MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Preprocessing and multi-dimensionality in an unsupervised opinion extraction system

LICENSE THESIS

Graduate: **Cristian Alexandru COSMA**

Supervisors: **Prof. dr. eng. Rodica POTOLEA**
Prof. dr. eng. Mihaela DINSOREANU

2014

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Cristian Alexandru COSMA**

Preprocessing and multi-dimensionality in an unsupervised opinion extraction system

1. **Project proposal:** *In a world where unstructured information is becoming more and more abundant, the need to give meaning and structure to it is ever increasing. An unsupervised learning system, independent of domain is proposed in this thesis. Having its roots in grammar and the relations between words, the approach extracts subjective information from preprocessed text and assigns a sentiment polarity to each extracted feature.*

2. **Project contents:** *Presentation page, Advisor's Evaluation, Introduction, Project Objectives, Bibliographic Research, Analysis and Theoretical Approach, Detailed Design and Implementation, Testing and Validation, User's Manual, Conclusions, Bibliography, appendices.*
3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**:

5. **Date of issue of the proposal:** March 1, 2013

6. **Date of delivery:** July 3rd, 2014

Graduate: Alexandru Cristian COSMA

Supervisors: dr. prof. eng. Rodica POTOLEA
dr. prof. eng. Mihaela DINSOREANU

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) COSMA CRISTIAN ALEXANDRU, legitimat(ă) cu C.I. seria XH nr. 370024, CNP 1910124055084, autorul lucrării *Preprocessing and multi dimensionality in an usupervised opinion extraction system* elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea Calculatoare din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea IULIE a anului universitar 2014, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data	Nume, Prenume
03.07.2014	Cosma Cristian Alexandru
	Semnătura

Table of Contents

Table of Figures

1. Introduction

In a world where subjectivity dictates trends and objectivity is left for politics and laws, mining subjective information has become more and more relevant. People all around the world are sharing their opinions about everything from products to movies, music and world events. We see stock markets go up or down based on propagated opinion of possible conflicts rising in different parts of the world. We see music stars being brought down and actors being decimated or praised by critics. Everything is subjective opinion which can be analyzed, quantified and used to extract patterns and/or behavior.

1.1. Project context

There is enough information to feed everyone's need and more however too much of it is in unstructured form. Giving some meaning or structure to information is a delicate and sometimes cumbersome process and it basically involves teaching the machine (*the computer*) to think like we do and do something which only we would be able to do: *organize and give meaning.*

1.1.1. Data mining

From time to time, a new field of study which ends up attracting a lot of attention *"pops out"* and starts gaining momentum. In the era of the internet, cloud computing and huge social networking, the field of *data mining* was bound to pick pace. It started out in big companies which, from the beginning, offered services like *e-mail* and *internet vending*, companies like Yahoo or IBM. They started dealing with concrete *big data* in the form of text or pictures which caused a lot more than just space issues.

Dealing with *big data* creates both big responsibilities and great opportunities. Take for example Googles' search engine. Millions if not billions of searches are being performed each day throughout the world. Now one could think that there's not much to it, people search and Google simply forgets and returns the desired information. But that's a naïve way of thinking and it's really not what actually happens. Google learns, and it does so from what *you* searched, what *your friends* searched and what other only geographically related people searched. It looks through your interests and provides relative ads, results and information. This is certainly something very different from the naïve point of view presented a bit earlier. Google learns and it's the only one who does.

All these actions of learning from everyone's behavior and actions are related to data mining and analysis. The world would look terribly different without these hidden processes which take place without us even noticing it. Imagine if Google would not have provided such relevant and user-specific content with its search. It would have ended up as being just another ordinary search engine and we would not have had *Android*, for example. That accounts for roughly 1 billion devices world-wide with a totally different operating system. But Google did make it big, and it's almost all due to data mining and their ingenious ways of using that extracted data to solve our problems.

Yes, they are our problems and yet we did not even know we had them. The curious thing about innovation is that it usually takes people by surprise, solving a need which they did not even know they had. Except for the innovators who are actually involved in the process of creating new technologies and ways of making everyone's life

easier, the everyday user never felt like content-sensitive information is a must on a simple Google search, until they actually got that! Take it away from them now, as they are already accustomed to it, and they will feel like something is different, and this is only one of the beauties of innovation.

Data mining innovated and created countless opportunities to expand in just as many domains. Imagine a simple and cute example of *data mining*: which color is the most present one on *Facebook* pictures? Building something like this would involve creating a complex crawler and simple saving every photo publically available on Facebook and then analyzing it pixel by pixel. Daunting task but it provides a great incentive on what colors people find pleasing the most nowadays.

1.1.2. Opinion mining

The field of opinion mining is a branch of data mining which deals with subjective information which is to be retrieved from text source materials. Its goal is to offer an accurate (as close to how humans perceive opinion) process of extracting opinion and classify it or analyze it. The field combines natural language processing (NLP), data mining, artificial intelligence and computational linguistics. All these are interleaved into creating a system which simulates how humans extract opinion.

Opinion is everywhere. From BBC comments to Amazon reviews, subjective opinion is preparing to become the most read thing on the internet. Of course there is great interesting in mining something so huge! As we will see in the next chapters, there are quite a few reasons for doing so, and quite a few methods of doing it as well. What we have tried to do is offer a more general approach to *opinion mining*, an approach which does not take into account and is not affected by context or domain. Yes, it is language dependent however it is an unsupervised system which eliminates context dependency and thus covers a broader spectrum of texts which can be analyzed.

1.2. Thesis structure

The structure of everything hereafter is as follows:

- **Chapter 2** conveys a general overview of the purpose of this thesis together with some sub-goals, reasoning and motivation for choosing the subject and some specification of the underlying project

- **Chapter 3** covers related work which was done in the same field, a bibliographic study that was done before this thesis was even began. Here, the reader will find similar approaches to the same problem and the main differences to this approach.

- **Chapter 4** illustrates the theoretical foundation on which this project was built on. This includes some techniques and algorithms which form the backbone of the system and which ended up being used in its implementation.

- **Chapter 5** presents the implementation details covering design decisions and code-level analysis of the project. Here, the reader can see how the theoretical foundation found in chapter 4 was abstracted and represented into code.

- **Chapter 6** revolves solely around how the system was evaluated and tested, which parameters were involved and how it behaved in different situations. Measurements and *before and after* tests are also present in this chapter.

- **Chapter 7** consists of a step-by-step guide to installing and using the built system accordingly.

- **Chapter 8** presents a summary of the proposed solution with emphasis on scientific contributions, results and innovation.

- The **Bibliography** is also provided at the end of the paper

- The final section is comprised of an **Appendix.**

2. Project Objectives

2.1. Problem statement and motivation

The main purpose and motivation can be found in the fact that, at the moment, due to the huge impact the internet is having on the world, opinions are present almost everywhere and people are relying on them more and more as a knowledge source. Terrifying thought if one thinks that some people may actually be looking for *facts* but are fed *subjective information* (opinions). Moreover, in the area of business intelligence, companies are become keener on analyzing their products' reviews given by their customers more thoroughly. They are trying to eliminate the cumbersome process of analyzing them *by hand*, having a team of analysts which continuously read reviews and collect data from them. Yes, that might offer quality over quantity, but is it really worth it, and is it really a compromise which technology cannot solve?

More and more interest has been given to fields such as data mining, big data analysis and so on. Opinion mining is the *new kid on the block*. Analyzing metadata like the time of the day on which the most users post comments on social networks is one thing. Analyzing the content which is posted in itself is another. A more difficult and challenging task, yes, but a much more rewarding one as well.

There is a huge difference between the types of text one was able to find up until around 10 years ago in any source, and what the current trend is. Until 100 years ago, the only text which was publically available could be found in books (99%) and newspapers and magazines. Up until 50 years ago, television was nowhere to be found and social media was not born yet. With the coming of the internet which gained momentum roughly 20 years ago, everything started heading in a totally different direction. If 100 years ago, 99% of the information was to be found on actual, written or printed paper, nowadays, 99.99% of the information can be found on *electronic paper*, on the internet. This shift of paradigm brought changes not only to the way people interact with information but with the way information is being delivered to the everyday person. Objectivity started to slowly but steadily make way for subjectivity. Everyone had the right to an opinion before as well, but unless you were a known publisher, public speaker or government representative, no one would actually listen to it. However, things have changed. People care less and less about the social status of the person which actually conveys an opinion and more and more about other factors like context, feeling and how much they resonate with what that person is saying. I may be interested in the war in the Middle East, but CNN may seem too politically unbiased for me. I may even consider BBC as being subject to influence from certain political parties but when 20 years ago one had almost no choice but to listen to what was available, now people have choices; and plenty of them. It is nothing unheard of to prefer a speaker which is no one on the social rank but exceeds in creating a *bond* with its listener. There are countless examples of such behavior all over the internet with YouTube channels being filled with charismatic social-unknowns which gather followers (or subscribers) only by their charisma. Fact is being replaced by subjectivity and feeling and that is indeed something worth following more closely.

There are so many ways in which a person can get informed nowadays. Information has become more and more abundant. Twitter has reported an astonishing 51

million tweets each day. Amazon and EBay both gather hundreds of thousands of reviews every day and news channels/sites like CNN or BBC manage to receive thousands of comments for every post. These are all but a few of the multitude of sources in which information surges like never before; subjective information, for the most part. Never before has the world felt the influence of subjectivity as strongly as it is right now. Never before did one man with an opinion have so much opportunity to share and make it viral in just a matter of hours. All these are examples of reasons for which opinion mining has begun to flourish in the past few years. Let us examine a few more domains in which opinion mining can be used and the effects it might have there.

