system used for this project.

1. Extract n-grams from the MPQA corpus files, including their classification, and merge them into one file
2. Annotate the n-grams with lemmata and tokens using TreeTagger
3. Extract scores from SentiWordNet for each word in each n-gram of the corpus
4. Accumulate and normalise the scores within each n-gram
5. Count the number of words each n-gram is comprised of and add the result as a final feature
6. Convert the corpus resulting from the previous steps so that it is comprised entirely of numerical values, allowing procession with libSVM

In the following, these steps are shortly described, with a detailed account of the equivalent process of converting user input to feature vectors following in chapter 6. A schematic view of the steps involved in constructing the corpus is presented in figure 5.3.

The extraction of n-grams from the MPQA corpus was predominantly a process of matching content from the annotation files with content from the text files. This meant extracting those parts of the text files pointed to by the byte specifications, discarding contents of the annotation lines that would not be used in the corpus and finally creating a new annotation line containing the n-gram and its class label, i.e. opinionated or non opinionated, as determined by the MPQA annotations. Consider again the example given in section 2.3.4:

766 1722,1756 string GATE_ on nested-source="w" is-implicit="" onlyfactive="yes"

With all the components of this annotation explained in section 5.1.1, we are in a position to digest the content of the annotation fully. The first digits, the identifier of the annotation, are of no value at this stage because we do not consider single documents and as such, the identifier loses its relevance, being unique only within a single document. The following digits allow us to retrieve the exact n-gram from the original text that is described by this annotation. As it is the same for all n-grams extracted, the data type, *string*, is of no relevance to us and is discarded accordingly. The remainder of the annotation specifies characteristics of the n-gram, some of which allow us to classify it as either opinionated or non opinionated. In the case of the example, the annotation *onlyfactive="yes"* tells us that the n-gram specified by the byte identifiers is strictly factual. An annotation such as *onlyfactive="no"* would indicate the opposite. A number of feature values and annotation types were used to identify the opinionatedness of an n-gram. For example, if the label of an annotation was *GATE_ expressive-subjectivity*, it was classified as opinionated. In the case of the example, we extract the n-gram from the original text, ranging in this case from byte *1722* to *1756*, and assign it the class label -1, i.e. *non opinionated*. This process is repeated for every line of annotation of every news article. Not all annotations contain information about the opinionatedness of an excerpt of the article that is annotated. These annotations were ignored as they did not contribute to building a training corpus containing the information needed.

All these labelled n-grams ware stored in a single text file which is then further processed in the next step. This further processing means passing the resulting file to the TreeTagger program described above. Running the TreeTagger program on the collection of n-grams extracted from the MPQA corpus provides twofold annotation. The first annotation following each word is the type of the word, as determined by the TreeTagger. This type is one of 42 types, all of which developed for the Penn TreeBank [44], an extensive linguistic project, used within many other settings and developments since its release (e.g [4], [29], [70]). The second piece of annotation added to each word that is provided by the TreeTagger program is the basic form of the word, which proves a valuable information for a later step or processing, namely calculating the SWN score.

As described above, SentiWordNet offers, in addition to other lexical information, three scores for every entry in the lexicon, a positivity score, a neutrality score and a negativity score. For each word in each n-gram, the basic form of the word is searched for in the SentiWordNet data base. The values are returned and accumulated for each n-gram. These scores are added and then divided by the total number of words in the n-gram, i.e. they are normalised. The final feature that is added is simply the length of the n-gram divided by ten, i.e. n is added as another feature.

The final step is not concerned with adding more features to the n-grams, but with converting those features not yet numerical into a numerical representation. This is necessary to allow analysis with the SVM, as the libSVM system demands a strictly numerical representation of the feature vector.

The final corpus is thus available in two different formats. The first format is that in which the final step of conversion has not been applied and the n-grams are still present in their worded form. The second representation discards the n-grams themselves and represents combinations of features and their class label. This second format is then used to train the classifier.

6

The system

With the basic mechanisms and sources of content in place, the next step in the project was building a system that would integrate the parts described above into a single piece of software. This meant constructing an architecture that interacts with a user and processes data according to the outcome of this user interaction. This chapter presents A-SVM, a system that gathers text provided by a user and subsequently analyses this text in two passes. The first pass sees the data classified by the SVM while the second pass takes these results and adds to them arguments acquired via user feedback.

The system was developed in C++, the same language libSVM [7] is written in. The GUIs prompted during execution were written using Qt, a UI framework for C++ and other languages, publicly available and distributed by Nokia [53]. Both TreeTagger and libSVM were, whenever executed within the program, called as external procedures.

6.1 System architecture

Very broadly, the system is comprised of a succession of input gathering, input conversion and input classification. A schematic view of a run through the system is given in figure 6.1.

A number of text files, all of which described in detail in chapter 5, are used repeatedly throughout a run of the program. Among these files are all 26 SentiWordNet lexicon files and the text corpus developed specifically for this system. In addition to these text files, a model file used by the TreeTagger and a file containing the results of training the SVM on the text corpus, i.e. the support vectors, are used within the system. The model file needed for using TreeTagger serves the same purpose as the file containing the support vectors. It is a file containing the results of training the TreeTagger system on a corpus, in this case a large collection adapted from the Penn TreeBank [44]. Both these files are used whenever either component, libSVM or TreeTagger, is called to classify or annotate an excerpt of the text supplied by the user.

All intermediate results obtained throughout a system run as it is described below are stored within text files. This has been done for the purpose of having access to all results after a run has been executed, not just the final classification. This is more for illustrative and investigative purposes than practical ones concerning the system execution. Were the system to be viewed as a final, dispatch ready development, it might be both simpler and more efficient to discard this measure of storing all intermediate results in text files (see section 8.2.7).

On execution, the system prompts the user with the first graphical user interface (GUI), which is shown in figure 6.2. The user types or pastes the text he or she wants analysed into the text field. Once the user has done so and presses the *Analyse!* button, the text is stored and submitted to the first pass of analysis. The first stage of analysis consists of a number of steps, the final result being the SVM classification of each possible n-gram within the text supplied by the user up to a length of five words. To attain this result, the following actions are taken:

1. Split the cohesive text submitted by the user into lines of single words
2. Annotate the lines of words by running them through TreeTagger and formatting them

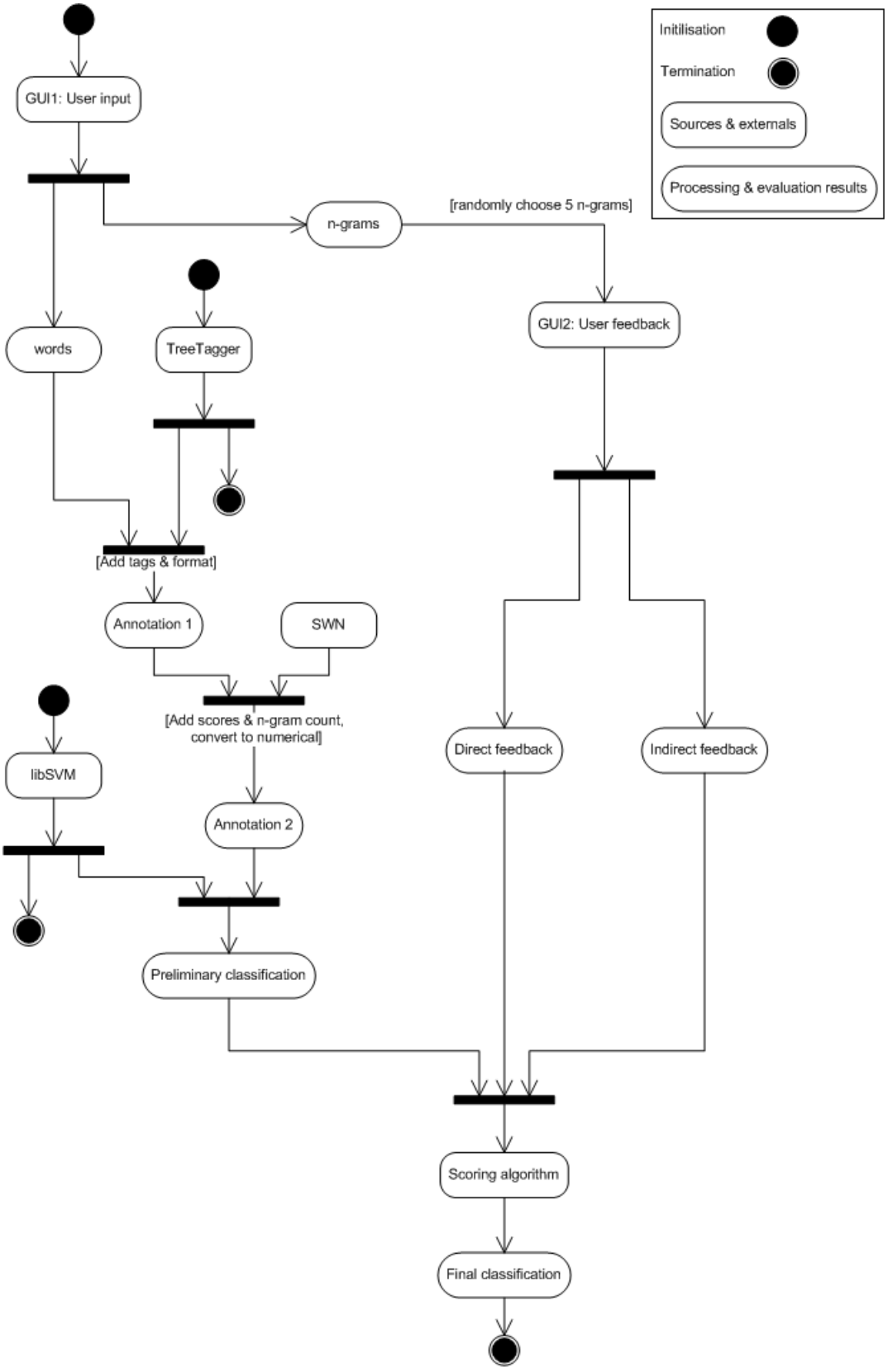

Figure 6.1: Schematic view of the system

Figure 6.2: GUI prompted to the user for text input

3. Calculate SWN scores for each word
4. Construct all possible n-grams up to a length of five words
5. Calculate SWN scores for each n-gram
6. Add n as a feature to each n-gram
7. Convert each line of now annotated n-grams to numerical values
8. Classify n-grams by running them through the trained SVM classifier

Before elaborating upon this process it is worth pointing out that the succession of analysis steps is rather similar to the actions that were performed when building and training the text corpus that is now used to classify the new text. This is the case, and intuitively sensible, because the outset and the goal of the analysis are the same in both cases. When building the corpus, n-grams were annotated and then analysed by the SVM. Now that new text is passed to the system to be analysed, the same steps of annotation and conversion have to be performed in order to obtain the same syntax for the user input and compare it to the result that sprang from training the SVM on the original corpus.

The first step in processing the user input in a manner that allows analysis of it is to break it up into single words, each of which is annotated on its own and then merged to form n-grams. The file containing one word per line is passed to and annotated with TreeTagger which, considering again the same n-gram as before, yields the following result:

has VHZ have
refused VVN refuse
to TO to
bow VV bow

Each line contains one word annotated with its type, or token, and its basic form, or lemma, the word and its annotations separated by a tab. The result of formatting this result is as follows:

has<VHZ>(have)
refused<VVN>(refuse)
to<TO>(to)
bow<VV>(bow)

With the basic form of each word in the user input at hand, the SWN scores are extracted from the SentiWordNet lexicon. A search procedure comprised of determining the correct lexicon file to search and then finding the word at hand and its according annotations was executed for each word. Whenever a word is found in the SentiWordNet lexicon, the according SWN scores are returned and stored alongside the word and the annotations already present. A word that is not found in the SentiWordNet lexicon is assigned with scores of zero, for we have no information about this word's SWN scores. Resulting from this search and annotate procedure are lines of single words now bearing five features each, the word's type and its basic form, together with scores for positivity, neutrality and negativity:

0.0 1.0 0.0 has<VHZ>(have)
0.0 0.625 0.375 refused<VVN>(refuse)
0.0 1.0 0.0 to<TO>(to)
0.0 0.5 0.5 bow<VV>(bow)

For reasons explained throughout earlier chapters, we want to analyse n-grams, not single words. Accordingly, the next step is the construction of these n-grams from the annotation result shown above. This step constructs each possible n-gram contained in the user input up to a length of five words. For the example, this means building ten n-grams. Generally, for any text larger than three words, we obtain $m = (n - 2) * 5$ n-grams when constructing n-grams up to a size of five words. When constructing the n-grams, some consideration needs to be placed on the SWN scores. Each n-gram that is constructed still has just one SWN score associated with it for positivity, one for neutrality and one for negativity. To obtain these scores the individual scores of the words are accumulated and normalised by dividing them by n. The result is, in part, as follows:

0.0 1.0 0.0 has<VHZ>(have)
0.0 0.8125 0.1875 has<VHZ>(have) refused<VVN>(refuse)
0.0 0.875 0.125 has<VHZ>(have) refused<VVN>(refuse) to<TO>(to)
0.0 0.78125 0.21875 has<VHZ>(have) refused<VVN>(refuse) to<TO>(to) bow<VV>(bow)
0.0 0.625 0.375 refused<VVN>(refuse)
0.0 0.8125 0.1875 refused<VVN>(refuse) to<TO>(to)
0.0 0.7083 0.2917 refused<VVN>(refuse) to<TO>(to) bow<VV>(bow)
...

Having constructed the n-grams, we can now add the final feature, i.e. the size of the n-gram:

1 0.0 1.0 0.0 has<VHZ>(have)
2 0.0 0.8125 0.1875 has<VHZ>(have) refused<VVN>(refuse)
3 0.0 0.875 0.125 has<VHZ>(have) refused<VVN>(refuse) to<TO>(to)
4 0.0 0.78125 0.21875 has<VHZ>(have) refused<VVN>(refuse) to<TO>(to) bow<VV>(bow)
1 0.0 0.625 0.375 refused<VVN>(refuse)
2 0.0 0.8125 0.1875 refused<VVN>(refuse) to<TO>(to)
3 0.0 0.7083 0.2917 refused<VVN>(refuse) to<TO>(to) bow<VV>(bow)
...