- *Product review:* of course, the first subject that comes to mind is the one which almost every internet user deals with almost daily. Products are being reviewed every day by many people around the world. Their weaknesses and strengths are thus portrayed and this helps the reader form a (*biased!*) opinion about that product. We mentioned biased because every opinion is subjective. Everything is relative! *"Long battery life"* might mean 3 hours for one user and 7 for the other. However, here is where opinion mining kicks in. If from thousands and thousands of reviews the majority talk about a *good* battery life (be that as relative as it may), you are most probably dealing with a product with a long and good battery life. The user might be discouraged from buying a product if the first 3 out of 500 reviews mention a weak battery! He/she will just stop reading any more reviews and will go on to the other product. There is a dire need to improve this process and opinion mining is the way to go.
- *Geopolitics:* In early 21st century, the *anonymous* movement began taking a stand for the injustices which were portrayed in everyday news and news bulletins. Gathering enough information about what is happening from sources like Twitter or Facebook, anonymous managed to help with quite a few conflicts including the Egyptian riots which at first led to a complete internet ban of the population. This was all subjective opinion being analyzed and processed, manually, yes, but imagine hearing the cries for help which may end up in some hidden forum and being able to act accordingly and in due time!
- *Stock Market:* As curious as it may seem at first, the stock market is influenced by world events and public opinion. If on specialized forums people will start talking about some stocks going down in the next period, others will see this as a threat and start selling thus entering a vicious circle which can end up in disrupting a companies' stock sales. Stock predictions are currently being built on top of stock evolution but rarely do those systems take into account the general *feeling* people have about a certain company selling at the stock market.

Imagine the impact which mining opinions can have on a day to day basis. From geopolitical influences to simple product reviews, opinion mining covers a broad domain spectrum which, if used properly, can ease the life of so many. Going online to search for a product may change its dynamic when the user will no longer be needed to read through the reviews but will be provided with a list of key-value pairs corresponding to the most

talked about aspects of that product and the average opinion on them, gathered from the other users. Knowing when to post a risky or controversial comment on a lengthy forum without reading all the topics' comments may become trivial.

2.2. Objectives

The main objective of this thesis and its backing project is to create a multi-dimensional tunable machine which is capable of doing unsupervised opinion mining on domain independent corpora. This solution comes from the need of automating extraction of opinion from source material for any of the purposes in any domain mentioned in the previous subchapter. The secondary purpose of the system is to cover the polarity assignment of the newly extracted opinions as to convey the user more information about that opinion.

Offering to eliminate the *human element* which is currently being used in opinion mining, the system relinquishes countless human hours which are needed to analyze texts and extract some valuable information from them; hours which, if the work currently being done *by hand* is to be replaced by this system, will be put to better use in other ways. The opinion extraction module can be used independently of the polarity aggregation one, however, not the other way around. The last module, the polarity aggregator depends on the output of the opinion extractor, as it is to be seen in the later chapters of this thesis. The sub-goals which are achieved in the full process are the following:

- Obtain a preprocessed version of the input data containing annotated text and generated syntactic trees for each sentence
- Extract *target* and *opinion word* tuples or triplets using the opinion extractor module and the above mentioned preprocessed corpus and syntactic trees
- Assign polarities to the extracted tuples based on some algorithm which covers the problem of context

Because there are more levels of granularity which can be used when tackling the opinion mining / sentiment analysis problem, our solution covers at least two of them as well. The first one, the *document level* analysis, offers a more *high level* view and results which for each input document. This helps create a general feeling of a set of reviews for example. The second granularity level which is covered by the system is the *aspect based* one which basically offers results for each target / aspect individually. Now this is very useful when you are trying to find out what others' opinions on a specific aspect is. For example, let's say you are the general manager of *CameraInc* which is a producer of digital cameras. You have just released a new model, the *BFG-LP2014* and already have 1420 reviews on EBay and Amazon. With this new model you have tested out a new lens technology which should eliminate shutter and noise and thus you are interested in what people think about the camera lens of this model. With the help of our system you can easily extract the opinion about the cameras' screen and see whether or not people are noticing any improvements from the previous models and what is their general feeling on that aspect.

As to highlight the exact focus of this thesis, the following parts of the system are covered in this paper:

- The *preprocessing module* in all its beauty with thorough explanations of every step, from input extraction to why lemmatization is favored against stemming, parsing and syntactic tree generation
- The *target extraction* part of the opinion extraction module
- The system parameters which give it the status of *multi-dimensional tunable machine*

The above stated objectives come from the basic need of easing up the human work needed in order to cover opinion mining and offer true business intelligence. The question which can be asked here is *couldn't this be made easier?* Many attempts to make it easier have been tried but did not offer a sufficiently good result. For example, the *star system* which is now available in any of its variations in almost every review-based feedback section from any product-selling website is such an attempt. However it offers minimal information which does not even get close to how much data can be extracted from the review itself. Imposing formatting restrictions and/or other constraints on the reviews is yet another solution, but it has never been tried nor implemented; and for a good reason! The user/customer almost never enjoys being constrained when he/she has an opinion about a certain product or idea. Either way you're looking at it, you're going to have a lot of unstructured data to handle and that's the main problem here, dealing with it!

Our unsupervised approach tried to *dance* with those problems and offer an elegant way of solving them. *Is it a perfect solution?* Not by a long shot, however it's a first step into the newly evolving universe, the opinion mining one.

2.3. Requirements

The starting point of each application is a set of requirements. These requirements can be either functional or non-functional, one dealing with what the application should be able to do and the others cover constraints and requirements on how these functional specifications should be implemented. The following section presents the two types of requirements.

2.3.1. Functional Requirements

The behavioral and functional aspects of the application are covered by the functional requirements. These requirements are basically the tasks which have to be done in order to get to the final goal. In our case, the functional requirements would be ***text preprocessing, opinion extraction and polarity aggregation*** each corresponding to a module of the application and intertwined offering the final goal, that of sentiment analysis.

The system will receive as input a set of files containing text. This text is to be preprocessed and syntactic trees will have to be created for each particular sentence. As a file is taken and preprocessed as a whole, we presume each file corresponds one subject or product which is to be analyzed. This presumption simplifies things in the way that the system does not have to keep track and associate texts coming from different files to the

same product. Obtaining these syntactic trees is the goal of the first functional requirement, the text preprocessing.

The second requirement will be accomplished by extracting tuples from the syntactic trees obtained from the previous module, tuples which contain either targets and/or opinion words. These tuples can be pairs or triplets depending of the rules used for extracting and can thus contain two or three words. The tuples will contain the relation between the containing words and each word will contain its associated value (the actual word), POS tag and any additional relevant fields. These extracted tuples will then be passed to the polarity aggregator to fulfill the final functional requirement.

Polarity aggregation will involve assigning polarity to the extracted tuples. The proposed algorithm should consider context when assigning the polarities to the words and should both assign them to the targets and the opinion words in an iterative manner.

2.3.2. Non-functional requirements

System qualities, as they are also called, are the requirements used to evaluate the behavior of a system. There is quite a long list of non-functional requirements but we will only analyze those which are relevant to our system.

- **Efficiency**: Dealing with big data is a pain when efficiency comes into play. The system must use resources in the most efficient way possible as to offer a decent processing time, therefore, the preprocessing step should only be done once for every file and then cached in order to save time.
- **Scalability**: Large amounts of data should not be an issue as the system should be able to operate and perform its tasks with small or large input all the same.
- **Modularity**: Adding more modules (like the ones which will be presented in chapter 8.2) should not cause issues nor require great architectural changes. Adding more modules should be a seamless problem with as little implementation impediments as possible.
- **Accuracy**: The system should provide accurate results. Delicate as it may seem, a *close to as if it were random* result set is of no use when dealing with opinion mining thus the system should provide good results both in terms of precision and recall.
- **Reliability**: Feeding the system with invalid or corrupt data should yield errors not crash it. Error handling should be present in all the layers of the applications.

Of course there could be other, less important non-functional requirements but which will not be mentioned here. Security might be a problem once a crawler is created on top of the application which should provide constant input to it, however, this is not yet the case.

3. Bibliographic Research

In this chapter we will walk through the research and additional reading which has been done prior to writing this thesis. Concepts and ideas relevant to the domain in question are presented from the point of view of different authors. The roots of this thesis will be found in the following subchapters. It is of great importance to present the work done by other researchers on the topic as it conveys a wider (if not also deeper) understanding of the problems which are encountered when dealing with these topics. Some articles present the background of opinion mining and others may present the starting point or *trigger ideas* which led to the development of this thesis. Reading this whole chapter is strongly recommended before venturing into the theoretical foundation and implementation details which are presented in later chapters.

3.1. Opinion Mining and Sentiment Analysis

The fast ascent of social media such as blogs and social networks sparked great interest in opinion mining and sentiment analysis. With reviews, recommendations and other such online means of expression, online opinion has become more and more valuable for businesses looking to explore a new market or come up with something innovative in an already saturated one. Businesses now want to automate their filtering process, understand forum conversations and chats about their products and these are done with the use of opinion mining. More and more research is being done in breaking the domain-dependence limitations as a business might sell different products each from a different domain (cameras, laptops) but would still want to use only one analysis tool to do the job.

3.1.1. Natural Language Processing

Natural language processing (NLP) stands at the base of this thesis's approach and it envelopes all the processing techniques and operations which deal with human language. Machines do not yet (and may never) understand human language like we do. For them, language is just a bunch of unstructured text which has no meaning put together, whatsoever. So we can start seeing now how NLP can be a bit tricky. It's somehow related to teaching a kid how to talk and react to certain phrases and expressions using rules and give meaning to what he/she (and now *it*) hears or reads. In [1] the author clearly states that one of the most daunting tasks of natural language processing is extracting the exact amount of information needed in order to structure the text as efficient as possible.

The most documented and widely used tool for natural language processing is by far the Stanford CoreNLP [2]. It provides all the necessary processing tools from sentence splitting and tokenizing to lemmatizing, stemming and parsing. It is fully compatible with English, Chinese and German texts and is terribly simple and intuitive to use. It is free, open source project developed by the Stanford Natural Language Processing Group.

Another tool which is used however less often is the Natural Language Toolkit (short NLTK) [3]. NLTK is a leading platform for building Python programs to work

with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, and tagging, parsing, and semantic reasoning.

3.1.2. Sentiment Lexical Resources

Every sentiment analysis tool needs a sentiment lexical resource. A lexical resource is a sort of database consisting of one or more dictionaries. In the case of opinion mining, a sentiment lexical resource is such a dictionary which contains polarities for each word. A polarity is simply a score (usually between -1 and 1) which indicates whether or not that word falls into the negative spectrum of opinion (<0), positive side (>0) or may even indicate that it is objective (somewhere very close to or exactly 0). Most new-generation sentiment dictionaries/lexicons also have incorporated context-sensitive polarities for words which can mean something totally different in different contexts. Take for example the word *good.* We are tempted to say that it is a strictly positive word, however, *a good* can simply refer to a commodity or article. This is where POS kicks in! The POS of a word sometimes can be used to identify context. The most complete sentiment lexicon is SentiWordNet. As seen in [4], SentiWordNet was generated using a semi-supervised method with the help of seed words and it is the lexicon we used in our approach.

3.2. Supervised, Unsupervised and in-between

There are two main types of approaches to opinion mining based on the underlying algorithms and tactic. We will discuss these two types and a third variation which is but a sort of combination between the two. Interesting results were obtained by both each having its strengths and weaknesses. Supervised approaches usually deal with language and domain dependent corpora. This is because behind every supervised approach, we find a trained classifier. This classifier has to be trained on a specific text with specific attributes. When given another text from a different domain, it's only obvious that it will yield bad results as different models are hidden in the text, models which cannot be detected by the classifier which was initially trained on a different type of data which had some other models hidden in it. As we can see in [5] and [6], supervised approaches lead to great results in terms of accuracy, recall and precision, however, with some costs! Unsupervised methods however are a different story.

Unsupervised algorithms require no training data to perform well. They find hidden models from the given input without any additional help and thus we can clearly see that they are both more difficult to implement and conceptualize and are bound to give worse results. Or are they? Domain independency or language independency is what differentiates unsupervised approaches to supervised ones. These are terribly valuable attributes which are not overlook lightly (and nor should they be) by anyone. Imagine a language independent sentiment analyzer which works just as well for Chinese and Greek as it does for English! Imagine a domain independent opinion extractor system which does not need any training on specific data but which can pull out opinions from tweets just as well as it does from product review data. We will analyze notable approaches to further investigate these advantages in the following subchapters.

The last type of underlying algorithm is the *semi-supervised* one. Supervised classifiers require training data whereas unsupervised methods require no data at all. Semi-supervised approaches on the other hand, require some auxiliary *pylons* or elements which help it build its models. Now we're not talking about training data but rather *small* elements like seed words or other bootstrap mechanism. Approaches will be discussed in the subchapters below.

3.2.1. Supervised Learning – not width, but length

Supervised machine learning is the task of deducing algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features, as defined in [7]. In the same article we find a great overview on what supervised learning is and different algorithms are presented each with its strengths and weaknesses. As mentioned above, supervised learning requires some externally supplied instances in order to produce general hypotheses. In the case of opinion mining, those externally supplied instances would actually be some text files containing annotated text which will be passed to a Naïve Bayes, Decision Tree etc. classifier. A great summary of almost all types of classifiers can be found in [8]. Here, both supervised and unsupervised methods are compared with more emphasis on the supervised part. As a conclusion from [8], the author says that sentiment classifiers are severely dependent on domain or topics. The author states that no classification model outperforms the other. Another great comparison between classifiers can be found in [9] where SVM, Naïve Bayes, Random Forest and Decision Tree based classifiers are being compared on the same dataset with interesting results.

An interesting approach can be found in [10] where, while trying to minimize the effort to create linguistic resources for more languages, the authors proposed a sentiment analysis tool for a high number of languages. Unlike other approaches, they employ a fully-formed machine translation system. They also study the difference which translation has on the sentiment classification performance. Their comparative results show that indeed, a machine translation technique can be reliable enough for multilingual sentiment analysis and also shows which are the main characteristics of the data for such approaches to be successfully employed.

As stated in the previous chapters, social networking (*Twitter, Facebook* etc.) are huge containers of valuable opinion on diverse topics ranging from politics and world events to movies and products. In [11] we find a novel approach of exploiting the user influence factor in order to predict the outcome of election results. The proposed supervised techniques is based on extracting opinion using indirect features of Twitter data based on SVM, Naïve Bayes, and Artificial Neural Networks (ANN). They combine Principle Component Analysis (PCA) with SVM in an attempt to perform dimensionality reduction. The paper also shows two different case studies of entirely different social scenarios, the 2012 US Presidential Elections and the 2013 Karnataka Assembly Elections. They conclude with the conditions under which Twitter may fail or succeed in predicting the outcome of elections. Some very powerful results were obtained especially

for the 2012 US Presidential Elections their SVM classifier having an astonishing 88% prediction accuracy.

3.2.2. Unsupervised Learning – more girth, not height

A great definition to *unsupervised learning* is given in [12] and it goes as follows: Consider a machine (or living organism) which receives some sequence of inputs x_1, x_2, x_3, etc., where x_t is the sensory input at time t. This input, which we will often call the data, could correspond to an image on the retina, the pixels in a camera, or a sound waveform. It could also correspond to less obviously sensory data, for example the words in a news story, or the list of items in a supermarket shopping basket. In *unsupervised learning* the machine simply receives input x_1, x_2, x_3 etc. but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However it is possible to develop a formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs etc. In some ways, unsupervised learning can be seen as a *pattern finding technique* which tries to find patterns in the data above and beyond what would be considered pure unstructured noise. Lengthy definition and somewhat unclear, but think of an unsupervised learning as a child who tries to learn everything by himself. He has no tutors and no examples of solved exercises however, he is intellectually gifted. Based on a few predefined rules, our child finds new patterns in solving new exercises. There are quite a few approaches to unsupervised sentiment analysis or opinion mining. We will discuss a few of them in the following subchapters.

Lexicon based approaches

A more traditional way of performing unsupervised opinion mining is the *lexicon based* method. A few of them can be found in [13], [14] and [15]. These methods require a sentiment lexicon in order to determine the overall sentiment polarity of a document or phrase, depending on the granularity level. In [16], the author identifies a few difficulties which may be encountered when using lexicon based approaches in social-media data analysis. First of all, texts in social media are short in length resulting in a lack of sufficient data which can be aggregated in order to calculate the total polarity. Furthermore, *slang* is almost omnipresent with expressions like "*it's so cool!*" or "*hardcore man, you rock!*" which renders most lexicons useless. Even more, this slang is constantly evolving and very difficult to track. Last but not least, context-awareness is terribly difficult to obtain whilst only having a sentiment lexicon. Words are being used with one meaning in one place and with another meaning someplace else. Thus, especially when dealing with social media texts, the lexicon based approaches fall short from quite a few points of view.

However, there have been some successful attempts to lexicon-based sentiment analysis, one of which can be found in [17]. Their Semantic Orientation CALculator (SO-CAL, as they call it) uses dictionaries of words annotated with their semantic orientation (polarity and strength) and incorporates negation and intensification. They use SO-CAL for polarity classification which is the task of assigning either a positive or negative label to a text. Some domain independence is assumed and shown but that is not the main

purpose of the article. A very interesting feature from [17] is the use of *intensifiers*. Intensifiers are words like *very, slightly* etc. which increase or decrease the semantic intensity of neighboring lexical items. They are split into two, *amplifiers* (very, most, etc.) and *downtoners* (slightly, somewhat etc.). Even though *intensifiers* were used before but as linear modifiers (meaning they were assign +1/2/3 or -1/-2/-3 scores and summed up to the lexical neighbor), their approach differed and introduced a percentage score and assigned them to each intensifier. An example is shown in Table 1.

Table 1

Intensifier	Modifier (%)
Slightly	-50
Somewhat	-30
Pretty	-10
Really	+15
Very	+25
Extraordinary	+50
(the) most	+100

Because our intensifiers are implemented using a percentage scale, they are able to fully capture the variety of intensifying words as well as the SO value of the item being modified. This scale can be applied to other parts of speech, given that adjectives, adverbs, and verbs use the same set of intensifiers. Negation however was handled the *normal* way, by simply switching the score to its negative (negating it / multiplying it with -1).

As a conclusion, lexical-based approaches are somewhat popular but can hit many difficulties along the way because of the lexicons' limitations and rapidly evolving slang in the case of social-network texts.

Emotional signals in text

A very, very interesting approach especially when dealing with social media and product reviews is the analysis of emotional signals. In [16] we get a definition of those *emotion signals* as being any information that could be correlated with sentiment polarity of the document or the words in the document. . For example, when communicating in the physical world, it is common for people to supplement vocal interaction with gestures and facial expressions. Similarly, in social media, users develop visual cues that are strongly associated with their emotional states. These cues, known as emoticons (or facial expressions), are widely used to show the emotion that a user's post represents. When the authors use emoticons, they are effectively marking up the text with an emotional state. In this case, an emoticon is considered as an emotional signal. Table 2 shows the list of emoticons used in the article.