The final step before classifying the user input for the first time, using just the SVM, is a conversion of any non numerical values to numerical values and the addition of a position identifier for each feature:

1:0.1 2:0.0 3:1.0 4:0.0 5:0.9524
1:0.2 2:0.0 3:0.8125 4:0.1875 5:0.9524 6:0.8372
1:0.3 2:0.0 3:0.875 4:0.125 5:0.9524 6:0.8372 7:0.5581
1:0.4 2:0.0 3:0.78125 4:0.21875 5:0.9524 6:0.8372 7:0.5581 8:1
1:0.1 2:0.0 3:0.625 4:0.375 5:0.8372
1:0.2 2:0.0 3:0.8125 4:0.1875 5:0.8372 6:0.5581
1:0.3 2:0.0 3:0.7083 4:0.2917 5:0.8372 6:0.5581 7:1
...

To fully grasp how this final, numerical representation is processed, a few characteristics need further explanation. First, notice how the first feature, the n has changed. The original values are divided by ten. This is done because all values that are passed to the SVM should lie in the range between zero and one. The values are divided by ten instead of five to make extending the program to considering n-grams of a length of up to ten words an easy task, should this be considered in future developments. The values that have replaced the words and their annotation represent one of 42 possible word types and have no relation to the words themselves. This is the case because we are interested only in the features that describe the n-gram we are analysing, not the n-gram itself. Each value is one of 42 values, ranging, equally spaced, from zero to one. The final change in comparison to the previous representation that has been made is the addition of the aforementioned position identifiers followed by a colon and the particular feature value. This representation conforms to the syntax that we need to enable processing of the feature vectors by the SVM. The SVM used generally demands such position identifiers because it is possible to process sparse feature vectors with this SVM, as well. In the case of sparse data sets it is indispensable to signify which value describes which feature.

The result of the SVM classifying the feature vectors as described is a text file containing one class label, $c \in \{1, -1\}$, per line, thus classifying each n-gram as being either opinionated or non opinionated. This concludes the first run of classification, upon whose results the second run is built. The process of converting, annotating and classifying the user input takes place hidden from user sight and accordingly the next step in the program execution is, to the user, the second step. Just as the first run of classification starts with the user prompting input to the system, the second run does the same. The user is presented with a second GUI through which he or she is asked to provide the system with some, in the current version five, of their own classifications of randomly selected n-grams. In addition to judging these n-grams on their being opinionated or non opinionated, the user gives a justification as to why he or she classified each n-gram the way they did. This justification is given in the form of arguments, one per n-gram, in a syntax as it is described in section 4.2. Figure 6.3 shows this GUI. Once this feedback is submitted, the second run of classification is executed, which builds upon the first run by reclassifying a number of the SVM classification results according to the user feedback. As the process of reclassification based on the user feedback is rather intricate, the following section is devoted to it to adequately elucidate how the final classification result is attained.

As a last step in the program execution, the final classification result is presented to the user. As explained in section 2.3.4, this is a crucial aspect of any system whose success builds on producing results that are accessible to its user. Recall that there are two basic ways of summarising and representing the results of an analysis process such as this. I have not explored textually representing the outcome of the classification process during this project, though it may prove valuable in certain areas upon further investigation. Instead, the system presents the results to the user in a graphical manner by plotting scores that are kept for each word of the user input. These scores are devised from the final n-gram classification. A graphical representation of the n-gram classifications has been neglected for two reasons. Firstly, the n-grams multiply overlap and for this reason do not lend themselves for an intuitively understandable graphical representation. Furthermore, the n-grams do not represent the text in the same coherent manner in which the user passed the text to the system in the first place. This may lead to confusion since the user is never informed about the process of n-gram construction as it is followed by the system. Only when passing judgement

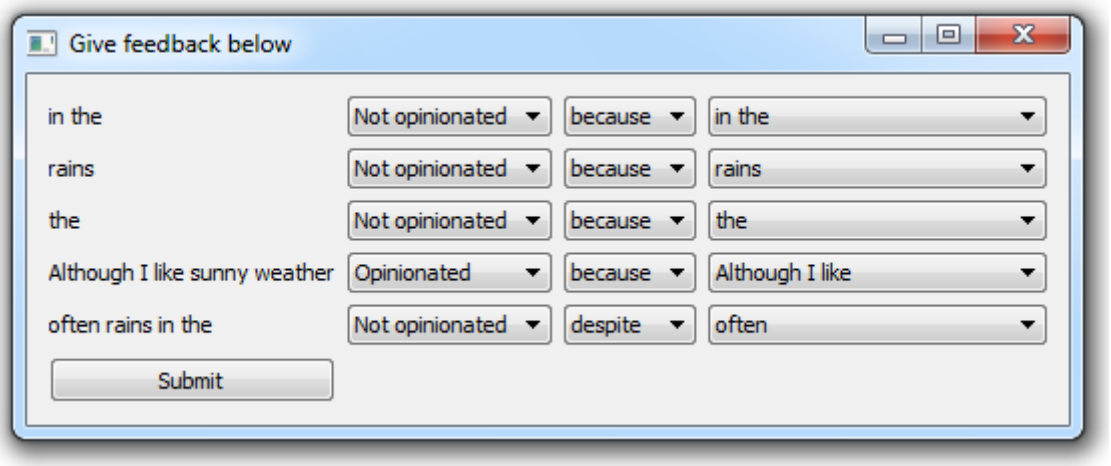

Figure 6.3: GUI prompted to the user for feedback input

on some n-grams is the user confronted with them, but at this stage, the n-grams are presented in isolation from most of the other n-grams and thus appear to simply be an excerpt of the user's input.

For these two reasons portraying the final classification to the user in relation to each word is a more intuitively understandable way and the process of accumulating the classification values for single words is explained below. However, it shall not go unmentioned that this choice of representation is just one of many fathomable for the sort of results that we obtain from this program. Some of these ways may be more appropriate than the one chosen, since parts of the information that may be garnered from the n-gram classifications are either left out or lost by breaking the classification down to word level. Nevertheless, representing classifications for single words and representing them in a graph is intuitively understandable for a user who is not familiar with the inner workings of the system or even Sentiment Analysis itself and thus constitutes a reasonable first choice at this point of the system's development. Other techniques that may prove to offer proper summarisation of the results are discussed in section 8.2.4.

In order to break down the n-gram classifications into single word classifications, for each word we have to consider all n-grams in which it appears. For each time a word appears in the user input it appears in up to 15 n-grams once the user input is split up. Each of these n-grams is classified by the system and the word of interest contributes to each of the classifications of these n-grams. For this reason I decided to accumulate the class labels of all n-grams a word appeared in, meaning that each word is assigned a score ranging from -15 to 15. Whenever the word of interest appears in an n-gram that is classified as opinionated the score of this word is increased by one, whenever it appears in an n-gram that is classified as non opinionated it is decreased by one. This means that the higher the score of a word the higher the certainty with which we can say that it is, in this context, opinionated, and the lower the score the higher the certainty that it is non opinionated. Whenever a score is closer to zero we can conclude that there have been ambiguities in the classification of this word, i.e. some n-grams that contain this word were classified as opinionated and some were classified as non opinionated. Figure 6.4 shows a graphical representation of both the classification results of the SVM and of A-SVM of one of the user inputs given during the evaluation described in section 7.2 (the result for the sentence shown in the GUIs above can be seen in figure 8.1):

> Although I like sunny weather it often rains in the United Kingdom during the winter.

A-SVM classifies the sentence as we might expect. The first few words express an opinion while the remainder of the sentence states facts about the UK weather. The SVM classification, on the other hand yields a rather ambiguous classification with a leaning towards opinionatedness for the entire sentence. The information we gain from the user feedback about the first few words agrees

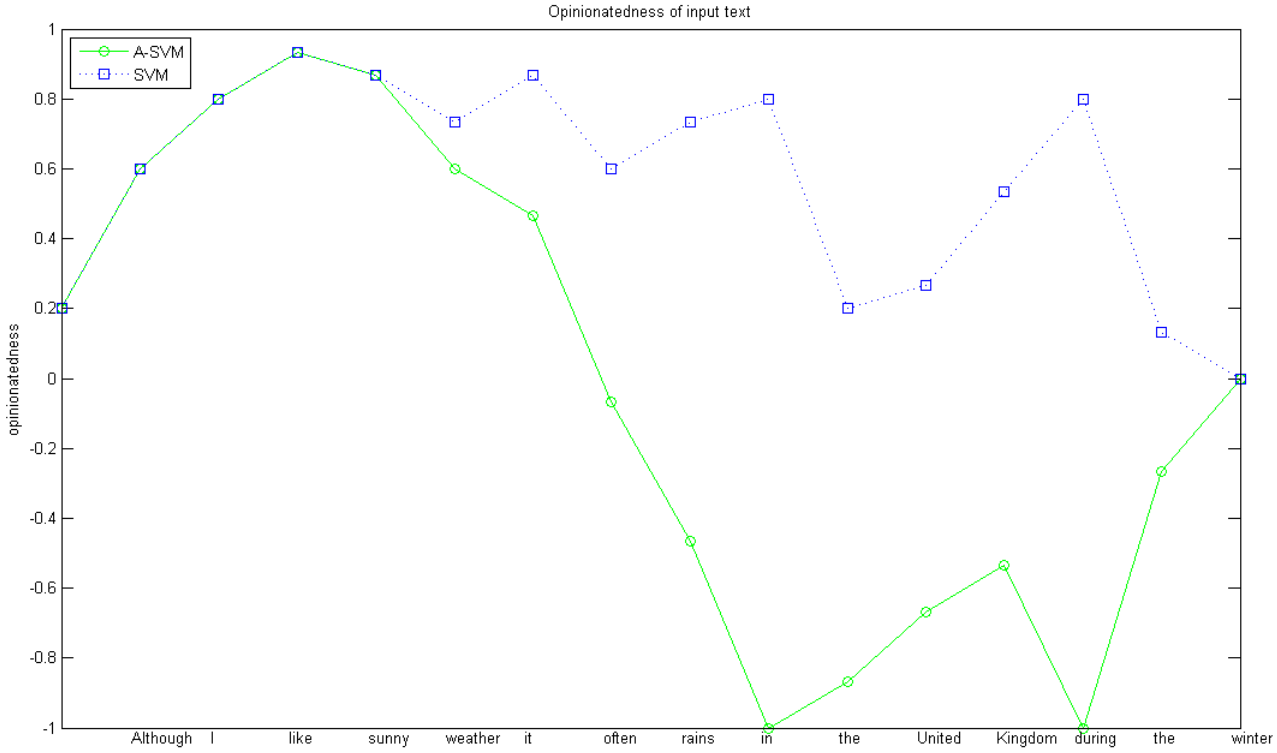

Figure 6.4: Plot showing classification results of an example user input for both SVM and A-SVM

with the SVM classification and consequently does not alter it while differences throughout the remainder of the sentence reflect the disagreements between user feedback and SVM classification. Thus, the arguments that the user provided correct the, in this case faulty or ambiguous, SVM classification of the second half of the sentence.

When presenting the results to the user, only the A-SVM classification would be given in the output since the SVM classification is not of interest to the user. Accumulating a score in this manner for each word and plotting them in a similar manner to what is shown in figure 6.4 would give the user a simple to understand representation of an abstraction of the classification results without demanding any deeper understandings of the system's works.

6.2 User feedback

As elucidated above, one of the central concepts of this project has been to incorporate arguments into the process of text classification in order to enhance the performance of a straight forward SVM classifier. The very basic question concerning this concept was how to acquire such arguments in an efficient way, i.e. with low effort, without compromising the quality of the arguments acquired. In order to accomplish this, I have developed a concept of user feedback which allows an individual to construct a small numbers of arguments from which information is extracted that aides in classifying not just those n-grams reasoned upon, but also those sharing features with these arguments. This section describes the mechanism I have put in place to maximise the utility of a limited amount of information in the form of arguments. It is important to emphasise that this limitation is of purpose since it is, in most cases, not feasible to have a human user, i.e. annotator, construct a large number of arguments or annotate a large amount of text manually. It is thus vital to find ways of maximising the value of information at hand while at the same time minimising the effort that is required of human annotators.

Gathering information through user feedback as is done here offers a compromise between clas-

sification that fully disregards human intervention in the classification process and straight forward manual classification of contents. Automatic classification rids us of the need for time consuming annotation processes, but it may, depending on the domain, be less reliable than manual annotation. This has, up to this date, generally been the case in Sentiment Analysis. Considering that entirely manual classification of contents, on the other hand, defeats the purpose of a system such as the one presented here, I have proposed collecting user feedback that only demands a few seconds to produce as a viable alternative.

6.2.1 Generating arguments

As shown in figure 6.3, the arguments that contribute to the final classification of all n-grams extracted from the user input are generated by the user. Recall that the arguments acquired take one of two possible forms,

C because *Reason*

or

C despite *Reason*

where $C \in \{1, -1\}$ and the *Reason*, in this case is a succession of words appearing in the n-gram that is judged. The user selects three values:

1. The class label, $C \in \{1, -1\}$, or *opinionated* versus *non opinionated*
2. the direction of reasoning, i.e. *because* or *despite*
3. The part of the n-gram that is *most* responsible for the user's judgement

One resulting argument, reasoning about the n-gram we have used before, *"...has refused to bow..."*, may look as such:

"Opinionated" "because" "refused to bow"

In this hypothetical scenario the user has judged the n-gram to be opinionated and identifies words two to four as the reason for his or her judgement. In the same manner, the user constructs four additional arguments by passing judgement upon four other randomly selected n-grams and providing reasons to support the decision. These judgements, together with the attached reasoning, are then passed to the system and from them, two separate scores are calculated that change the SVM classification results to what is the final classification. How these two scores are extracted from the arguments and how and when the SVM classification result is changed according to these scores is explained in the following section.

6.2.2 Direct and indirect user feedback

In order to maximise the gain of information taken from the user's input, the arguments supplied by the user are processed to yield twofold information about the user input, namely direct feedback and indirect feedback. Direct feedback works in a rather straightforward manner while indirect feedback is gained through a more intricate mechanism.