Table 2

Positive	:)	:)	:-)	:D	=)
Negative	:(	: (	:-(		

The article shows good classification accuracy and great gain from using these emotional signals. It's a very interesting approach and it offers great insight on the impact which emoticons have on text classification.

Grammar to the rescue

Grammar can be thought of as a common denominator in most types of texts excluding however some very short and slang filled social media ones. Using grammar as such a common feature which is to be present in the analyzed text proved to be a great idea as it offered domain independence from the get-go. Grammar rules are the same in all types of fields from customer reviews of products to tweets and comments *if* and I am to underline this, the person who wrote that review actually cared for some grammar rules and did not just throw in some words there. But starting from with this assumption, great progress was made in the *unsupervised* portion of the opinion mining and sentiment analysis classification field due to using grammar. The first notable contribution would is [18]. Here, the author uses grammar patterns of part of speech (POS) tags in order to extract two-word phrases from reviews. Table 3 shows the POS tags patterns which are to be used in extracting those two-word phrases.

Table 3

First Word	Second Word	Third Word (not extracted)
JJ	NN or NNS	anything
RB, RBR or RBS	JJ	not NN nor NNS
JJ	JJ	not NN nor NNS
NN or NNS	JJ	not NN nor NNS
RB, RBR or RBS	VB, VBD, VBN, VBG	Anything

- JJ = adjective
- RB/RBR/RBS = types of adverbs
- NN/NNS = nouns
- VB/VBD/VBN/VBG = types of verbs

The patterns are explained in more detail in the article. After extracting them, the second step proposed was to estimate the semantic orientation of the extracted phrases, using pointwise mutual information – information retrieval (PMI-IR) algorithm. Equation 1 shows the PMI-IR formula.

$$PMI(word_1, word_2) = \log_2 \frac{p(word_1 \,\&\, word_2)}{p(word_1)p(word_2)}$$

Equation 1

The semantic orientation of a phrase is calculated as seen in Equation 2.

$$SO(phrase) = PMI(phrase, "excellent") - PMI(phrase, "poor")$$

Equation 2

Another great approach to unsupervised sentiment analysis using grammar can be found in [19]. Here, the author uses the known and previously discovered in [20] double *propagation algorithm.* The idea here is slightly different to what [18] used as it uses a more dynamic method which does not involve any static fixed-position requirements for words.

Table 4

Rule	Used	Obtained	Dependencies
1	OW	T	1.1 $RB \xrightarrow{MR-Rel} NN$ 1.2 $JJ \xrightarrow{MR-Rel} NN$ 1.3 $RB \xrightarrow{MR-Rel} PR$
2	OW	T	2.1 $RB \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} NN$ 2.2 $JJ \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} NN$ 2.3 $RB \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} PR$ 2.4 $JJ \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} PR$
3	OW	OW	3.1 $JJ \xrightarrow{Conj-Rel} JJ$ 3.2 $RB \xrightarrow{Conj-Rel} RB$
4	T	T	4.1 $NN \xrightarrow{Conj-Rel} NN$ 4.2 $PR \xrightarrow{Conj-Rel} NN$ 4.3 $NN \xrightarrow{Conj-Rel} PR$
5	OW	OW	5.1 $JJ \xrightarrow{Conj-Rel} X \xrightarrow{Conj-Rel} JJ$ 5.2 $RB \xrightarrow{Conj-Rel} X \xrightarrow{Conj-Rel} RB$
6	T	OW	6.1 $NN \xrightarrow{MR-Rel} RB$ 6.2 $NN \xrightarrow{MR-Rel} JJ$ 6.3 $PR \xrightarrow{MR-Rel} RB$
7	T	OW	7.1 $NN \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} JJ$ 7.2 $NN \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} RB$ 7.3 $PR \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} RB$ 7.4 $PR \xrightarrow{MR-Rel} X \xrightarrow{MR-Rel} JJ$

As sentences can become more and more complex, the chances of having some fixed positions for a noun its associated adjective become slimmer and thus another approach is needed. [19] proposes a rule based system which takes into account the grammatical dependencies between words in sentences. This implies that parse trees (semantic graphs / syntactic trees as they are called in literature) have to be created for each sentence. Table 4 show an example of such rules, similar to the ones in [19] however with a few modifications. Using such rules, the dynamic element of the extraction is restored but problems may still occur. Parsing is not (yet) a perfect process. Parser may behave abnormally when more intricate sentences are involved and their output may be off. A wrong parser output means misbehavior in the rule-based proposed extraction model and will thus lead to bad extraction results.

Overall a more dynamic approach, the last one surpassed the more static one presented in [18] but with its own disadvantages when it comes to implementation complexity and reliability.

3.2.3. Semi-supervised – somewhere in the middle

Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabeled data for training - typically a small amount of labeled data with a large amount of unlabeled data. There are a few assumptions used in semi-supervised learning, like *smoothness*, *cluster* and *manifold*, however their description is not in the scope of this paper.

Even though it was mentioned in the previous subchapter when talking about unsupervised approaches, the one presented in [19] can fall into either of the two (*unsupervised* or *semi-supervised*) categories depending on one thing: *the number of seed words used*. With a very large number of seed words you will be dealing with a semi-supervised approach rather than an unsupervised one, however using just two seed words for example (or one representative for each class in case there are more than two) implies an unsupervised approach as a few words can hardly be considered *semi-supervised*. Another great example can be found in [21] which uses a very complex statistical approach to extracting targets and aspects from source material.

There are many more articles available on the topic of *semi-supervised* opinion extraction which the reader is encouraged to research and read from but which do not fall in the scope of this paper.

4. Analysis and Theoretical Foundation

In this chapter, we focus on the theoretical stand point of our system. As the reader may already know, there are countless difficulties and obstacles to be tackled when dealing with an unsupervised approach to opinion mining and sentiment analysis. Even though supervised approaches usually yield better results we will try and examine the advantages and disadvantages of such approaches in correlation to unsupervised ones.

In the first part we take a quick journey through some basic theoretical notions regarding opinion mining. We will here include some definitions of terms which are usually encountered in the context of opinion mining and / or sentiment analysis. We will also compare the two types of approaches, supervised and unsupervised and as state above will try to examine their weaknesses and strongpoints.

The second part will focus on creating a quick overview look on the proposed system as a whole, briefly mentioning its parts and their purpose. Here, we will briefly talk about each component from a theoretical point of view and will offer references to a more in detail examination of the same components.

The third will consist of an in depth look on a few of these components from a theoretical point of view in order to create a seamless transition towards the implementation chapter which will present the same components however from an implementation point of view. We will also try and give a comparative approach to some procedures which are part of these components and could have been replaced with something else.

4.1. Opinion mining fundamentals

Opinion mining (also known as sentiment analysis) is a branch of data mining which refers to the processes involving natural language processing, computational linguistics and text analysis which are used to identify and extract subjective information from source materials.

There are two main approaches to opinion mining: *supervised* and *unsupervised*. Indeed there are also semi-supervised approaches however they are not to be discussed in this paper as they are simply a hybrid of the two. *Supervised* learning is the branch of machine learning which has the role of deducing a function from labeled training data. This training data is consisted of a set of training examples. Basically, transposing this to our case, the case of opinion mining, supervised opinion mining would involve having a set of data from which an algorithm would have to learn what and how to extract opinions. Now considering the scenario in which the classifier is trained on data which comes from the field of digital photography. It will learn words and phrases which are specific to that field and will disregard (or regard as being useless) everything else. Use the same classifier for a set of data which comes from the field of *music criticism* and it will end up being utterly useless. Now unsupervised learning on the other hand deals with unlabeled data and tries to find some hidden structure in it. Given our context, unsupervised learning would mean that regardless of where that data is coming from, we can extract some features which will be helpful for data which comes from a different place therefore, we get *domain independence*.

Domain independence is key. Having this means that your solution may be applied to text from any context. How this is done will be in the following subchapters.

4.2. Achieving domain independence through grammar

What all texts have in common regardless of their domain or context is grammar. Grammar is present in almost all forms of text and it is thus a common denominator. The underlying structure of the unsupervised approach which was mentioned earlier is grammar. Our approach was to create a grammar–rule based system which will be able to find these hidden models in domain-independent text.

When talking about computational linguistics, one of the first terms that come to mind is *part of speech.* A part of speech (POS, for short) is an element of grammar associated to each word of a sentence which represents the role that word has in that particular sentence. For example, in the sentence *I went home and took a nap*, the word *I* is a pronoun. POS tagging is the process in which the POSs are assigned to each word in a sentence or document. POS tagging will be further detailed and exemplified in the sections below. Context awareness at POS level is also problematic however it will be discussed in the implementation part of this paper.

In structured grammar we can also find relations between words. These relations will be strongly detailed in chapter 4.5 and are an important theoretical stand point of our approach. Along with the above mentioned POS tags, they will end up forming *syntactic trees*. Syntactic trees (or semantic graphs, how they are also found in literature) are structures which are formed from one and one sentence only. They represent the relations between the words in that sentence and are built with the help of their associated POS tags.

Figure 4-1 shows the syntactic tree for the sentence *my cousin saw a large ship*.

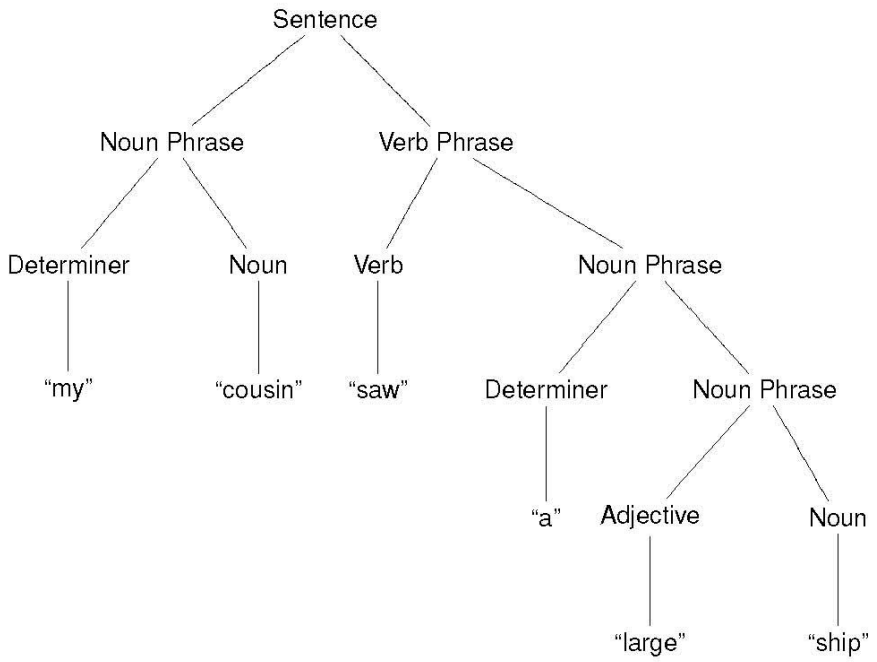

Figure 4-1 Syntactic tree example

The leaves of the tree are always words. The leaves' parents are always their POS tags. The *noun phrases* and *verb phrases* are part of sentences and basically describe the relations between the words.

Syntactic trees are the backbone of our approach. They are structured representations of sentences from which information like opinion or fact can be extracted. Syntactic trees are both the end point and the start point of our process. They are the end point because they are the final step of preprocessing which has to be done on each input source and are a start point for the opinion extraction which, based on grammar rules which are to be discussed a bit later on, will try and extract words (tuples, triplets etc.) from those generated syntactic trees.

Now that we introduced some basic computational linguistic concepts, we are off to uncover the steps needed for every input to go through in order for us generate its corresponding syntactic trees.

4.3. Preprocessing pipeline

In order to obtain these syntactic trees (or, as mentioned before, syntactic graphs as they are also referred to in literature) quite a few steps have to be taken in preparation. The source input material (or corpus) will always be unstructured text. This not imply that the text does not follow grammar rules. Let us remember we are interested in extracting opinion and in extracting it from intelligible sources. Having only a bunch of untagged, unprocessed and plain sentences as input, they have to, in some way, be modeled as to make room for the creation of each sentences' associated syntactic tree.

The preprocessing pipeline consists in 5 steps: *Tokenizing, sentence splitting, POS tagging, lemmatizing* and *parsing*. Figure 4-2 shows an overview diagram of this pipeline.

4.3.1. Tokenizing

It is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens as defined in [22]. In our case, the tokenization process splits the input text into words. For example, the sentence "*The little prince was impressed"* will be split into the following 5 tokens: *the* $^{(1)}$ *little* $^{(2)}$ *prince* $^{(3)}$ *was* $^{(4)}$ *impressed* $^{(5)}$. As stated, tokens are words and tokenization is simply the process of deciding where a word starts and where it ends and extracting them accordingly.

4.3.2. Sentence splitting

At a lower degree of granularity we find the sentence splitter. After identifying and creating tokens (words) from the input text corpus, sentences have to be identified. This block simply splits the input into sentences and thus makes room for the next block which is the *POS Tagger*. The sentence splitter will split the following three sentences like so:

Initial input: *The little prince was impressed. He did not such an answer from the clown. After all, he was just a clown.*

Sentence splitting output:

The little prince was impressed. $^{(1)}$
He did not expect such an answer from the clown. $^{(2)}$
After all, he was just a clown. $^{(3)}$

4.3.3. POS tagging

As defined in [23], in corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context—i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc. Part-of-speech tagging is harder than just having a list of words and their parts of speech, because some words can represent more than one part of speech at different times, and because some parts of speech are complex or unspoken. This is not rare—in natural languages (as opposed to many artificial languages), a large percentage of word-forms are ambiguous. For example, even "dogs", which is usually thought of as just a plural noun, can also be a verb:

The sailor dogs the hatch.

Correct grammatical tagging will reflect that "dogs" is here used as a verb, not as the more common plural noun. Grammatical context is one way to determine this; semantic analysis can also be used to infer that "sailor" and "hatch" implicate "dogs" as 1) in the

nautical context and 2) an action applied to the object "hatch" (in this context, "dogs" is a nautical term meaning "fastens (a watertight door) securely).

4.3.4. Lemmatizing

The best definition for the lemmatisation process can be found in [24] and it states that lemmatisation (or lemmatization) in linguistics, the process of finding a basic word form or a “lexical headword” of a given word. The use of some lemmatization preprocessing has been shown to be especially important for the highly inflected languages in the various tasks of natural language processing, such as keyword spotting or information retrieval.

In many languages, words appear in several inflected forms. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that one might look up in a dictionary, is called the lemma for the word.

Lemmatizing vs stemming

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is $\Rightarrow$ be

car, cars, car's, cars' $\Rightarrow$ car

An interesting comparison is given by [25] and states that *stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. If confronted with the token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source. For these reasons we chose *lemmatizing* over *stemming* as it is much leaner.

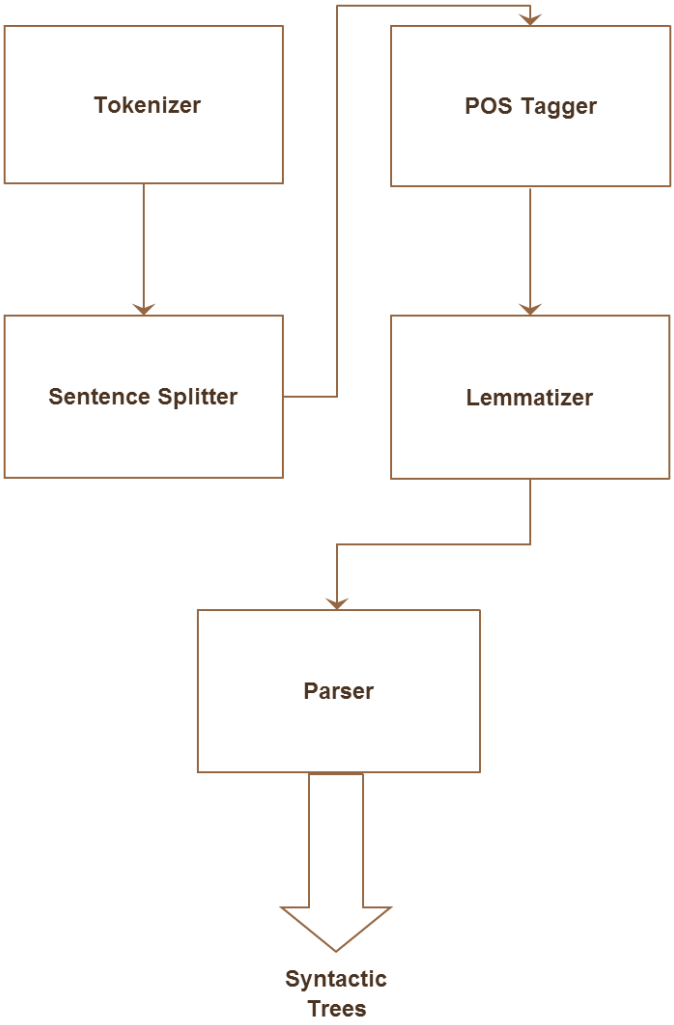

Figure 4-2 Preprocessing pipeline

4.3.5. Parsing

A good definition to parsing can be found in [26] and states that parsing or syntactic analysis is the process of analyzing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. The term parsing comes from Latin pars (orationis), meaning part (of speech). Within computational linguistics the term is used to refer to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information. The last output consists of one syntactic tree per sentence. So if the initial corpus contained 413 sentences which were correctly identified and split by the *sentence splitter*, the output would be 413 syntactic trees, one for each of those sentences.

Thus we explained the theoretical foundation of the preprocessing module of the application. This module stands behind every opinion extraction or polarity aggregation and without it, none of them would be possible. We will further offer some details about the general theoretical architecture of the system in hopes that it will create a clearer image of what is to come. For detailed theoretical knowledge of the following modules please refer [27] or [28].

4.4. Opinion extraction and polarity aggregation fundamentals

The next logical steps of our system are the *opinion extraction* and *polarity aggregation*. The opinion extraction module uses the generated syntactic trees and quite a few grammar rules in order to extract tuples or triplets which are to contain opinion. The rules will not be described in this paper however the grammatical relations which were used for the opinion extraction algorithms will be.

4.4.1. Double propagation – a brief description

At the core of the extraction module lays the double propagation algorithm. A method proposed in [19], it is an iterative algorithm which is based on some bootstrapped seed words and the above mentioned and described syntactic trees.

Figure 4-3 shows the pseudo-code for the double propagation algorithm. The input of the algorithm consists of a collection of syntactic trees, which must be provided at the beginning of each process, from which a collection of tuples is generated as output. The output contains both opinion words and targets. For more details on the double propagation algorithm, please refer to [27].

4.4.2. Polarity Aggregation – a brief description

The assignment of sentiment polarity to the extracted targets and opinion words is the last module of the application. The polarity assignment algorithm is similar to the *double propagation* described above however it does not use any grammatical rules.