Direct feedback is constructed, as the name suggests, directly from the user feedback. The user classifies five n-grams as either opinionated or non opinionated and adds reasoning to those classifications. Disregarding the reasons leaves a classification of five n-grams which overrules the classification of the SVM for those particular n-grams. This overruling is done based on the choice to trust the manual classification over the SVM classification. Whenever the manual classification agrees with the SVM classification, the class label remains the same. When the user classifies an

n-gram differently, the class label of this particular gets changed according to the user's judgement. Accordingly, at most five class labels from the original SVM classification results get changed either from -1 to 1 or from 1 to -1.

The indirect feedback utilises all three sources of information to obtain scores for all n-grams that were originally obtained from the user input and is thus more far reaching than the direct user feedback. Initially, each n-gram is assigned a score of zero and whenever one of the arguments given by the user holds some information about one or more of the n-grams, their scores are changed according to the content of this information.

Before constructing this score, though, some preprocessing has to be foregone with the arguments. The first step in this preprocessing checks the direction of the justification, i.e. whether the user classified the argument the way he or she did *because* of the reasons given or *despite* the reasons given. Consider again the following example, an argument on the n-gram *has refused to bow*:

"Opinionated" "because" "refused to bow"

A different judgement of another user may be as such:

"Non opinionated" "despite" "refused to bow"

Both these examples equally indicate that the succession of words *refused to bow* is an indicator for the n-gram being opinionated. What is thus extracted from these two arguments is equivalent, i.e. in case of the second argument we switch the class label from *non opinionated* to *opinionated* and store the n-gram *refused to bow* associated with this label. In a second step of processing the words are passed to the TreeTagger to extract their types, as was described in chapter 5. Replacing the words themselves with their types when deducing the indirect feedback allows greater generalisation when amassing indicators for text being *opinionated* or *non opinionated*. Having the types as indicators for a class means that is suffices if a an n-gram in the text contains the same combination of type labels as the n-gram that was used to construct the argument, the n-gram doesn't have to be exactly the same. Once this preprocessing is completed the arguments given above would be represented in the following form:

1 VVN TO VV

What we are left with is the class label of the partial n-gram that was identified by the user to be responsible for his or her chosen classification, as well as the types of the words contained in the partial n-grams. The score is now accumulated in the following manner: Whenever one of the n-grams extracted from the original user input contains a combination of types equivalent to one of the combination of types determined through the arguments the score is either increased by one or decreased by one. It is increased when the argument determined this combination of types to be an indication for the n-gram being *opinionated* and it is decreased when it indicates that the n-gram is *non opinionated*. Any n-grams whose score surpasses a threshold after all arguments have been processed adapt the class label suggested from the scoring system, replacing the original SVM classification. The remaining n-grams keep the label that was originally suggested. A score surpasses the threshold if it is either larger than one or smaller than negative one. If it is larger than one it means that at least two arguments have voted for this particular n-gram being *opinionated*, if it is smaller than negative one, at least two arguments have voted for the n-gram not being *opinionated*. If arguments supply conflicting suggestions, i.e. one argument votes for an n-gram being *opinionated* and one votes for it being *non opinionated*, these votes cancel each other out and the n-gram retains its original class label.

The final classification is thus a result of combining evidence from SVM and arguments provided by a human user where the classification results of the SVM form the basis, with the arguments overruling the SVM classification whenever the evidence supplied by them is deemed strong enough. How this mechanism of deducing insight from user feedback and combining it with the SVM results is put in place to form a functioning system is described in the following chapter.

The calculation of the final classification is summarised below in algorithm 1. Line ten checks whether the current feature vector has the equivalent word type feature values as one of the n-grams classified by the user feedback. If this is the case the class label l_m is overwritten with the user's classification (line eleven). If this is not the case, we check whether the combination of of words that the user chose as being responsible for his or her choice of classification is part of the current n-gram's features (line twelve). If this is the case, we either increase or decrease the *feedback_ val*, depending on the class label of the sub n-gram (lines 13 to 16). After all $F = \{f_0, ..., f_m\}$ have been processed, all the n-grams whose indirect feedback value v_j surpasses the threshold is changed accordingly (lines 22 to 28).

Algorithm 1 Pseudocode describing the final classification procedure

1: *let* $N = \{n_0, ..., n_4\}$ *be the* $n - grams\ for\ user\ feedback$
2: *let* $U = \{u_0, ..., u_4\}$ *be the user* $feedback\ class\ labels$
3: *let* $A = \{a_0, ..., a_4\}$ *be the* $feedback\ reasons$
4: *let* $F = \{f_0, ..., f_m\}$ *be* $feature\ vectors\ representing\ original\ user\ input's\ wordtypes$
5: *let* $L = \{l_0, ..., l_m\}$ *be the class labels determined for* $F = \{f_0, ..., f_m\}$ *by* SVM
6: *let* $V = \{v_0 = 0, ...v_m = 0\}$ *be the indirect* $feedback\ values\ for\ F = \{f_0, ..., f_m\}$
7: $counter \leftarrow 0$
8: **while** $counter < m$ **do**
9: **for** $i = 0$ to 4 **do**
10: **if** $f_{counter} == n_i$ **then**
11: $l_{counter} \leftarrow u_i$
12: **else if** $a_i \in f_{counter}$ **then**
13: **if** $l_{counter} == +1$ **then**
14: $v_{counter}++$
15: **else**
16: $v_{counter}--$
17: **end if**
18: **end if**
19: **end for**
20: $counter++$
21: **end while**
22: **for** $j = 0$ to m **do**
23: **if** $v_j \geq 2$ **then**
24: $l_j \leftarrow +1$
25: **else if** $v_j \leq -2$ **then**
26: $l_j \leftarrow -1$
27: **end if**
28: **end for**

7

System evaluation

The prime question I have tried to answer during this project has been whether adapting multimodal approaches to Sentiment Analysis may prove valuable to the cause of furthering the development of systems. To allow an answer to such a question, it is imperative to compare the system utilising different modalities of analysis with a baseline classifier of similar structure. In the case of this project, the most apt comparison has been to put the classification performance of the developed system up against the classification performance of the SVM incorporated in the system, stripped bare of everything but the SVM itself. In addition to this comparison of classification performance, a user centred evaluation was conducted which aimed at complementing the results of the quantitive evaluation with qualitative insight into the subjective impression users get from the system's performance. Such qualitative analysis has merits here because we are dealing with a topic that is inherently subjective, itself. Many times, classifying text as opinionated or non opinionated is a rather ambiguous process and depends on individual understanding of opinionatedness. It is thus sensible to let the individuals who provide the arguments during a system run also judge the classification results which depend upon these arguments.

7.1 SVM vs A-SVM

In order to evaluate the system's classification performance, the corpus of n-grams, described in chapter 5, was split into a training set and a test set. 2/3 of the corpus were used to train the SVM classifier and 1/3 of the corpus was used as a test set. This division yielded a training corpus of roughly 8,000 n-grams and a test corpus of around 4,000 n-grams. Due to the size of the data set, it was not necessary to apply evaluation techniques such as cross validation or bootstrapping, which are frequently applied when a data set is not sufficiently large to split it in the way done here. Before evaluating the classifier's performance and subsequently comparing it to the system's performance, it was necessary to obtain the optimal setting for all adjustable parameters that determine the SVM's performance. Accordingly, a gird search was executed for all parameters and the effect of changing them. The parameters and the range of values tested during this search are listed in table 7.1.

Starting from a sparse grid to narrow down the search space of tested values, changing *gamma* and the *cost* parameter turned out to yield the most significant improvement in performance. Thus, a wider range of values was tested for these values. The different kernel functions are listed in figure 3.1 where we can also see the effects of the parameters. Some have no influence on the process

Kernel type	Kernel degree	gamma	Kernel coefficient	Cost C	bias
linear	1	0.001	0	1	1
polynomial	to	to	to	to	to
RBF	10	100	1	60	10
sigmoid					

Table 7.1: parameter values tested for SVM training

Kernel type	Kernel degree	gamma	Kernel coefficient	Cost C	bias
RBF	N/A	80	N/A	1	1

Table 7.2: parameter values yielding the best classification result on the test set

	Precision	Recall	F1
SVM	78.1%	82.08%	80.04
A-SVM	83.7%	84.49%	84.09%

Table 7.3: Evaluation results of SVM and A-SVM

depending on which kernel function we choose. Using the Radial Basis Function (RBF) yielded the best results:

$$K(x, y) = \exp(-\gamma * |x - y|^2) \tag{7.1}$$

We can see that both the kernel degree and the kernel coefficient do not influence the kernel calculation using the RBF. The bias determines the offset of the separating plane from the origin and is thus relevant regardless of which kernel function we choose. Recall that we expressed the problem of SVM originally as

$$y(x) = \mathbf{w}^T \phi(x) + b. \tag{7.2}$$

In this formula, b is the bias. The values that produced the best classification performance of the SVM on the test set is shown in table 7.2. Classifying the test set with the SVM trained with these parameter settings yielded precision of 78.1%, recall of 82.08% for opinionated n-grams and an F1 value of 80.04 (see table 7.3. This constituted the baseline which the system was compared to.

In order to allow a comparison between the performance results of the SVM with the system itself, the test set had to be treated as user input to the system and analysed accordingly. It was not feasible to simply pass the full test set to the system as one text, because doing so would have rendered the system's potential benefits void. Consider the size of the test set, which consists of roughly 4,000 n-grams with an average size of three words each. This means that the test set is comprised of approximately 12,000 words. Were we to pass a text of this size to the system to analyse it, it would be reasonable to assume that no significant differences could be found between the classification resulting from the SVM classification and the classification results returned by the system. Passing all words comprising the test set to the system as one would yield roughly $(12,000 - 2) * 5 = 59,990$ n-grams, only five of which would be reasoned about by the user. We can see that a less brute way of comparison had to be employed to evaluate the system. This issue of sheer size was conquered by splitting up the test corpus and executing multiple runs, each time passing an excerpt of the text corpus to the system and storing the resulting classification. To find an appropriate word count according to which the test set could be split, 20 randomly selected news articles were extracted from web presences of two English speaking newspapers, namely the Guardian [26] and the New York Times [74]. All articles were split according to their paragraphs and the average word count of the paragraphs was calculated, which was 43.65. Using this word count as a reference for paragraph size, the test corpus was split into 200 paragraphs, representing roughly the size of 1.5 online news article paragraphs, each of which was analysed by the system on its own.

The values proposed here were chosen rather heuristically and further investigation will need to be employed to determine an optimal relation between word count and the number of arguments that are used. This is further discussed in section 8.2.5.

The result of analysing all subparagraphs that make up the test corpus when it is divided in this manner is the mentioned collection of roughly 60,000 n-grams. From these n-grams all those were extracted that actually occur in the test set. These n-grams had, at this point been fully analysed and classified by the system. It was thus possible to evaluate the classification performance of the

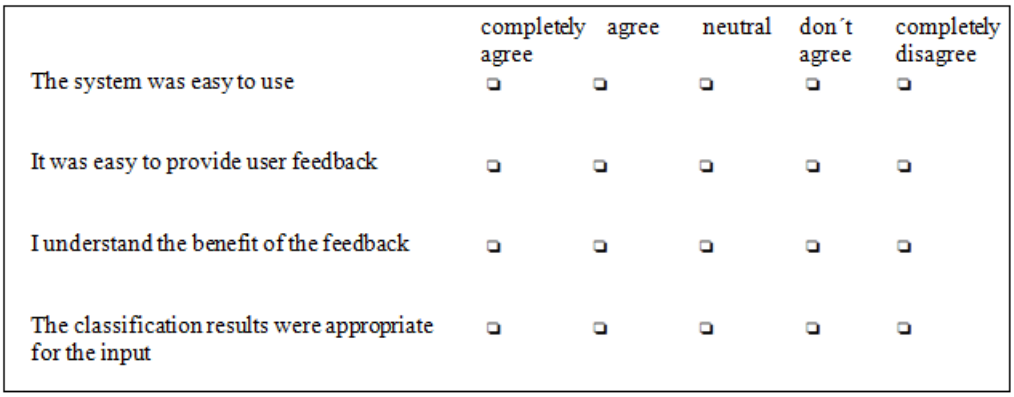

Figure 7.1: Excerpt from the questionnaire filled out by the evaluation participants

system from this result equivalently to the SVM classification results. The classification results using the system as described above yielded precision of 83.7%, recall of 84.49% for opinionated n-grams and an F1 value of 84.09 (see table 7.3). Using the system to classify paragraphs thus yields a performance increase of 5.6% in precision, 2.41% in recall and 4.05 points for the F1 measure.

7.2 User centred evaluation

The previous section presented a quantitive evaluation of the system by comparing its classification performance with that of an SVM on its own. This meant attaining straight forward and easy to interpret performance measures upon which a judgement of the system's quality could be based. This provides valuable insight, but needs further support from a qualitative evaluation for two reasons: Both the system and the SVM evaluation rely in their workings on qualitative judgements that have either been passed during the development of the corpus or during the classification process itself. In addition to this issue, analysing text with regards to its sentiment often involves ambiguities that are owed not just to the context words and phrases are set in, but also the context a pieces of text may be written or read in or who it is written or read by. Consider again, the same comment made by Tony Blair that we have used before:

> After I'd left, the agenda lost momentum. But the papers and the work are all there.

A person reading this sentence may just take it as a piece of factual information while the next person reads a sarcastic remark showing the author's animosity towards the issue. Equivalently, while uttered by Tony Blair in this context the remark was intended as a snide comment on the current government's policies, another author may have meant to provide helpful information about paperwork that is already in place. Additionally, one of the sentences preceding or following these two may put them into clearer context. It is clear that we have to pay respect to numerous uncertainties whenever we are trying to disseminate text according to its opinionatedness.

For this reason, complementing a quantitative evaluation such as the one described in the previous section with a qualitative evaluation allows us to gain a clearer understanding of whether or not the system is performing in the way we wish it to. By asking a user to work with the system and judge it afterwards, we can tackle both problems described above. The qualitative judgements, the user feedback, that influence the final classification are provided by the same entity that evaluates the evaluation result. This means that at least part of the qualitative input during the system execution is measured by the same standard as the evaluation of the results. A problem that remains and which will need further investigation (see section 8.2.5) is the fact that the annotations from which the classifications of the the text corpus contents were devised have been made based

	mean	std
Run 1		
The system was easy to use	4.667	0.516
It was easy to provide user feedback	4	0.655
I understand the benefit of the feedback	4.067	0.961
The classification results were appropriate for the input	3.867	0.561
The representation of...the results...was understandable	3.6	0.828
Automatic detection of opinions is useful	4.333	0.617
Run 2		
The classification results were appropriate for the input	4.067	0.594
Providing feedback was cumbersome	1.8	0.561

Table 7.4: Evaluation results of all eight agreement questions asked in the questionnaire

on a different person's understanding of opinionatedness. Optimally, though, the large size of the corpus will diminish any potential issues arising from this. Furthermore, the judgement about the system's performance that a user evaluating it passes is based on his or her own understanding of how the piece of text that is analysed should be classified. The user has its own idea of the context the paragraph he or she passes to the system should appear in and what the context would entail with regards to the classification. The same holds for language artefacts such as sarcasm and irony. Accordingly, the user will judge the system's classification by these standards and only when it conforms to them will the user evaluate the system favourably.

Thus, the qualitative evaluation conducted and described below offers additional insight and provides, together with the quantitative evaluation, a clearer picture of the system's actual quality.

The user was asked to use the system twice, once analysing a piece of text that was provided and once choosing his or her own text to be analysed. Having multiple users judge the same piece of text meant attaining easily comparable results while having the users choose their own text allowed an analysis of the system's robustness to unexpected input. After each of the two system executions the user was asked to judge a number of statements on a fivefold scale indicating how much the user agreed with the statement made. Figure 7.1 shows an excerpt from the questionnaire, the full questionnaire can be found in appendix A.

In addition to passing judgement on statements such as those shown in figure 7.1, the user was also asked to store the general system's quality on a scale from one to five, without giving detailed instructions upon which this judgement should be based. This was asked of the user to complement the more specific questions with a broader evaluation of the user's confidence in the system's performance and output. The average score given to the system by this measure lay at 3.8 ($std = 0.414$) for the classification of the text that was given and at 3.667 ($std = 0.408$) for the user's own input. The results of the remaining questions are shown in table 7.4, with a value of five representing full agreement and a value of one representing full disagreement with the statement. The questions measuring user agreement are designed to evaluate both the user's judgement of the classification results and his or her opinion on the value of Sentiment Analysis, in general. Out of the six questions asked after *Run 1*, only question four is asked again after the second run, since the agreement with the other statements should not change from one run to the next. The second question asked after *Run 2* asks the as question two after *Run 1* from a different perspective. This is done to check the validity of the user's answers and have a measure to check the answers' consistency. Generally, the participants of the evaluation showed agreement with the statements attributing positive characteristics to the system and disagreement with the one statement with an opposite sentiment. The lowest agreement was shown with the statement evaluating the intuitive understandability of the classification outcome's representation. This was to be expected since the graphical representation shown to the user was a very rudimentary one with much room for improvement, as discussed in section 8.2.4. Both runs that each user executed yielded similar scores of classification quality, suggesting that the system should be relatively robust to varying types and genres of input.

8

Conclusion and outlook

Sentiment Analysis is a complex field, many of whose problems still offer many opportunities to further improve upon, and the same holds for this project. By introducing a novel approach to Sentiment Analysis I have aimed at providing insight into the potential benefits of tackling Sentiment Analysis in a way that goes beyond pattern recognition. Doing so has had a twofold effect: The results have shown that following this path of bringing together Machine Learning, NLP and Argumentation may prove to benefit future developments in the area. At the same time, new unknowns have been introduced which, in addition to the issues Sentiment Analysis was faced with before, will need investigation when progressing in the field along the lines of multimodal solutions. This chapter critically reflects on the outcome of this project, i.e. the developed system and the evaluation results, and then goes on to present an overview of future challenges that will need to be addressed when moving forward on both the approach I have taken during this project and Sentiment Analysis, in general.

8.1 A critical reflection on A-SVM

In order to put the following discussion of the developed system into perspective, I will briefly review what I have accomplished during the project. From a high level perspective, the project has been concerned with three interrelated tasks:

1. Constructing a text corpus
2. Developing a Sentiment Analysis system that determines text's opinionatedness
3. Evaluating the system's performance based on the text corpus

The corpus was constructed from three different sources, the MPQA corpus [82], the TreeTagger system [70] and SentiWordNet [20]. The result from extracting n-grams from the MPQA corpus and annotating those n-grams with a number of features using TreeTagger and SentiWordNet was a collection of roughly 13,000 feature vectors describing the same number of n-grams.

A system was then developed that classifies user input in two passes. The first pass yields a classification of all n-grams contained in the user input up to a length of five words. This classification is accomplished using an SVM classifier trained on the developed text corpus. The second pass of classification joins together these results with user feedback in the shape of arguments. The final results are presented to the user via simple plotting facilities. Comparing the system's classification performance with the classification performance of just the SVM used within the system yielded an increase in classification precision of 5.6% and in two different runs, users gave the system's classification performance an average score of 3.8 and 3.667 on a scale from one to five.

The results of the evaluation show that both an increase in classification performance and a fairly high degree of user confidence in the system have been achieved. This allows the conclusion that investing the additional resources in development and computation time during the system's

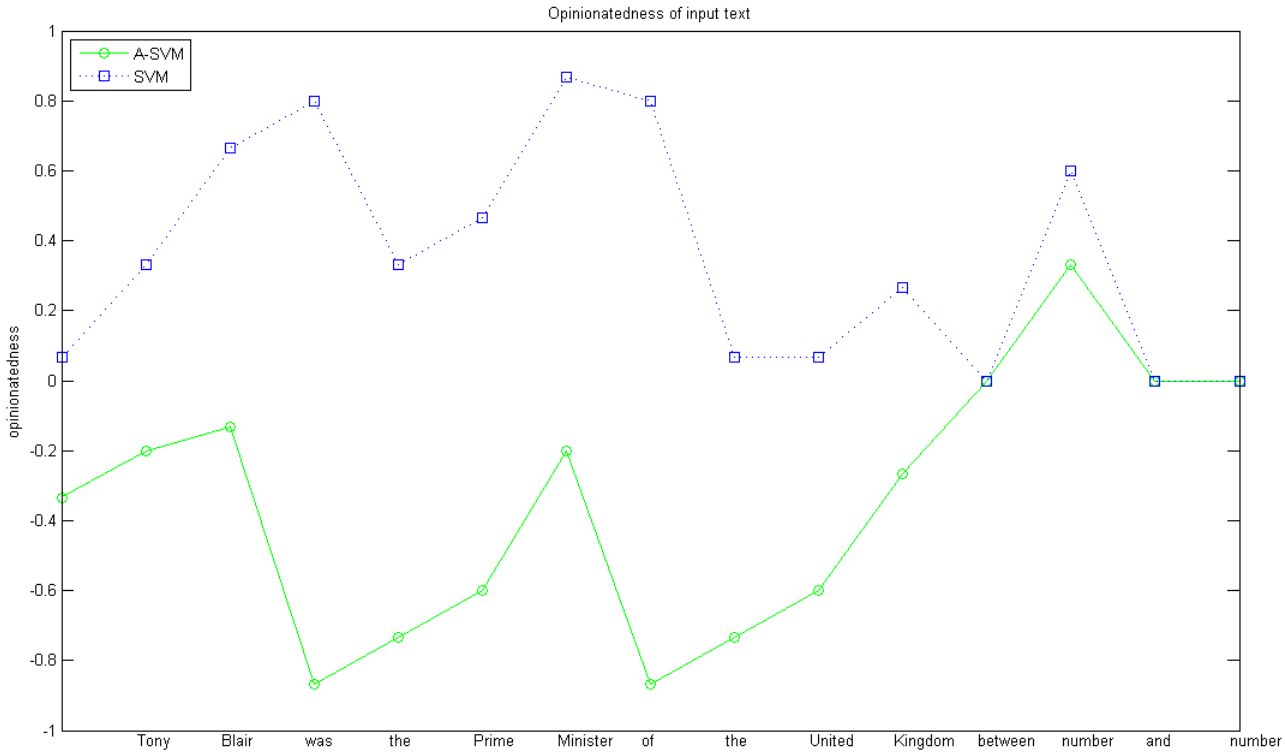

Figure 8.1: Extreme case of low agreement between SVM and A-SVM, hinting towards a bias of the SVM to classify text constituents as opinionated. The word *number* is a place holder replacing numbers in user input for the original input, here *1997* and *2007*

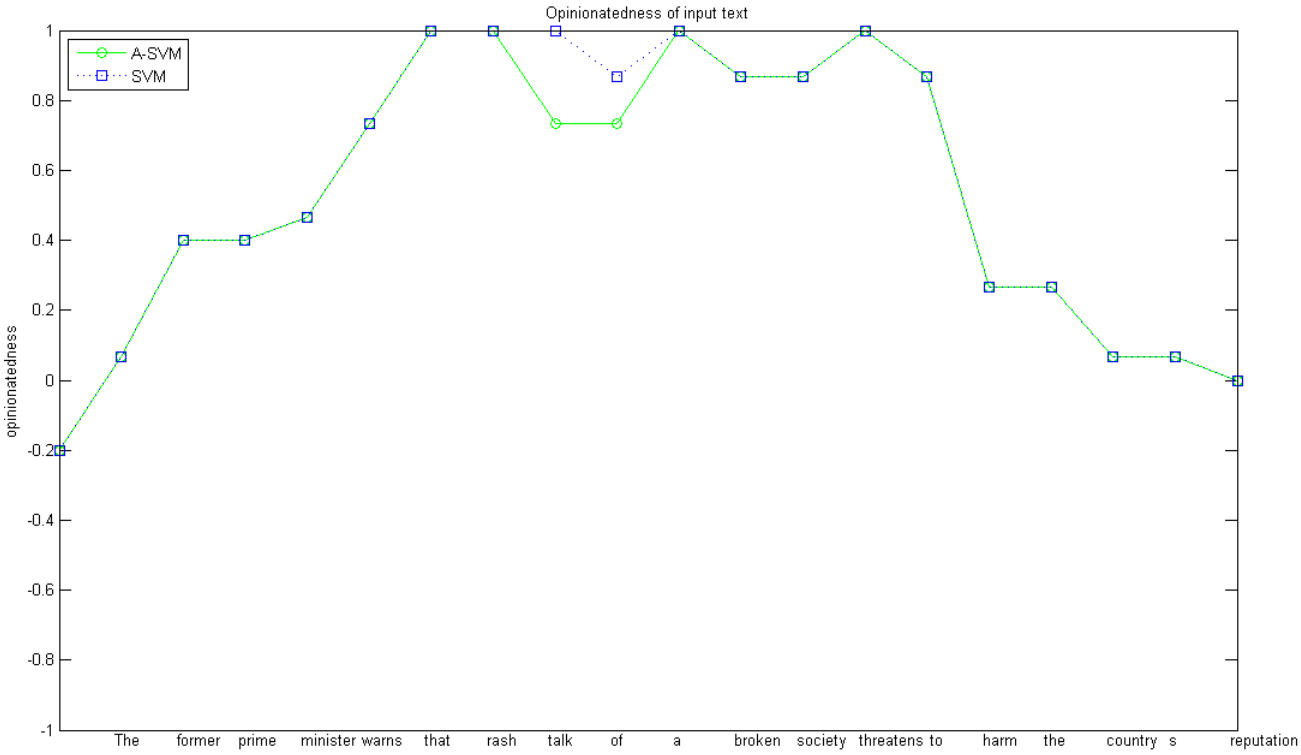

Figure 8.2: Extreme case of high agreement between SVM and A-SVM

execution yield an improvement adequately compensating the added effort. As has been suggested by results of other works in Sentiment Analysis, e.g. [61], using SVM as the Machine Learning algorithm of choice seems to hold promise when developing Sentiment Analysis tools. Nevertheless it is arguable that classification precision of 78.1% with the SVM may be improved upon by pouring effort into developing algorithms tailored strictly to the purpose. By achieving a higher classification performance through an improved classifier it may also be hoped to increase the total classification precision after adding user feedback. Alternative formats for the feedback itself may prove to increase the performance, as well, though this was not investigated during the project. Issues such as this may be counted among those that pose additional questions to those that already stand unanswered in Sentiment Analysis. In the following sections I discuss both issues that arise from incorporating arguments to the process of sentiment classification and those issues that are vital to Sentiment Analysis but that have not been tackled during the project. I thus provide insight on both general issues that need close consideration in Sentiment Analysis and unaddressed challenges characteristic to this project.

One further aspect of the classification results shall be mentioned, here. The SVM classification seems to exhibit a bias towards classifying n-grams as opinionated. Evidence for this bias being present can be gathered from comparing classification outcomes of the SVM with A-SVM. It seems that currently the majority of corrections that take place through the user feedback occur on n-grams that have falsely been classified as opinionated. Assuming such a bias is actually present, a restructuring of the training corpus would be the most urgent prerequisite for further progressing on the system's performance. Figure 8.1 and 8.2 show example classifications of two extreme cases, one opinionated sentence for which the two classification results agree almost completely and one non opinionated sentence showing very little agreement in the classification results. For each sentence, both the SVM classification results and the A-SVM classification results are shown. We can see how, for the opinionated sentence, the classification results differ only slightly from each other, whereas the classification results for the non opinionated sentence exhibit a clear correction through the user feedback.

As further explained in section 8.2.2, such issues arising from a lack of robustness of the training data may be conquered by introducing additional features that assist the training and classification process.

8.2 Future challenges

The following account of challenges that are posed by Sentiment Analysis but have not been addressed during the project or demand improvement is subdivided according to the nature of the problems that arise when dealing with these issues. Though I aim at providing information on the most pressing issues and those closely associated with what I have worked on during the project, the list of challenges is by no means all encompassing. This goes to show how much work still lies ahead for researchers and developers working on Sentiment Analysis solutions.