As it is still an iterative approach, it needs some sort of initial assignment and therefore the initial seed words are assigned to a polarity at the start of the algorithm. There are quite a few extra rules to the polarity assignment but they are not to be detailed here. A detailed description of the polarity assignment can be found in [28].

```
Input: Seed Word Dictionary {S}, Semantic Graphs {T}
Output: All Features-Opinion Pairs {F}, All Opinion-Feature Pairs {O}
Constant: Objectivity Threshold {Th}
Function:
1.  {O} = {S}
2.  {F1} = Ø, {O1} = Ø
3.  For each graph in T:
4.          if( Extracted features not in {F})
5.                Extract features {F1} along with the opinion words used by using R1, R2 with {O}
6.          endif
7.         if( Extracted opinion words not in {O} and opinion words objectivity < {Th})
8.               Extract opinion words {O1} along with the opinion words used by using R3, R5 with {O}
9.         endif
10. endfor
11. Set {F} = {F} + {F1},  {O} = {O} + {O1}
12. For each graph in T:
13.         if( Extracted features not in {F})
14.               Extract features {F2} along with the features used by using R4 with {F1}
15.         endif
16.        if( Extracted opinion words not in {O} and opinion words objectivity < {Th})
17.              Extract opinion words {O2} along with the features used by using R6, R7 with {F1}
18.        endif
19. endfor
20. Set {F1} = {F1} + {F2},  {O1} = {O1} + {O2}
21. Set {F} = {F} + {F2}, {O} = {O} + {O2}
22. Repeat 2 until size({F1}) = 0 and size({O1}) = 0
```

Figure 4-3 Double propagation pseudo code

4.5. System parameters – what makes it multi-dimensional

As stated in chapter 2, the purpose of this system is to create a multi-dimensional tunable machine for opinion extraction and sentiment analysis, therefore, in this sub-chapter, we will concentrate on the parameters (*knots and bolts*) which give this system its multi-dimensionality property.

a) Target frequency threshold

The first of the two threshold which are used in the system is the *target frequency threshold* (denoted with *TFT*). The *TFT* is a lower limit which if the number of times a target word is extracted falls under, that target is pruned from the extraction and removed. The obvious question which arises is *why is this needed?*

Noise is omnipresent in unstructured text and when dealing with internet reviews on different products, one is bound to encounter quite a lot of it. It's not anything unusual every now and then for someone to mention something completely unrelated to the product which they are reviewing in what they have just written; and that is noise. We are interested in only those aspects and targets which are closely related to the product which is reviewed and because the above mentioned noise happens only ever so often, it can be detected by using a filtering approach. For a digital camera, targets like *picture quality* and *lens* are most certainly going to show up in almost every review hence will generate a high count number. For the same digital camera, an example of noise is a reviewer talking about how the weather was on his way to buying that digital camera (and yes, this happens!) So having a target like *weather* which appears only once in the whole corpus of digital camera reviews raises a few questions. *Why is it there? Can there be only one reviewer talking about an important aspect of the camera?*

Now that we clarified what noise is in the context of target extraction, we can state that having such a threshold is a valuable contribution to the overall result. Not only does it improve target extraction precision but it also removes noise and thus ensures a *cleaner* result.

b) *Polarity threshold*

When talking about an opinion, one must first clarify that an opinion always falls either in the positive or negative polarity spectrum. This is important because objectivity points to *facts* not *opinions*! Opinion mining differs from fact mining because it only takes into account those phrases, sentences or words which are *opinion baring* and thus not objective. Objectivity on the other hand is important when one is trying to find facts and non-subjective information. The difficulty in measuring weather or not a word is either objective or subjective is determining the context in which it appears. For example, the word *short* in the sentence "*The skirts were short*" holds no information on whether or not that was a bad thing or a good thing. In the context of a church, having a *short skirt* would most certainly be a bad thing, however, in the context of a party, *short skirts* might actually be encouraged and thus be a good thing. Now looking at the sentence "*The food is in short supply*" we have a more obvious answer to whether or not the word *short* is baring a positive or negative opinion; context is key!
Objectivity is given by Equation 3.

$$Objectivity = 1 - \left(polarity_{positive} + polarity_{negative}\right)$$

Equation 3

Computing this objectivity of a word or sentence from their positive and negative polarity helps us trim down the objective ones and keep only those which are of value in the context of an opinion mining system. The polarity spectrum ranges from -1 to +1 where 0 represents complete objectivity. The *polarity threshold* creates a distance *delta* from 0 towards both +1 and -1 and thus an interval of type (*-delta, +delta*) in which if the polarity of either a word or sentence falls, it is considered as being objective and pruned. For example, having the *polarity threshold* 0.25 will result in the interval (-0.25, +0.25) and a word with a polarity of 0.14, that word will not be taken into consideration as it is not taken into consideration by this threshold. More details on how this is implemented in section 0.

c) Seed words

Bootstrapping the double propagation algorithm is done with the use of seed words. They are simply opinion words which are initially artificially injected into the algorithm in order for it to have some basis of propagation. The number, type and quality of these seed words are configurable and all of them influence the overall performance of the system. When referring to the *type* of a seed word, we are actually referring to whether or not that seed word is an *adjective* or an *adverb*. As discussed in chapter 4, a great advantage to other OMSs is the possibility of extracting adverbs and adjectives at the same time. The quality of a seed word is a more subjective and somewhat context sensitive issue. For example, in the context of a digital camera review, having seed words like *relieving*, *cold* or *embroiled* may not propagate anything through the algorithm as they belong to a totally different context (acting, maybe?) However, there are some seed

words which are universally available and which were actually used in order to promote this system to a context-independent and thus unsupervised OMS. Two examples of such seed words would be *good* and *bad*, representatives of each class (positive and negative). The number of seed words plays a huge role in the runtime of the system but not so much in the quality of the results. Chapter 6.3.1 shows how using only two seed words and thus being a truly unsupervised approach yields similar if not better results as opposed to using a large number of seed words.

d) Part Of Speech tags

The parts of speech which are involved in the double propagation algorithm are of great importance. Along with the syntactic relations, they form the extraction rules which are detailed in the previous chapter. They are also tunable and play a vital role in the accuracy of extraction. Experiments were done with and without certain POS tags in order to decide which ones are more relevant and which, on the contrary, introduce more noise and should be removed. There are two classes of POS tags which we identified. The *JJ* class which is mainly responsible for containing POS tags usually applied to opinion words (hence the *JJ* which stands for adjective) and the NN class which contains POS tags usually applied to targets (hence the *NN* which stands for nominal phrase).
As of yet and as seen in [27], after plenty of experiments, the following POS tags yielded best results

- **JJ:** *JJ (adjective), JJR (adjective, comparative), JJS (adjective, superlative), RBR (adverb, comparative), RBS (adverb, superlative)*
- **NN:** *NN (noun, singular or mass), NNS(noun, plural), NNP(proper noun, singular), NNPS(proper noun, plural), PRP (personal pronoun)*

e) Syntactic dependency relations

As stated in the subsection above, along with the POS tags, the syntactic dependency relations form the rules based on which the double propagation algorithm extracts both opinion words and targets. They are invaluable for the process and thus were the focus of great attention during development. The relations gathered even more attention than the POS tags as they more in number and are harder to examine without the help of a linguist. Nevertheless, progress was made and with the help of online documentation and lots of test, we were able to figure out which combination of relations offer the best tradeoff regarding the precision and recall of the system. As found in [19], there are two types of relations depending on what they link together. *MR* relations are between a target and an opinion word (and vice versa) whereas *CONJ* relations are only between the same types of entities (target → target, opinion word → opinion word). Each of these two types of relations contain subtypes which in themselves contain specific relations, as shown below.

- **CONJ:**
    - *conj – conjunct (A conjunct is the relation between two elements connected by a coordinating conjunction, such as "and", "or", etc.)*

- **MR:**
  - *mod:*
    - *acomp – adjectival complement (An adjectival complement of a verb is an adjectival phrase which functions as the complement, like an object of the verb).*
    - *amod – adjectival modifier (An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.)*
    - *rcmod – relative clause modifier*
    - *npadvmod – noun phrase as adverbial modifier (This relation captures various places where something syntactically a noun phrase (NP) is used as an adverbial modifier in a sentence. These usages include: (i) a measure phrase, which is the relation between the head of an ADJP/ADVP/PP and the head of a measure phrase modifying the ADJP/ADVP; (ii) noun phrases giving an extent inside a VP which are not objects; (iii) financial constructions involving an adverbial or PP-like NP, notably the following construction $5 a share, where the second NP means "per share"; (iv) floating reflexives; and (v) certain other absolutive NP constructions. A temporal modifier (tmod) is a subclass of npadvmod which is distinguished as a separate relation).*
    - *nn – noun compound modifier (A noun compound modifier of an NP is any noun that serves to modify the head noun).*
    - *advcl – adverbial clause modifier (An adverbial clause modifier of a VP or S is a clause modifying the verb, for example, temporal clause, consequence, conditional clause, purpose clause, etc.)*
  - *subj:*
    - *nsubj – nominal subject (A nominal subject is a noun phrase which is the syntactic subject of a clause. The governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb, which can be an adjective or noun).*
    - *csubj – clausal subject (A clausal subject is a clausal syntactic subject of a clause, i.e., the subject is itself a clause. The governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb. In the two following examples, "what she said" is the subject).*
    - *nsubjpass – passive nominal subject (A passive nominal subject is a noun phrase which is the syntactic subject of a passive clause).*
    - *csubjpass – clausal passive subject (A clausal passive subject is a clausal syntactic subject of a passive clause).*
  - *desc:*
    - *dep – dependent (A clausal passive subject is a clausal syntactic subject of a passive clause).*

Quite a few extra relations were tested but later removed and are worth mentioning here. They all belonged to the ***MR*** subtype. Here follows a list of the removed relations.