8.2.1 On the polarity of opinions

In chapter 2 I have identified four central challenges of which Sentiment Analysis is comprised:

1. Gathering text
2. Discriminating opinionated text from non opinionated text
3. Determining the polarity of opinionated text
4. Summarising the results of analysing text's sentiment

Throughout this project I have focused on gathering text and discriminating opinionated text from non opinionated text, also touching upon the issue of summarisation. Determining the polarity

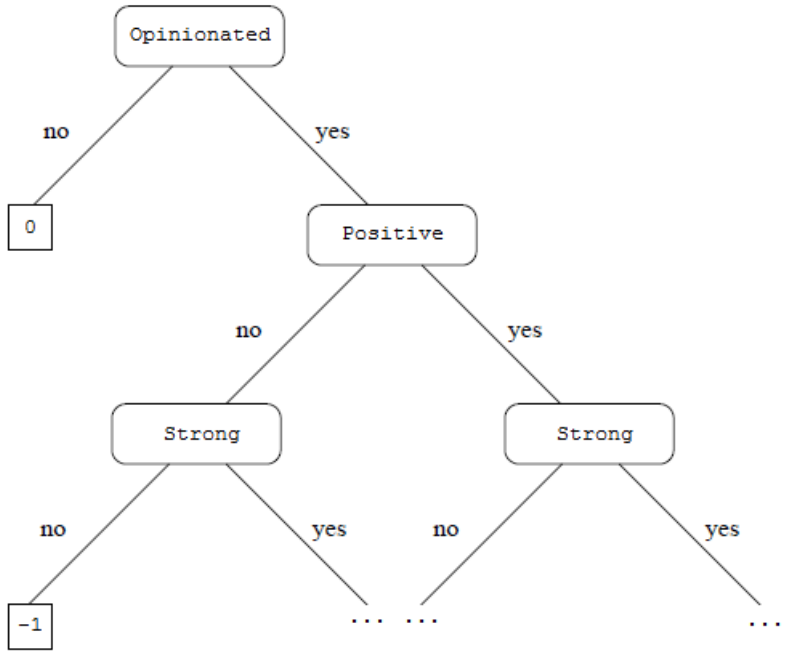

Figure 8.3: Decision tree representation of using multiple binary classifications to determine fine grained sentiment polarity

of opinionated text has not been of concern. This does not go to say that this issue is any less vital to Sentiment Analysis than the others that were investigated, but is rather a reflection of limitations in time and resources that had to be accounted for. This is equally valid for all challenges explained subsequently.

Numerous approaches have been presented to address the issue of determining sentiment polarity, e.g. [8], [61], [76], [85]. Some of these efforts have focused on sentiment polarity exclusively, while others integrated their propositions within solutions that deal with the entire spectrum from determining opinionatedness up to the summarisation of classification results. Generally, we can say that a system that aims at performing Sentiment Analysis in a way that is valuable to a human user in a day to day usage will need to successfully incorporate all three stages of the analysis process.

In light of this project, there are two basic ways how polarity determination may be integrated into the system:

1. Add additional passes of binary decisions to the classification process
2. Develop a classifier that is able to make decisions on multiple classes at once

While the first alternative may prove to be a simpler solution than the second, it may bring with it excessive computational demands and thus prove to be an unsustainable quick fix to the issue. Let us nevertheless consider this approach first, as it will provide some intuition for the second, more sophisticated approach. Following this approach encompasses an additional binary decision for each subdivision that is implemented. Thus, distinguishing among the content that was classified as opinionated which parts are negative and which are positive would involve one additional binary decision. Were we to further subdivide these opinions into categories of high intensities and low intensities, two additional stages of classification would be needed, one for negative content and one for positive content. Such subdivisions can be viewed as a decision tree, as can be seen in figure 8.3 where we find the classification result at the leaf nodes. Such a method could be further subdivided into more binary decisions, adding more levels to the decision tree and resulting in a finer grained sentiment classification.

To incorporate such additional binary decisions into the system, two major adjustments would have to be made, one to the text corpus and one to the system itself. Firstly, the text corpus would

have to be further annotated, as for each additional classification another class label would have to be attached to each of the feature vectors that make up the corpus. Recall the sample n-gram used in earlier sections, *"has refused to bow"*, which would be represented in the current corpus as

1 1:0.4 2:0.0 3:0.78125 4:0.21875 5:0.9524 6:0.8372 7:0.5581 8:1

or, in a non numerical representation

1 0.0 0.78125 0.21875 has<VHZ>(have) refused<VVN>(refuse) to<TO>(to) bow<VV>(bow)

The feature vector has attached to it a single class label, which tells us that the particular n-gram, which the feature vector represents, is opinionated. No information is conveyed as to what the polarity of the n-gram is. This information would have to be added to the corpus, since it is what the classifier is trained on. Assuming that the example n-gram conveys a negative sentiment, annotating the feature vector may yield the following:

1 **-1** 1:0.4 2:0.0 3:0.78125 4:0.21875 5:0.9524 6:0.8372 7:0.5581 8:1

An additional class label was attached to the feature vector, with $C \in \{1, -1\}$ and -1 representing negative polarity and 1 signifying positive polarity. Annotating the corpus in such a manner would demand twofold training of the SVM, as it presents one binary decision at a time. Accordingly, we would either have to adjust the format in which the SVM takes input or create separate corpora, one classifying each feature vector according to its opinionatedness and one subcorpus containing just the opinionated feature vectors. Such a subdivision would have to be extended further accordingly for each additional level of distinguishing the opinionated contents more finely grained. In addition to adjusting the content of the corpus it would be necessary to apply some changes to the system itself. The conversion of the user input to a numerical representation of the n-grams that make up the input would have to be altered to cater to the changed syntax of the corpus upon whose basis the input is classified. Subsequently, an extra pass of user input classification would have to be implemented for each additional subdivision of the opinionated parts of the user input. The user feedback would have to be extended to allow the user not just to judge upon an n-gram being opinionated or not, but rather whether it is non opinionated, positive, negative, positive with a high intensity, and so on. Lastly, the summarisation of the results as it is currently presented to the user would have to be tailored to include the additional information in a manner that retains the intuitive comprehensibility that the summarisation should provide. This could mean, for example, presenting more than one plot or replacing the plot currently output with one showing not the probability of the text being opinionated but rather plotting the development of opinion polarity and intensity.

The second approach to integrating opinion polarity into the system's concerns would require more fundamental changes to the system architecture but may prove its benefits with regards to computational efficiency. This approach would mean replacing the classifier entirely with a multi class classifier. Though SVM have been extended beyond binary classifications, their prime purpose and pedigree lies with such binary problems. Accordingly, choosing to implement a single multi class classifier may demand the application of a different algorithm, entirely. The upshot of this approach lies in avoiding multiple passes through the data, which is especially costly when the data sets are large. Since large data sets are often vital in text analysis, utilising the advantage of fully classifying the data in one pass may be crucial. As with the other solution proposed, both the corpus and the system would have to be tailored to the new demands.

The class labels attached to each of the feature vectors within the corpus would have to be changed from $C \in \{1, -1\}$ for example to $C \in \{1, 0, -1\}$, where 1 signifies positivity, 0 represents neutrality and -1 stands for negativity. Reflecting this adjustment onto the example, the class label would be changed to yield the following:

-1 1:0.4 2:0.0 3:0.78125 4:0.21875 5:0.9524 6:0.8372 7:0.5581 8:1

libSVM ([7]) offers facilities to extend SVM beyond binary classification, but applying these may not be optimal due to two reasons. Firstly, the approach that is used when using SVM as a multi class classifier is more or less to letting the classifier make multiple binary decisions, which entails the same potentially negative effects as the approach described above. libSVM uses the so called one versus all approach, for which the data set is divided into two parts once for each class that can be found in the training set. Considering the current scenario, in a first step the classifier would divide the training set into a set that contains all feature vectors that are annotated as representing a positive n-gram and a set that contains all other feature vectors. After training according to this subdivision, the classifier would then split the data into one set containing all neutrally annotated feature vectors and a set containing all other data. This process is repeated as many times as we have different classes. We can see that the training effort is virtually the same as it is with multiple binary classifiers. A second reason for being sceptical about extending SVM beyond binary classification lies in the fact that, as is pointed out in [30], how to effectively extend it for multi class classification is still an ongoing research issue. Other algorithms, such as feed-forward neural networks, have been studied more extensively for multi class problems and may thus, at the current state of the art, provide better choices. The changes that would have to be made to the collection of the user feedback would be similar to what was described above when applying different algorithms.

8.2.2 On the corpus and Natural Language Processing

Section 2.2 discusses a number of NLP aspects that need to be taken into consideration when developing a Sentiment Analysis program, or, as a matter of fact, any sort of NLP system. These aspects entail system design choices at different stages of the development process. The choices made during this project and what the potential benefit of exploring alternative routes may be is discussed below.

At an early stage of annotation, all n-grams in the corpus and all n-grams of system input are converted to their basic form and are subsequently processed in this manner. Underlying this is the simplifying assumptions that tense, case etc. do not carry significant information about the word and can thus be disposed of. There might be reason to doubt that this assumption holds in all cases and accordingly there is merit in investigating potential differences between analysing only the basic forms of all words the system deals with and leaving text in its original state and considering all possible declinations, conjugations etc. of all words. There is an an obvious trade off between the need for significantly larger corpora and lexica when analysing all words in their original form and potentially loosing valuable information when stemming all words before analysing them. In order to determine the value of paying the price of enlarged corpora and lexica to retain as much information as possible, comparative analyses would have to be brought under way that concern themselves not only with differences in classification performance, but also pay respect to potential differences in domains and whether in some applications it might be worth putting in the effort while in others it may not.

Tokenisation is another issue that needs to be addressed when analysing text. It is questionable whether choosing to consider n-grams ranging from a size of a single word to a size of five words is always an optimal choice of parameters. Additionally, it might be worth investigating whether a division of text into overlapping sets of n-grams is the optimal choice, in the first place. Depending also on the application and what the aimed for nature of the classification results is, different types of tokenisation may prove either computationally more efficient or yield better classification results. When the aim of a system lies in classifying text at document level, i.e. take a less fine grained approach to identifying opinionatedness, it may suffice to analyse larger excerpts of the text at hand and thus save computational costs. On the other hand, to gain more reliable results and put higher emphasis on respecting the context, a larger number of n-grams may have to be extracted and analysed.

Each n-gram contained in the corpus developed during this project is represented by a number of features, namely the size of n-gram, three SWN scores and all word types occurring in an

n-gram. Various existing NLP techniques offer potential to deduce numerous additional features and determine those for each n-gram in the corpus. Intuitively, generating more features should enable a more accurate classification of new text, as we gather more information about it according to which we can discriminate different types of n-grams. Other POS tagging systems such as the TreeTagger are available (e.g. [4], [13]) and with them most syntactical features that words or phrases exhibit can be determined automatically. Characteristics such as the tense of phrases or their voice, i.e. active or passive, can all be determined more or less reliably by POS tagging systems.

As the application of SentiWordNet shows, it is not just POS tagging systems that can supply valuable information about text that may be translated into features describing it. There is, for example, a number of systems and algorithms other than SentiWordNet available which supply different kinds of scoring facilities for words and phrases according to different measures. Some of these have been used in Sentiment Analysis systems before and have contributed to developing corpora in a similar manner as I have done within this project. In [50], Mullen and Collier train SVM on data that is collected using a number of scoring methods for words and phrases. Their method is based on the same intuition as the development process of the corpus in this project has been. They use a number of sources that give different measures of words' characteristics to build feature vectors describing words and phrases. These feature vectors are then used to train an SVM classifier which in turn classifies text according to its sentiment. Among a number of other features, Mullen and Collier use the following two measures to construct their feature vectors, both of which could be incorporated into the project's corpus, as is explained below.

1. Semantic orientation (SO) with pointwise mutual information [51]
2. Osgood semantic orientation with WordNet [32]

SO "refers to a real measure of the positive or negative sentiment expressed by a word or phrase. (...) The SO of a phrase is determined based upon the phrase's *pointwise mutual information* (PMI) with the words 'excellent' and 'poor'. " ([50]) PMI is defined as

$$PMI(w_1, w_2) = log_2\left(\frac{p(w_1 \& w_2)}{p(w_1)p(w_2)}\right) \tag{8.1}$$

where $p(w_1 \& w_2)$ is the probability that w_1 and w_2 co-occur. A phrase's SO is deduced from the PMI by calculating the difference between its "PMI with the word *excellent* and its PMI with the word *poor*. The probabilities are estimated by querying the AltaVista Advanced Search engine for counts. The search engine's 'NEAR' operator, representing occurrences of the two queried words within ten words of each other in a text, is used to define co-occurrence." ([50]) Accordingly, we calculate the SO as follows:

$$SO(phrase) = log_2\left(\frac{hits(phrase\ NEAR\ "excellent")hits("poor")}{hits(phrase\ NEAR\ "poor")hits("excellent")}\right) \tag{8.2}$$

Thus, we obtain values greater than zero for phrases with greater PMI with *excellent* while values below zero are the result of a greater PMI with *poor*. A value of zero indicates a neutral semantic orientation.

The second measure that Mullen and Collier use to build their feature vector is the result of a method developed by Kamps and Marx ([32]) which uses WordNet relationships to deduce three values describing the sentiment expressed by an adjective. As can be seen, strong parallels to what SentiWordNet offers are found, here. The two main differences between SentiWordNet and this method lie in the fact that only adjectives are annotated with these three scores and also that the values ascertained carry different meanings in both methods. The values determined by this method identify the adjective's *potency*, i.e. strong or weak, its *activity*, i.e. active or passive, and its *evaluation*, i.e. good or bad. These measures were introduced by Charles Osgood in his Theory of Semantic Differentiation ([54]), hence the name. All three values are determined by measuring

relative minimal path length (MPL) in WordNet between the word in question and certain other words known to be characteristic for one or the other value potency, activity and evaluation.

Mullen and Collier go on to elucidate upon a number of other measures that they incorporate to form the final feature vectors that they analyse. Most of these measures may be added to the corpus as it is without need of changing any of its syntactical features. For the purposes of illustration the measures described above suffice and those not further mentioned here could mostly be implemented in an analogous manner. Recalling the structure of the text corpus, as described in chapter 5 it should be clear that both measures explained above could be added to the corpus the same way that the SWN scores were added to it. Assuming that both measures would be added to each n-gram, we would extend each feature vector to contain an appropriate number of additional values, one signifying the SO of the n-gram and three values for each word, representing its potency, activity and evaluation. We would only need one SO value for the same reason we only need three SWN scores for an entire n-gram. The SO value is a real value that we could accumulate and normalise analogously to the SWN score calculation procedure. This is not feasible for the value obtained from *Osgood semantic orientation* method because these describe non numerical characteristics of single words, such as a word being in active voice or passive voice. Accordingly, these values would have to be added to the feature vector for each word contained in the n-gram separately.

8.2.3 On system features and performance

As with any prototype of a system, a number of features could, with varying degrees of difficulty, be implemented to enhance the system's performance. We shall consider four examples that are worth implementing and investigating:

1. Adaptable number of arguments provided by the user
2. Continuous learning after each system execution
3. Allowing the user to pass judgements upon an adjustable number of features
4. Variable degrees of trust in user feedback

The system's current functionality in providing facilities for the user to create arguments is limited to five arguments with each system execution. Accordingly, we pay no attention to how large the amount of text the user passes to the system is when collecting arguments at a later stage of the execution. This is arguably not optimal. Consider a case in which the user passes just a short phrase of maybe four words to the system. With a maximum n-gram length of five this would mean that the system produces and classifies ten n-grams. Out of those ten n-grams, five are randomly selected and presented to the user. Not only may it very well happen that the user sees the same n-gram turning up for argumentation more than once, but also does the relation between ten n-grams and five arguments not seem to stand up to the entire purpose of the system: Attaining user input at a rate that is beneficial to the outcome, but also economical, as in maintaining low effort in comparison to the benefit. The workload and the benefit do not seem to be in appropriate relation, here, as it may be argued that annotating four words entirely manual may be quicker than providing five arguments about these four words. On the other hand, consider the issue that was pointed out in section 7.1. Here we discussed that providing five arguments for a large amount of text would arguably have no significant effect on the final classification. A remedy for this issue may lie in introducing a component that dynamically adjusts the amount of arguments constructed by the user in relation to how many words the user passes to the system in the first place. We could introduce multiple word count thresholds and with each threshold that is surpassed have the user construct more arguments. Through testing, optimal thresholds could be found and implemented, maintaining a similar relation between word count and number of arguments regardless of how much text is analysed by a single system execution.

The second suggestion listed above relies on the system being used frequently. Consider a scenario in which many runs of the system are executed, each time storing the classification results of both the SVM and after the user feedback. In this case, we could amass information over time which can be utilised whenever new classifications are being made. To gain an intuition, consider again the Unsupervised Learning technique of building classifications automatically from a small number of seed words (see section 3.2). This technique starts out with a small number of manually annotated, using these to iteratively build up an ever growing collection of classified words. A similar approach could be taken to utilise user feedback in a more lasting manner than simply to enhance one run of classification as is the case at this point of the system's development. Each time the user gives feedback about a number of n-grams, the classification of the feature vectors underlying the n-grams that the user judges are adjusted accordingly. We thus gain insight from the user feedback about the sentimental nature of those feature combinations that goes beyond the SVM classification. The SVM classification strictly relies on the results of training on the feature vectors contained in the training set. These do not change and thus the gain from adjusting classification according to SVM classification results may arguably be low. This is not the case for user feedback. The user judges not the feature vectors, but rather the words from which they are constructed. This means that, as the same feature vectors are, over the course of multiple executions, reoccurring represented by different words, the user may, through his or her feedback, introduce a sensitivity in classification to subtleties in the feature combinations opinionatedness that are not picked up by the straight forward feature analysis of the SVM. The basic prerequisite for utilising user feedback to improve the classification results of not just one execution, but rather increasing it over the course of multiple executions, is that the user feedback be stored in a manner that allows easy access and incorporation of it at subsequent runs. Thus, let us assume that we have available a collection of all arguments provided by previous system executions. A number of ways can be envisaged how to put such a collection to use in order to improve classification results. One of those ways might be a simple scoring algorithm that, whenever a certain combination of features is reasoned upon by the user, this combination is either added to the collection with the appropriate class label or, if this feature combination has been encountered before, either a score for being *opinionated* or one for being *non opinionated* is increased. We could then use such a score, in addition to the current user feedback, to overrule SVM classification results whenever such a score is large enough and we believe it to be representative for all occurrences of this particular feature combination. Other solutions may include more intricate scoring techniques taking into account the SVM classification, the current user feedback and the feedback history in unison to deduce the final classification of an n-gram.

Another approach to utilising user feedback history may result in a non random selection of n-grams that are given to the user to reason about. Instead of randomly selecting one from the entire collection of n-grams that make up the user input, a preprocessing step may be introduced to rule out certain n-grams because enough information has been accumulated about this combination of features during previous runs. Such a preprocessing step may allow a more efficient use of the user feedback by avoiding gathering information about feature combinations that have been classified many times before and are left with little ambiguity.

The user currently builds arguments by passing judgement about what part of the n-gram he or she is presented with is the most discriminative in deciding upon the n-gram's opinionatedness. this means that the user implicitly judges which types of words indicate a text being either *opinionated* or *non opinionated*. It may be valuable to build arguments that provide information about features other than the word types. The main issue here would arguably be the question of how to represent feature values other than word types to the user in a manner that is intuitively understandable. For some features this would be simpler than for others. Were we, for example to incorporate features such as the n-gram's tense or voice, this could simply be presented to the user as it is and he or she could judge the feature's responsibility. The issue is not as straightforward when it comes to presenting any of the scores presented in section 5.1.2 and section 8.2.2 to the user. Allowing a user to pass judgement on numerical values such as the SWN scores would, in some cases demand more

or less lengthy explanations about their meaning before the user could provide an educated decision. Such explanations would increase the time needed by the user to construct arguments considerably, which is what we are trying to avoid in order to retain the value of the entire procedure. Thus, in order to allow the user judgement on such scores, a way would have to be found to represent those values in an intuitively understandable manner without compromising the information that the value actually holds. This may be accomplished by devising a conversion of the numerical values into a worded representation through which these measures would be represented in a syntax closer to the features already used in the current version.

The system currently incorporates direct user feedback in a straight forward manner. Whenever user and SVM classification of an n-gram disagree, the SVM classification is simply overruled to conform to the user feedback. This means that we trust the user over the SVM classification. As progress is made in the classification quality of the algorithm applied, we might want to consider a less brute way of processing disagreements in classifications between the classifier and the user feedback. One way of doing this could be through the introduction of two measures of trust, one for the classifier and one for the user. These trust measures would then be compared whenever disagreements arise and the classification result of the entity carrying the higher trust value would be the one accepted as the final classification.

Figure 8.4 recapitulates the system's architecture as it is described in chapter 6 and figure 6.1 with instances of the suggestions made above added to the procedure. Three additions are made from figure 6.1 to figure 8.4:

1. Additional features and an n-gram count are extracted from the user input to construct GUI2
2. A trust score influences the outcome of the indirect and direct feedback and is influenced by them reciprocally.
3. Together with the scoring algorithm, stored classification results, collected from previous runs, decide upon the final classification.

8.2.4 On summarisation

When considering summarisation in Sentiment Analysis, we generally concern ourselves with merging the original data we have analysed with the data that is the outcome of the analysis. The aim is to produce an output and present it to the user that allows him or her to digest the opinionated contents and the information that goes with it more easily than it would have been possible from the original input. As explained in section 2.3.4 this may be done either in a textual or a graphical manner.

During the project I have only briefly addressed the issues that come with appropriately summarising the classification results provided by the system. This means that we are left with considerable room for improvement and some aspects shall be pointed out, here. Recall that the graphical output that serves as a summary of the system's classification results is a simple plot showing the system's confidence in single words being opinionated or non opinionated. The most fundamental issue that this choice of representation prompts is the fact that it does not show the actual classification results, but rather a simplification of them. The classification of the text that is supplied by the user concerns itself with analysing n-grams of varying sizes, not single words. The classification of n-grams which the system returns is abstracted to a single word classification in a rather rudimentary manner. By accumulating the scores of all n-grams in which a word appears to yield a single word score for opinionatedness it is arguable whether or not the amount of insight provided by the n-gram classifications is retained. A second reason that emphasises the worth of investigating alternative summarisation techniques is the fact that not only do we represent the classification results by means of a gross simplification, but we also leave out information that might be valuable

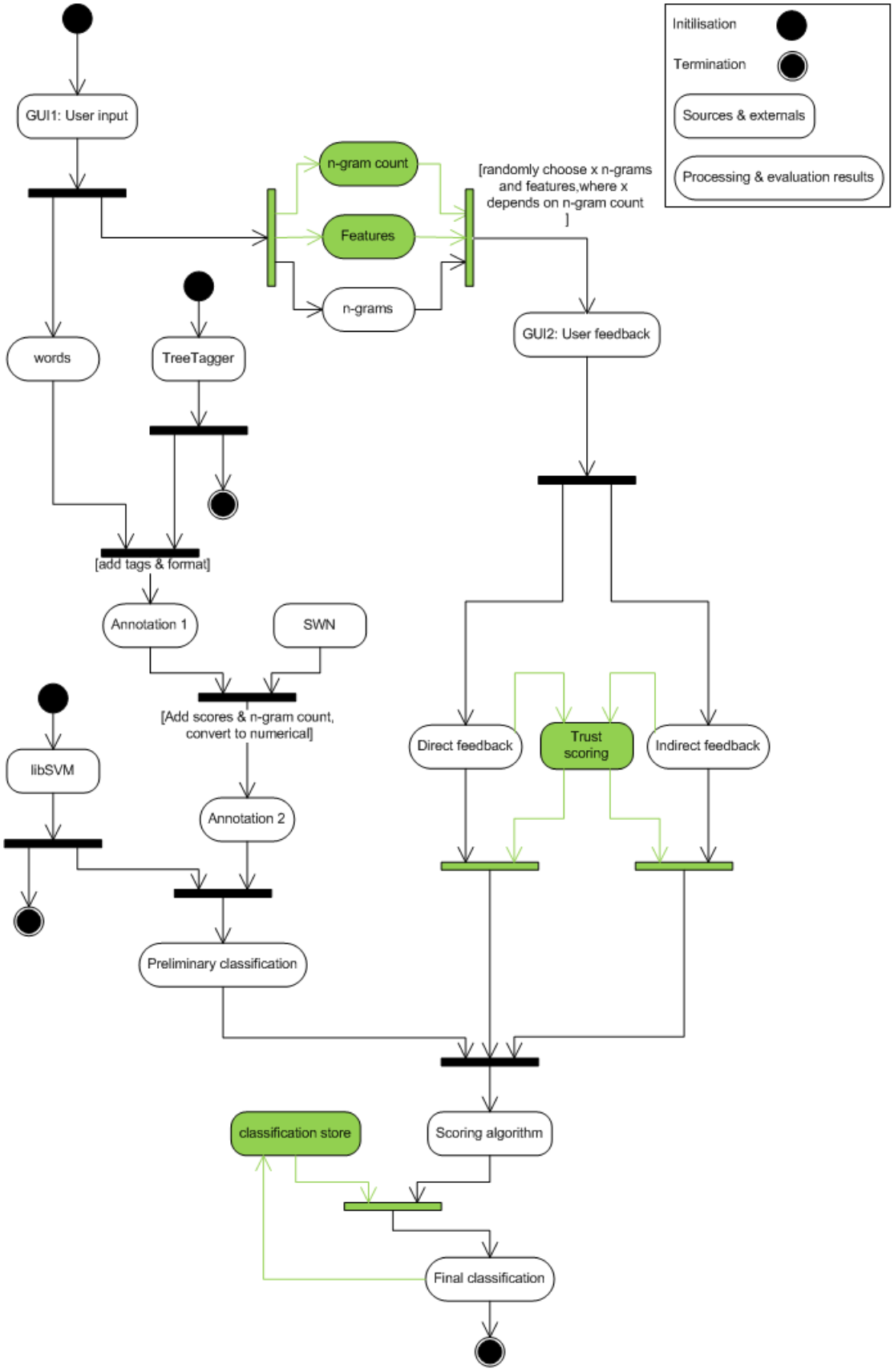

Figure 8.4: Schematic view of the system with additional features as described in section 8.2.3. The added features are highlighted in green.

to the user entirely. For user input that consists of n words, we attain two sets of classifications, each comprised of $m = (n-2) * 5$ n-grams in the case of a maximum n-gram size of five. Both sets are available in two different representations, one comprised of n-grams with class labels and one made up of feature vectors with class labels. In addition to that, we obtain five arguments attached to five randomly selected n-grams. We can see that we obtain a wealth of information about the user input that is either left out of the representation of only partially represented.

Below we shall consider advances in summarisation with regards to the project in two stages. Firstly, we shall investigate some approaches to incorporating more information into summarisation to achieve a richer summary while maintaining an intuitive comprehensibility of the summary. Secondly, some implications of further developing the system itself to the issue of summarisation are pointed out. For example, what are implications for summarisation that would spring from integrating polarity detection into the system, as described in section 8.2.1.

One of the central issues that will have to be addressed is the question of how we can maximally utilise the fact that we have at our disposal multiple classifications of the same word in different contexts. To some extent this is already done. The final score for each word output in the plot that the user sees basically reflects the system's confidence of classifying a word as either opinionated or non opinionated. Recall that the score is obtained starting from a value of zero, increasing it by one whenever the word is member of an n-gram that was classified as opinionated and decreasing the value by one whenever the word is included in an n-gram whose class label identifies it as non opinionated. This value is then divided by the number of n-grams this particular word has appeared in. Thus, if all n-grams in which the word appears have the same class label, the final score for the word is either one or negative one, indicating a high confidence of the system in the word's classification. Nevertheless, it should be possible to make better use of the information provided to us by the full set of classified n-grams.

As discussed above, multiple ways of summarising and presenting classification results are imaginable. Some of these solutions portrait the same information to the user by different means of representation. At the same time, some of the approaches not only differ in appearance, but also represent either different information garnered from the classification outputs or represent information with a different emphasis. At this point of the system's execution, the user has already been involved in the classification's outcome by providing arguments. Through a similar measure the user could provide additional judgement during the process of summarisation. We may present multiple of the different interpretations discussed in a manner that allows the user to compare them. The most basic realisation of this concept would be to present to the user both the original SVM classification and the classification adjusted according to the user's feedback. The user could then choose which of the two classification results he or she deems to be more accurate or, in a more advanced setting, determine different excerpts of the text where one or the other result produces the more apt result. An additional benefit to gain from this solution would be an evaluative one. Only if the user chooses the classification result based on the feedback significantly more frequently than the original SVM classification result would the extra effort of reclassification be justified. Other scenarios involving a second run of collecting user feedback during summarisation could be devised without yielding excessive demands to the user. For example, we might present the user with a similar plot as the one currently used during system execution, but allow him or her, once presented with this plot, to adjust it per mouse drag at places where the user deems the classification result inappropriate. The two proposals made above to introduce user feedback at this stage are by no means the only ways imaginable how this may be achieved, but they suffice to illustrate that considering this concept may yield further improvements in the system's performance.