- **MR:**
  - *mod:*
    - *mark – marker (A marker is the word introducing a finite clause subordinate to another clause).*
    - *num – numeric modifier (A numeric modifier of a noun is any number phrase that serves to modify the meaning of the noun with a quantity).*
    - *number – element of compound number (An element of compound number is a part of a number phrase or currency amount. We regard a number as a specialized kind of multi-word expression).*
    - *poss – possession modifier (The possession modifier relation holds between the head of an NP and its possessive determiner, or a genitive's complement).*
    - *preconj – preconjunct (A preconjunct is the relation between the head of an NP and a word that appears at the beginning bracketing a conjunction (and puts emphasis on it), such as "either", "both", "neither").*
    - *predet – predeterminer (A predeterminer is the relation between the head of an NP and a word that precedes and modifies the meaning of the NP determiner).*
    - *prep – prepositional modifier (A prepositional modifier of a verb, adjective, or noun is any prepositional phrase that serves to modify the meaning of the verb, adjective, noun, or even another preposition).*
    - *prt – phrasal verb particle (The phrasal verb particle relation identifies a phrasal verb, and holds between the verb and its particle)*
    - *quantmod – quantifier phrase modifier (A quantifier modifier is an element modifying the head of a QP constituent).*
    - *tmod – temporal modifier (A temporal modifier (of a VP, NP, or an ADJP is a bare noun phrase constituent that serves to modify the meaning of the constituent by specifying a time).*
    - *vmod – reduced non-finite verbal modifier (A reduced non-finite verbal modifier is a participial or infinitive form of a verb heading a phrase).*
    - *rcmod – relative clause modifier*
    - *advcl – adverbial clause modifier (An adverbial clause modifier of a VP or S is a clause modifying the verb)*
    - *neg – negation modifier (The negation modifier is the relation between a negation word and the word it modifies.)*

For any further improvements on the system which are to lead it into the *negation detection* realm, the negation modifier will have to be included. However, even though it was not in the scope of this system, it should be implemented for a more complex and complete system.

- *obj:*
  - *dobj – direct object (The direct object of a VP is the noun phrase which is the (accusative) object of the verb).*
  - *iobj – indirect object (The indirect object of a VP is the noun phrase which is the (dative) object of the verb).*

f) Scoring threshold

Although not covered in this paper, the last module of the system, the polarity aggregator can also be configured through this threshold. The design and implementation of the module can be found in [28].

5. Detailed Design and Implementation

The purpose of this chapter is to give some concrete insight on the internal doings of the system and give details regarding the actual implementation of the application. Presenting all the concepts' implementations, this chapter will provide arguments for specific implementation decisions. The first subchapter will give a general overview of the system with some detail on how all the modules they are integrated together and what makes them reusable. The next chapters will focus on the description and in detail view of the first two modules, with emphasis on the first, preprocessing module.

5.1. General overview

Just as a quick reminder, the purpose of the system was to create a multi-dimensional tunable machine which, by having plenty of 'knots' which can be manipulated at will, will be able to find the configuration which yields the best results. Having this purpose in mind, the application had to be made as modular and flexible as possible as to allow more and more of these 'knots' to be added as the system evolved. The application is divided in three big blocks:

i) The Retriever Services / preprocessing block
ii) The Feature-opinion pair identification block
iii) The Polarity aggregation block

An overview diagram of the system can be found in Figure 5-1. Please note that all the cloud-figures are simply the input or output for the three main blocks and they do not represent any functionality but are solely used to show the passing of data from one module to the other.

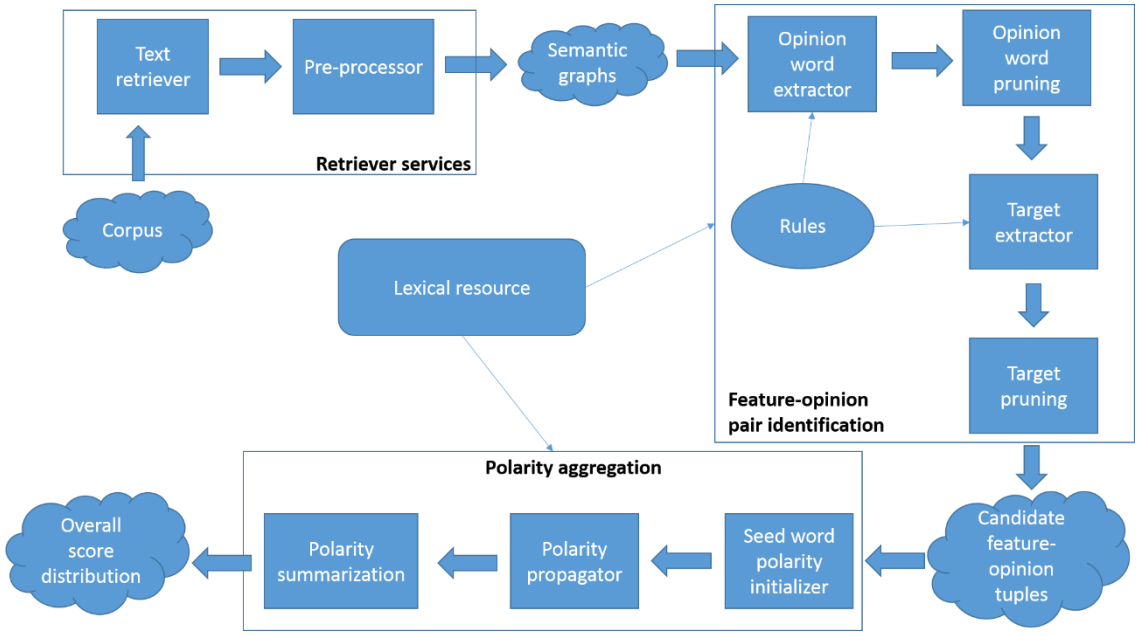

Figure 5-1 General system architecture

We will concentrate heavily on the first block and show some implementation details on the second. The third block is covered exhaustively in [28] and the parts which are not covered from the second will be present in [27].

5.2. Design level view of the Preprocessing and Extraction modules

In this subchapter we will, in more detail, examine the design decisions which led to the current implementation of the system. Both the first and second modules (*preprocessing* and *extraction*) will be thoroughly analyzed, however, with more emphasis on the first as it is the main topic of discussion in this paper.

The architecture of both above mentioned blocks is a modular one. As seen in Figure 5-1, they can be split into a pipeline consisting in different sub-modules. This *pipeline approach* gave great room for expanding the solution in an iterative manner, more and more modules being added as more requirements had to be fulfilled.

The first module, the *preprocessing* one, covers every step that has to be performed from the moment the run button is clicked, to the moment where the double propagation algorithm kicks in. More details on this in section 5.2.1.

From the extraction module only the *target extraction* and *experimentation services* are covered, the rest are to be found in [27].

5.2.1. Preprocessing module design description

The preprocessing module is responsible for every action which has to be performed before the second (extraction) module starts. It covers both fetching, caching and text manipulation all which are to be detailed in this chapter. Section 5.3.1 will cover the technological / implementation related aspects of this module.

In Figure 5-2 we take a more detailed look at the preprocessing module through a top-level architecture. The *seed* and *document* corpora are retrieved through the *input service* which then passes them to the *model creator* section of the system. The role of the *model creator* is to give an underlying structure to the unstructured documents which were just retrieved. There are *wrappers* for both input, output and algorithm data which are the abstractions of either of the three. The reason behind using such wrappers is to provide a structural backbone to everything from input text to output files. Wrappers are similar to DTOs in the context of a web / mobile application in which the frontend user never deals with concrete entities but with some sort of wrappers or *data transfer objects* which are more lightweight. This offers great modularity and extensibility to the system, making, for example, the input sources interchangeable. The input wrappers are then passed to the main preprocessing service which manipulates the text and is responsible for the creation of *syntactic trees*. The preprocessing service will be more thoroughly analyzed in section 5.2.1. The purpose of the cache service is to be a mediator between the next 2 main modules, the *feature-opinion pair identification* and the *polarity aggregation* modules. It first has the role of passing the obtained *semantic trees* to the output service and thus storing them somewhere for future use. It also serves the role of checking whether or not an input file has a corresponding cached preprocessed counterpart. If it indeed has one then it simply retrieves it and passes it to the next big module, if not, it requests it being built by the preprocessing service thus conveying the mediator role.