8.2.5 On adjustable parameters

Throughout the system's execution, but also during the corpus' development, many parameters influence the eventual text classification that is the system's output. In the following, potential benefits, but also risks, of changing some of these parameters are discussed. Consider a non-exhaustive list of adjustable parameters below:

- SVM parameters
- TreeTagger parameters
- n-gram extraction criteria
- Maximum length of n-grams considered

As explained in section 7.1, training the SVM classifier meant testing various combinations of parameters in order to find an optimal combination. This need for extensive parameter testing is generally needed when using SVM and will have to be, to a certain extent, be repeated should some of the developments described in this chapter be realised. The current setting of SVM parameters yields the best possible classification performance on the test set, given the current form of the corpus. This means, were we to add more features to to the feature vectors, i.e. change the both the data that the SVM is trained on and the data it then classifies, the necessity would arise to reevaluate the current parameter settings.

For the application of TreeTagger, I have used the default settings of both the model data used for tagging and the parameters. Similarly to libSVM, TreeTagger offers both a training algorithm and a classification algorithm. We can train TreeTagger by supplying it with a lexicon and a set of hand annotated data. Training TreeTagger would thus entail a process that is analogous to the development of the data set that was constructed for SVM training, with differences in the syntax of the training data and an additional lexicon. Since TreeTagger, along with those algorithms, provides a readily trained model file that is based on the Penn TreeBank data, we are already given a high quality model file and investing the effort in constructing a model file from scratch was thus unnecessary. At the same time, one could expect that constructing both the lexicon and the training data for TreeTagger on the same grounds as the training data for the SVM instead of on a separate set of data may yield more accurate tagging results. Whether or not constructing such training data for TreeTagger and basing training on it subsequently would improve the results was not investigated due to the limitations of the project's scope. Both during training and during classification, a number of parameters can be adjusted to the data and to each other. Thus, training TreeTagger would entail an investigation of optimal parameter settings, which was of no concern during this project. Some of these parameters are concerned with the tagging procedure itself, such as:

- Setting tokenisation options
- Including heuristics in the tagging procedure
- Including an auxiliary lexicon
- Dealing with non word content

Other options determine the output that is written to the file once the tagging has been executed, such as the content that is stored or probability thresholds that have to be exceeded in order to store certain information. Some of the options available in TreeTagger may prove to yield more reliable tagging results when set to certain values or when providing certain additional information. This may in turn improve the classification results of the data annotated using TreeTagger.

The very first step in constructing the text corpus described in chapter 5 was to extract n-grams up to a length of five words together with their classification as being either opinionated or non opinionated. The decisions upon the class label of the n-grams and which n-grams should be extracted in the first place, were made based on the n-grams features, determined by the annotations. The annotation scheme used to construct the MPQA corpus provides a number of features that let us decide whether an n-gram annotated in such a manner is opinionated or non opinionated. For example, an n-gram whose annotation contains the feature-value combination *onlyfactive="no"* may

unambiguously be identified as opinionated and stored in the corpus with the according class label. A number of other features such as this have been used during the extraction of the n-grams from the MPQA corpus. However, this procedure did not result in an extraction of all n-grams contained within the MPQA corpus, despite the fact that it should be possible to assign any n-gram to one of the two categories. This means that by improving some of the extraction criteria and allowing a more holistic extraction result, we may be able to obtain a larger text corpus. Assuming that the n-grams additionally extracted are of the same quality and their classifications are as reliable, attaining a larger corpus may further improve the subsequent training and classification procedures.

A rather heuristic parameter setting has been employed in choosing the maximum length of n-grams that are processed by the system. Setting the maximum length of n-grams to five words was seen as a compromise between to aspects that seem to be mutually exclusive. On the one hand, we want to capture as many interrelations between words within a text. As discussed previously, often times opinions are not expressed by single words and can, in many cases, not be identified by analysing single words. This is the basic motivation for using n-grams rather than a bag-of-words approach during the analysis. Considering that the opinionatedness of a word may very well rely on a phrase that is positioned numerous words later, we could argue that the larger the n-grams we consider for our analysis, the better the chances of identifying opinions in general; but dealing with an ever increasing size of n-grams poses one crucial problem. Computationally, it might turn out to be infeasible to increase the maximum size of n-grams by a large number. As explained, if we split up user input into all possible n-grams up to a length of five words, we obtain $m = (w-2)*5$ n-grams, where w is the number of words we are analysing. Generalising the n-gram count to any maximum n-gram length we get

$$m = (w - n + 1) * n + \sum_{i=1}^{n-1} i \tag{8.3}$$

given that $w > n$, where n is the maximum n-gram length. Consider, for example, a text of 50 words. Considering all n-grams of a length up to five words would yield 240 n-grams. Extending the analysis to a maximum n-gram length of eight words would mean analysing 372 n-grams. Despite a moderate increase of n and a relatively small number of words that are to be analysed we have an increase in number of n-grams of 55%. When running the program, each n-gram is parsed multiple times in order to obtain the syntax needed for analysis. This included extensive search procedures, meaning that a large increase in analysed n-grams means a rather significant increase in computation, as well. We see that there is trade off between potential benefit in using larger n-grams to pay respect to dependencies among words and increased computational demands in processing larger and larger n-grams. Accordingly, we will have to investigate whether we could gain higher classification accuracy by considering larger n-grams. Equivalently evaluation is needed of whether we can achieve the same quality of results as we do now when analysing n-grams, say, just up to a maximum length of four words. Comparing the classification outcomes of different maximum n-gram lengths entails changing both some of the system's procedures and the corpus itself. The corpus will need to include all n-grams found in the MPQA corpus up to the n-gram length specified while the system will need to split the user input into a larger, or smaller, amount of n-grams.

8.2.6 On Linguistics

In section 2.2.1, two contributions of insights from linguistics to NLP and Sentiment Analysis have been pointed out.

1. The automatic procession of text along a fixed set of linguistic rules
2. The use of heuristics to prune search space or increase the contribution of a priori knowledge

While the first aspect constitutes a significant part of the realisation of the A-SVM system, the second does not. The entire process of extracting, annotating and classifying data relies on the

application of rules such as those determining the type of a word or its basic form. On the other hand, heuristics have neither been used to try and increase the efficiency of any procedures nor have they contributed to maximising the impact of information in any significant manner. Accordingly, incorporating heuristics may offer room for improvement in a twofold manner. It may be used to

1. Increase classification efficiency and accuracy
2. Increase the impact of gathered information

To increase classification efficiency and accuracy, a similar approach to that taken by Ding and colleagues [15] as described in section 2.2.1 may offer valuable improvements. Using either results from an evaluative study preceding the system's further development or evidence accumulated through multiple system executions, indicators for certain combinations of words and features being opinionated or non opinionated could be amassed to influence future classifications. If strong evidence for a certain succession of features being, say, opinionated are already available and we are confident in the value of these indicators, we may not ask the user to provide any more arguments on these specific feature combinations and instead allocate valuable user feedback to feature combinations we have less information about.

Similarly, the information gathered may be generalised to encompass other combinations of features of which we know, by some measure, that they are likely to be equally classified. Consider the following example: We have encountered two distinct combinations of features many times and each time a user classified both these features, he or she assigned them to the same class. During another run, both feature combinations appear in the analysed text, but the user provides feedback on only one of the two. If we trust the evidence collected during previous run, suggesting that whenever these two feature combinations appear together, they are assigned the same class label, we could generalise the user feedback to the second feature combination without the user ever judging it.

8.2.7 On Software Engineering aspects

A-SVM has been implemented in C++, using Qt for the GUI deployment. Some comments shall be made here about the way the system was implemented and how improvements may be achieved in the future. Two main aspects to point out are:

- The integration of external programs
- The handling of data storage in text files

During the system's execution, two external executable files are called:

- TreeTagger
- libSVM

These two processes fulfil a function that is crucial to the system as a whole and thus calling these procedures during run time poses a necessity. However, an improvement may be achieved by changing the method of integration of these components. Both these external systems could, in one way or another, be integrated into the system's execution in a less expensive manner. Some additional programming effort should enable an integration of those components' functionalities as regular calls within the programs environment, instead of having to call an external executable twice during a single system run. Parts of the two systems are available as CPP files and thus the basis for achieving this integration is already provided.

The second aspect we shall consider is the way how data is handled during system execution. The system's execution involves the generation and use of a number of text files. The information needed for conversion and classification is partially provided by text files such as the SWN lexicon,

```cpp
#include <fstream>
#include "conio.h"
#include <string>

using namespace std;

void convert_to_numerical(int, string, string)

int main(int argc, char *argv[])
{
        //...
        int final_ngram_count;
        convert_to_numerical(final_ngram_count, "user_input_annotated", "user_input_numerical");
        //...
}

void convert_to_numerical(int final_ngram_count, string filename_input, string filename_output)
{
        //Initialising the output stream
        ofstream output;
        output.open(filename_output, ios::out);

        //Initialising the input stream
        ifstream input;
        input.open(filename_input, ios::in);

        string class_label;
        string features_converted;
        //Conversion operations

        //stream
        int counter=0;
        string temp;
        while(counter < final_ngram_count)
        {
                //for each pass through the while loop, one line is extracted from the input stream
                getline(input, temp);
                //This line is then converted and written to the output file
                //conversion operations here
                output << class_label << "      " << features_converted << endl;
                ++counter;
        }

        output.close();
        input.close();
}
```

Figure 8.5: Example of input/output streaming as it is used during the system's execution to either convert intermediate results (as in the example) or add new features

the MPQA model file and the TreeTagger model file. In addition to that, each conversion step yields a text file that contains the representation of the user's input at that current stage. Storing the results of each intermediate step during the analysis process was mainly born from the motivation of retaining these outcomes for illustrative purposes. With regards to efficiency this solution is not optimal, since every step of processing involves *streams* for reading text and writing text to the files containing the results of each processing steps (see figure 8.5). Storing the results in vectors and passing them on from one processing step to the next, may for example be one choice that is more efficient.

As this project's focus lay rather on evaluating the propositions made with regards to their final outcome than on developing a highly performant system with regards to resource utilisation, their is likely to be room for improvement with regards to efficiency. These are, at the current stage of the system's development not necessarily crucial to its success, as all processing is executed at a reasonable speed. Nonetheless, considering some of the propositions made in this chapter, for example an increase of n-grams that are analysed at each system run, evaluating potential improvements in the system's efficiency may prove to be vital during further developments.

8.3 Conclusion

Sentiment Analysis is still a young research field, but throughout the past decade, progress has been made on all issues that this field entails. Nevertheless, there is much ground to be covered and it has been my motivation to make a contribution to this progress by means of this project. By providing both the theoretical foundation and motivation, and subsequently achieving measurable improvements compared to a unimodal pattern recognition procedure, I believe to have made a rather substantial case for the potential benefits of going beyond pattern recognition algorithms in Sentiment Analysis. As has been suggested by some, e.g. by Pat Langley in [39], co-founder of *Machine Learning* journal, it may prove to be necessary in the future to shift the focus away from

sheer statistical analysis back to more complex tasks as envisioned when Machine Learning was still a very young and emerging field. As Langley puts it:

> I do not believe that we should abandon any of the computational advances that have occurred in the [past] 25 years [...]. Each has been a valuable contribution to our understanding of learning. However, I think it is equally important that we not abandon the many insights revealed during the field's early period, which remain as valid today as when they initially came to light. The challenge for machine learning is to recover the discipline's original breadth of vision [...].

The suggestion of viewing Machine Learning as an experimental science that builds solutions upon a combination of statistical analysis and *symbolic* representations of knowledge reflects the basic motivation of this project. By including both a Support Vector Machine algorithm and logic based arguments to represent the knowledge amassed by the system, and subsequently evaluating the system both quantitively and qualitatively, I have made the attempt to pay respect to the inclination of broadening the view on one particular Machine Learning problem: Sentiment Analysis.

Bibliography

[1] C. Baker, C. Fillmore, and J. Lowe. The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

[2] M. Baroni and S. Vegnaduzzo. Identifying subjective adjectives through web-based mutual information. In *Proceedings of KONVENS*, volume 4, pages 17–24. Citeseer, 2004.

[3] C. Bishop. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[4] T. Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics, 2000.

[5] E. Breck, Y. Choi, and C. Cardie. Identifying expressions of opinion in context. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[6] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of the National Academy of Sciences*, 97(1):262, 2000.

[7] C. Chang and C. Lin. Libsvm: a library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`, 2001. [Online; accessed 31-August-2011].

[8] Y. Choi and C. Cardie. Learning with compositional semantics as structural inference for subsentential sentiment analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 793–801. Association for Computational Linguistics, 2008.

[9] Y. Choi, C. Cardie, E. Riloff, and S. Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 355–362. Association for Computational Linguistics, 2005.

[10] CIA. World fact book. `https://www.cia.gov/library/publications/the-world-factbook/geos/uk.html`, 2011. [Online; accessed 31-August-2011].

[11] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine learning*, 3(4):261–283, 1989.

[12] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] H. Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[14] K. Dave, S. Lawrence, and D. Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM, 2003.

[15] X. Ding and B. Liu. The utility of linguistic rules in opinion mining. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 811–812. ACM, 2007.

[16] H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization. *Neural Networks, IEEE Transactions on*, 10(5):1048–1054, 1999.

[17] K. Eguchi and V. Lavrenko. Sentiment retrieval using generative models. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 345–354. Association for Computational Linguistics, 2006.

[18] A. Esuli and F. Sebastiani. Determining the semantic orientation of terms through gloss classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 617–624. ACM, 2005.

[19] A. Esuli and F. Sebastiani. Determining term subjectivity and term orientation for opinion mining. In *Proceedings the 11th Meeting of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, pages 193–200, 2006.

[20] A. Esuli and F. Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of LREC*, volume 6, pages 417–422. Citeseer, 2006.

[21] J. W. et al. Mpqa readme file. `http://www.cs.pitt.edu/mpqa/databaserelease/Database.2.0.README`, 2011. [Online; accessed 31-August-2011].

[22] T. Furey, N. Cristianini, N. Duffy, D. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906, 2000.

[23] D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.

[24] N. Godbole, M. Srinivasaiah, and S. Skiena. Large-scale sentiment analysis for news and blogs. In *Proceedings of the International Conference on Weblogs and Social Media (ICWSM)*. Citeseer, 2007.

[25] R. Grishman. Information extraction: Techniques and challenges. *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*, pages 10–27, 1997.

[26] T. Guardian. Latest news, comment and reviews from the Guardian. `http://www.guardian.co.uk/`, 2011. [Online; accessed 31-August-2011].

[27] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1):389–422, 2002.

[28] V. Hatzivassiloglou and J. Wiebe. Effects of adjective orientation and gradability on sentence subjectivity. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 299–305. Association for Computational Linguistics, 2000.

[29] J. Hockenmaier and M. Steedman. Ccgbank: a corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396, 2007.

[30] C. Hsu and C. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[31] D. Jurafsky, J. Martin, A. Kehler, K. Vander Linden, and N. Ward. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, volume 163. MIT Press, 2000.

[32] J. Kamps, M. Marx, R. Mokken, and M. de Rijke. *Words with attitude*. Citeseer, 2001.

[33] W. Karush. Minima of functions of several variables with inequalities as side conditions, 1939. Master thesis.

[34] S. Kim and E. Hovy. Determining the sentiment of opinions. In *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004.

[35] S. Kim and E. Hovy. Automatic detection of opinion bearing words and sentences. In *Companion Volume to the Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*, 2005.

[36] L. Ku, Y. Liang, and H. Chen. Opinion extraction, summarization and tracking in news and blog corpora. In *Proceedings of AAAI-2006 Spring Symposium on Computational Approaches to Analyzing Weblogs*, pages 100–107, 2006.

[37] H. Kuhn and A. Tucker. Nonlinear programming. In *Second Berkeley symposium on mathematical statistics and probability*, volume 1, pages 481–492, 1951.

[38] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Machine Learning - International workshop then conference*, pages 282–289. Citeseer, 2001.

[39] P. Langley. The changing science of machine learning. *Machine Learning*, 82:275–279, 2011.

[40] B. Liu. Sentiment analysis and subjectivity. *Handbook of Natural Language Processing*, 2, 2010.

[41] B. Liu, M. Hu, and J. Cheng. Opinion observer: Analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, pages 342–351. ACM, 2005.

[42] C. Macdonald, I. Ounis, and I. Soboroff. Overview of the TREC 2007 blog track. In *Proceedings of TREC 2007*, 2007.

[43] C. Manning, H. Schütze, and MITCogNet. *Foundations of statistical natural language processing*, volume 59. MIT Press, 1999.

[44] M. Marcus, M. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[45] G. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[46] T. Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 1997.

[47] M. Mozina, P. Tolchinsky, and U. Cortes. Project N002307 ASPIC, 2004.

[48] M. Mozina, J. Zabkar, T. Bench-Capon, and I. Bratko. Argument based machine learning applied to law. *Artificial Intelligence and Law*, 13(1):53–73, 2005.

[49] M. Mozina, J. Zabkar, and I. Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10-15):922–937, 2007.

[50] T. Mullen and N. Collier. Sentiment analysis using support vector machines with diverse information sources. In *Proceedings of EMNLP*, volume 4, pages 412–418, 2004.

[51] T. Nasukawa and J. Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

[52] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67. Citeseer, 1999.

[53] Nokia. Qt - a cross-platform application and ui framework. `http://qt.nokia.com/`, 2011. [Online; accessed 31-August-2011].

[54] C. Osgood, G. Suci, and P. Tannenbaum. *The measurement of meaning*. University of Illinois Press, 1971.

[55] I. Ounis, M. De Rijke, C. Macdonald, G. Mishne, and I. Soboroff. Overview of the TREC-2006 blog track. In *Proceedings of TREC*, volume 6. Citeseer, 2006.

[56] I. Ounis, C. Macdonald, and I. Soboroff. On the trec blog track. In *Proceedings of the International Conference on Weblogs and Social Media (ICWSM)*, 2008.

[57] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[58] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

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

[60] B. Pang and L. Lee. Using very simple statistics for review search: An exploration. In *Proceedings of the International Conference on Computational Linguistics (COLING)*. Citeseer, 2008.

[61] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[62] H. Prakken and G. Vreeswijk. Logics for defeasible argumentation. *Handbook of philosophical logic*, 4:218–319, 2002.

[63] C. U. Press. Cambridge Dictionary Online. `http://dictionary.cambridge.org/`, 2011. [Online; accessed 31-August-2011].

[64] A. Ratnaparkhi et al. A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142, 1996.

[65] E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the national conference on Artificial Intelligence*, pages 1044–1049, 1996.

[66] E. Riloff and J. Wiebe. Learning extraction patterns for subjective expressions. In *Proceedings of the 2003 conference on Empirical methods in natural language processing-Volume 10*, pages 105–112. Association for Computational Linguistics, 2003.

[67] E. Riloff, J. Wiebe, and T. Wilson. Learning subjective nouns using extraction pattern bootstrapping. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 25–32. Association for Computational Linguistics, 2003.

[68] G. Rios and H. Zha. Exploring support vector machines and random forests for spam detection. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, pages 284–292, 2004.

[69] J. Ruppenhofer, S. Somasundaran, and J. Wiebe. Finding the sources and targets of subjective expressions. In *Proceedings of LREC*. Citeseer, 2008.

[70] H. Schmid. Probabilistic part-of-speech tagging using decision trees, 1994.

[71] P. Stone, D. Dunphy, and M. Smith. *The General Inquirer: A Computer Approach to Content Analysis.* MIT Press, 1966.

[72] C. Sutton and A. McCallum. 1 An Introduction to Conditional Random Fields for Relational Learning. *Introduction to statistical relational learning*, page 93, 2007.

[73] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 116. Cambridge Univ Press, 1998.

[74] T. N. Y. Times. The New York Times - Breaking News, World News & Multimedia. `http://www.nytimes.com/`, 2011. [Online; accessed 31-August-2011].

[75] P. Turney. Mining the web for synonyms: PM-IR vs LSA on TOEFL. *Proceedings of ECML'01*, pages 491–502, 2001.

[76] P. Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[77] H. Wallach. Conditional random fields: An introduction. *Rapport technique MS-CIS-04-21, Department of Computer and Information Science, University of Pennsylvania*, 50, 2004.

[78] C. Whitelaw, N. Garg, and S. Argamon. Using appraisal groups for sentiment analysis. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 625–631. ACM, 2005.

[79] J. Wiebe. Learning subjective adjectives from corpora. In *Proceedings of the National Conference on Artificial Intelligence*, pages 735–741. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000.

[80] J. Wiebe, E. Breck, C. Buckley, C. Cardie, P. Davis, B. Fraser, D. Litman, D. Pierce, E. Riloff, T. Wilson, et al. Recognizing and organizing opinions expressed in the world press. In *Working Notes-New Directions in Question Answering (AAAI Spring Symposium Series)*, 2003.

[81] J. Wiebe, T. Wilson, R. Bruce, M. Bell, and M. Martin. Learning subjective language. *Computational linguistics*, 30(3):277–308, 2004.

[82] J. Wiebe, T. Wilson, and C. Cardie. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation*, 39(2):165–210, 2005.

[83] T. Wilson. *Fine-grained subjectivity and sentiment analysis: Recognizing the intensity, polarity, and attitudes of private states.* PhD thesis, Citeseer, 2008.

[84] T. Wilson, P. Hoffmann, S. Somasundaran, J. Kessler, J. Wiebe, Y. Choi, C. Cardie, E. Riloff, and S. Patwardhan. OpinionFinder: A system for subjectivity analysis. In *Proceedings of HLT/EMNLP on Interactive Demonstrations*, pages 34–35. Association for Computational Linguistics, 2005.

[85] H. Yu and V. Hatzivassiloglou. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In *Proceedings of the 2003 conference on Empirical methods in natural language processing-Volume 10*, pages 129–136. Association for Computational Linguistics, 2003.

[86] J. Zabkar, M. Mozina, J. Videcnik, and I. Bratko. Argument based machine learning in a medical domain. In *Proceeding of the 2006 conference on Computational Models of Argument: Proceedings of COMMA 2006*, pages 59–70. IOS Press, 2006.

[87] M. Zhang and X. Ye. A generation model to unify topic relevance and lexicon-based sentiment for opinion retrieval. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 411–418. ACM, 2008.

[88] J. Zhao, K. Liu, and G. Wang. Adding redundant features for CRFs-based sentence sentiment classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 117–126. Association for Computational Linguistics, 2008.

Appendix A

Questionnaire

Instructions

Dear evaluation participant,

I would like to thank you for taking your time to participate in the evaluation study that I am conducting as part of Individual Master Project at Imperial College, London.

You will be evaluating a system that performs Sentiment Analysis. It is the purpose of the system to disseminate text according to it either being opinionated or non opinionated.

The purpose of this evaluation is to collect a personal impression and user experiences of working with the prototype of the system you are about to evaluate.

As mentioned the system is a prototype which is still in the early stages of its development. The evaluation you are participating in is the first one in a development and evaluation cycle. These so called Usability evaluations aim, on the one hand, at testing the current state of the art. The actual focus lies on collecting data and information as to how the system may be improved upon in the further development.
You are asked to test the system twice, once entering a piece of text that is provided below and once entering a sentence of your own choosing.

Handling the system is mostly self explanatory. Should any problems occur nonetheless, please do not hesitate to ask for assistance.

Please start the system now. When you are prompted to enter text, please type the following sentence and hit the *Submit!* button. Disregard the numbers when typing the text, they are given to understand the classification output more intuitively.

1	2	3	4	5	6	7	8	9	10
The	former	prime	minister	warns	that	rash	talk	of	a
11	12	13	14	15	16	17	18	19	20
broken	society	threatens	to	harm	the	country	‘s	reputation	abroad

Once you have finished the first run executing the program from start to finish, please fill out page 2 of this questionnaire. After you have done so, execute the program for a second time, this time entering a sentence of your own choosing when asked to provide text.

After finishing this second run, please fill out page three of the questionnaire.

Thank you again for your time,

Lucas Carstens

Figure A.1: Questionnaire page 1: Instructions

	completely agree	agree	neutral	don´t agree	completely disagree
The system was easy to use	❑	❑	❑	❑	❑
It was easy to provide user feedback	❑	❑	❑	❑	❑
I understand the benefit of the feedback	❑	❑	❑	❑	❑
The classification results were appropriate for the input	❑	❑	❑	❑	❑
The representation of the classification results was intuitively understandable	❑	❑	❑	❑	❑
Automatic detection of opinions is useful	❑	❑	❑	❑	❑

On a scale from 1 to 5, please rate the quality of the system output with regards to the correctness of classification: _____

Figure A.2: Questionnaire page 2: Questions following *Run 1*

	completely agree	agree	neutral	don´t agree	completely disagree
The classification results were appropriate for the input	❑	❑	❑	❑	❑
Providing feedback was cumbersome	❑	❑	❑	❑	❑

On a scale from 1 to 5, please rate the quality of the system output with regards to the correctness of classification: _____

Figure A.3: Questionnaire page 3: Questions following *Run 2*

Appendix B

Mathematical background

The explanations of Lagrange multipliers and the Karush-Kuhn-Tucker conditions are adapted from [3], where more detailed examples can also be found.

B.1 Lagrange multipliers

Lagrange multipliers are used to determine the stationary points of a function of multiple variables, subject to one or more constraints. Consider a D-dimensional variable x with $x_1, ..., x_D$. We then set a constraint equation $g(x) = 0$ to represent a $D - 1$ dimensional surface in the x-space. We then find a point $x*$ on this constraint surface that maximises $f(x)$. The point $x*$ that maximises $f(x)$ has the property that the gradient $\nabla f(x)$ is orthogonal to the constraint, because $f(x)$ could otherwise be increased by moving along the constraint surface. This means that the gradients of $f(x)$ and $g(x)$ are parallel, or anti-parallel, vectors. Thus we know that the exists a parameter λ for which

$$\nabla f + \lambda \nabla g = 0 \tag{B.1}$$

where $\lambda \neq 0$ is the Lagrange multiplier. The Lagrangian function is denoted by

$$L(x, \lambda) \equiv f(x) + \lambda g(x). \tag{B.2}$$

B.2 Karush-Kuhn-Tucker conditions

Above, we have considered maximising $f(x)$ the equality constraint equation $g(x) = 0$. When instead maximising $f(x)$ subject to the *inequality* constraint $g(x) \geq 0$, we obtain two kinds of possible solutions instead of one.

- The constrained stationary point may lie within a region where $g > 0$. In this case, the constraint is *inactive*.
- The constrained stationary point may lie on the boundary where $g = 0$. In this case, the constraint is *active*.

In the case where the constraint is *inactive*, the function $g(x)$ has no relevance and the stationary condition is just $\nabla f(x) = 0$, i.e. $\lambda = 0$ in equation B.1. When the constraint is *active*, we have $\lambda \neq 0$, instead. The difference to the equality constraint in this case is that the sign of the Lagrange multiplier may *not* be either positive or negative, because the function $f(x)$ will only be at its maximum if its gradient is oriented away from the region $g(x) > 0$. It is therefore imperative that

$$\nabla f(x) = -\lambda \nabla g(x) \tag{B.3}$$

fore some value of $\lambda > 0$. For both cases we have that $\lambda g(x) = 0$. "Thus the solution to the problem of maximising $f(x)$ subject to $g(x) \geq 0$ is obtained by optimising the Lagrange function B.1 with respect to x and λ subject to the conditions

$$g(x) \geq 0 \tag{B.4}$$
$$\lambda \geq 0 \tag{B.5}$$
$$\lambda g(x) = 0 \tag{B.6}$$
$$\tag{B.7}$$

These are known as the *Karush-Kuhn-Tucker* (KKT) conditions ([33], [37])." ([3]).