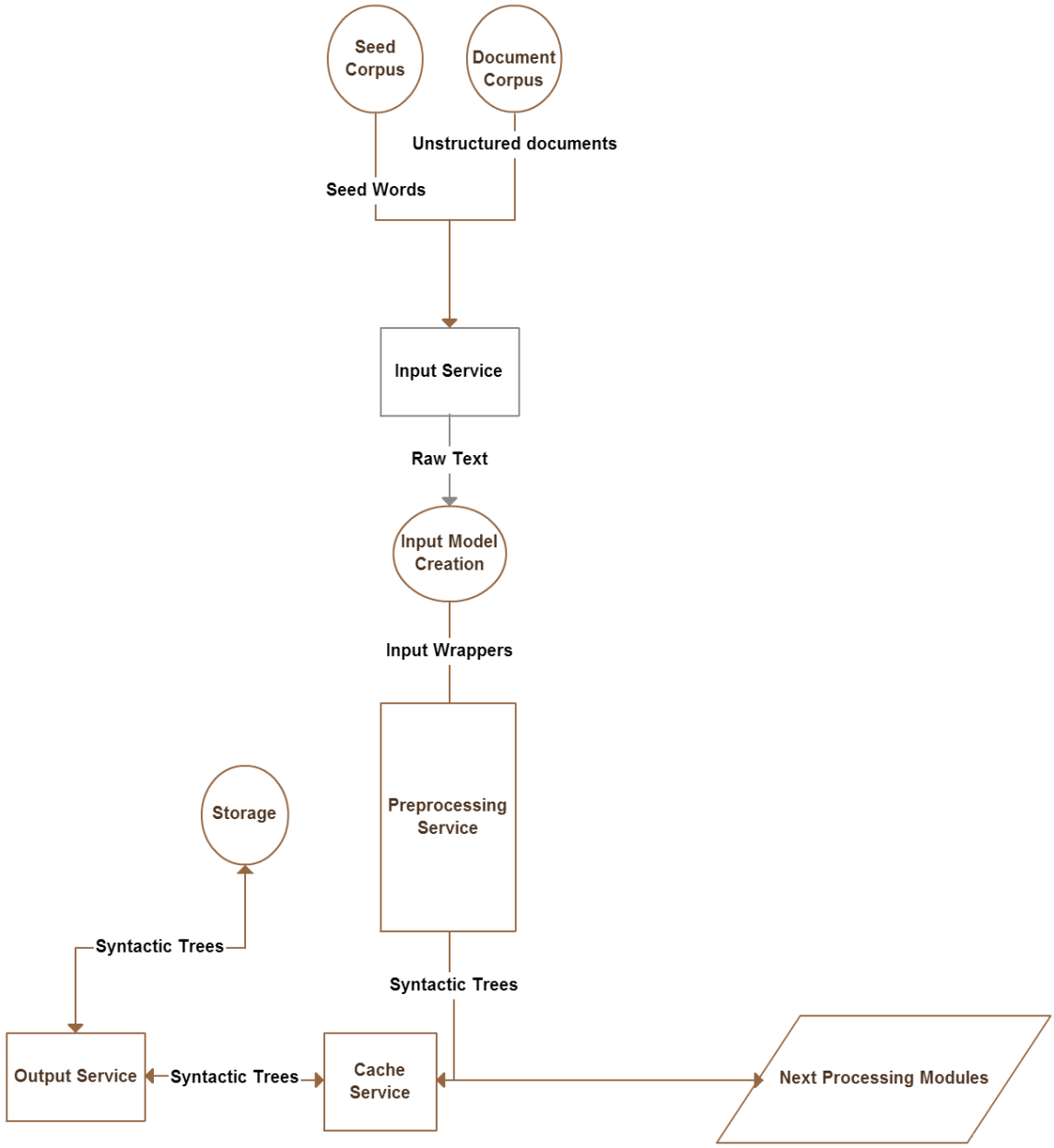

Figure 5-2 Preprocessing module architecture

Input / Output Services

When dealing with any form of text processing, you're bound to need some form of retrieving and storing either from/to the file system or to/from a database. However, rarely will you find natural language processing related corpora in some sort of database format, like *sql*. More often than not, especially benchmark and well known sets of data (reviews, book chapters) which have been used before as test a test corpus will be available as plain text files. This comes with its advantages and disadvantages. It points toward a less structured approach but not without a reason! Data scattered all over the internet is almost never structured data (or at least, not as structured as it could be)

therefore, especially for a system which at its core relies on being a data mini / processing tool, dealing with unstructured (or less-structured) data is actually a requirement.

Preprocessing service

The center point of this module resides in this service. It is, at its core, a pipeline which takes the raw data coming from the input service and creates the *syntactic trees*, the basis on which the feature-opinion extraction and polarity aggregation is built upon. Figure 4-2 shows a more in detail view of the pipeline which resides in the *preprocessing* service. The contribution of each block is also presented in chapter 4.3. A note for the reader: the terms *syntactic trees* and *semantic graphs* are used interchangeably as to avoid repetition. They are however equivalent and should be considered accordingly. Because the theoretical approach was already done in chapter 4.3. We will not further discuss it at design level and will simply skip to the implementation found in section 5.3.1.

Cache Service

The role of the cache service is to speed up the preprocessing and implicitly the overall processing time of the system. It does so by saving the result of the preprocessing in order for them to be quickly accessible if they are needed in the future. For example, take the file *example_file* which contains 210 reviews and 852 sentences in total. Each one of those sentences is a syntactic tree which has to be created by the previously mentioned preprocessing sub-module. This process of creating said syntactic trees is the most time consuming one from the whole module and therefore raised the need to somehow cache the whole process. Files containing these syntactic trees are created for each corresponding input file and sentence. Of course, in the case in which there is no need for running the whole system more than once, this *caching* mechanism is somewhat redundant as the cached files are never used afterwards. However, if this is not the case and the system is to be run more than once on the same input files, section 6.1.2 will show the drastic improvement which caching brings to the systems' total runtime.

5.2.2. Extraction module design description

The center point of this whole system lays in the extraction module. Figure 5-3 presents an overall view of the second the *feature-opinion pair identification (FOPI)* module.

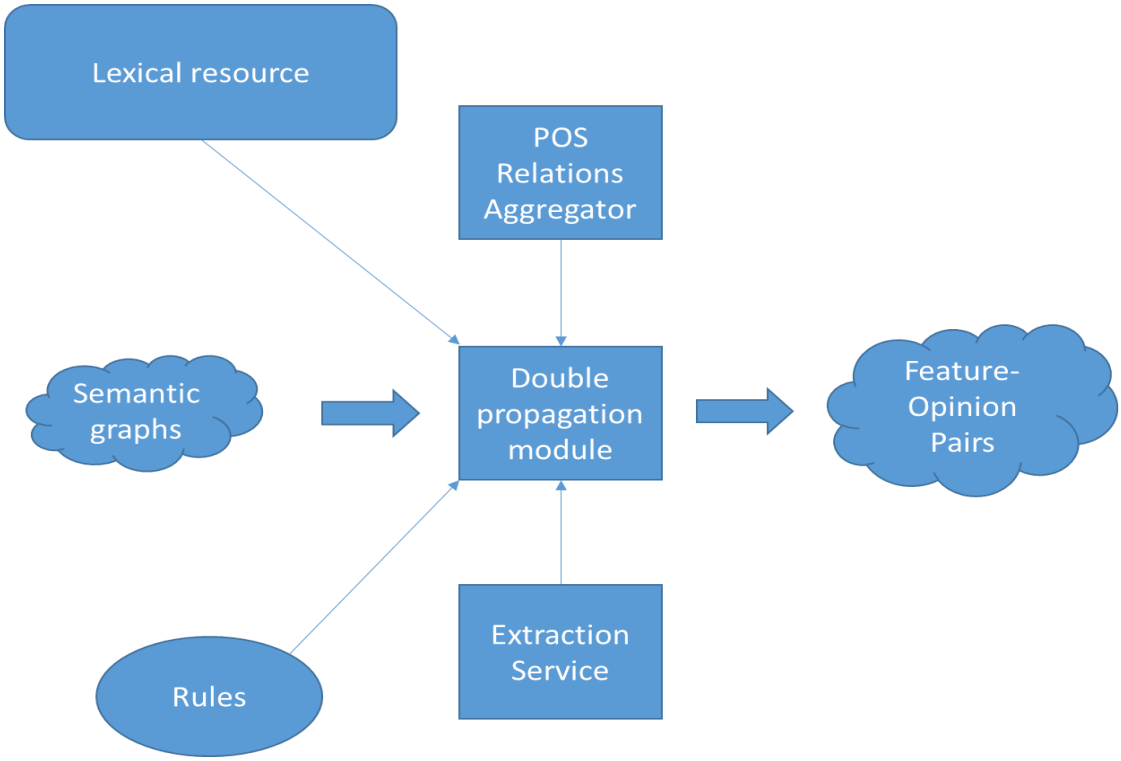

Figure 5-3 Extraction module architecture

Although many of its sub-modules will be documented and explained in [27], two of them will be detailed in this paper: the *extraction* (*target-extraction*) and *experimentation (not shown in* Figure 5-3*)*.

The FOPI module is designed to extract features or opinions from the syntactic trees (or semantic graphs) which come at its input. This module is the next logical step in the opinion mining system (or OMS) and precedes the *polarity aggregation* module.

Target extraction service

The *target extraction* service resides inside the *extraction service* seen in Figure 5-3. Relying on the rules described in chapter 4.5 and more specifically on rules 1, 2 and 4, the purpose of the target extraction service is to retrieve targets from the input syntactic trees using either opinion words or other targets as a propagation catalyst. This is done within the innards of the double propagation algorithm also described in chapter 4.5. The target extraction has two main steps: *pair* and *triplet* extraction. As seen in extraction rules table found in [27], the rules associated with pair extraction are 1 and 4 whereas rule 2 covers the triplet extraction. The equivalent *opinion word* extraction service which covers not the extraction of targets but their counterparts will be presented in [27] and will not be further investigated in this paper.

Let's take an example of each of the two cases (*pair* and *triplet*) in order to gain a better understanding of what was actually needed when implementing the target extraction service.

1) Pair extraction:

Let us consider the example sentence "*the game is good"*. As created by the preprocessing module, the syntactic tree for this sentence looks like Figure 5-4.

```
[
-> game-NN (root)
  -> the-DT (desnit)
  -> good-JJ (dep)
    -> be-VB (cop)
]
```

Figure 5-4 Syntactic tree CoreNLP representation example 1

Now because we are analyzing rule 1, we extract the target (which in this case is *game*) from previously having extracted *good* as an opinion word or perhaps having it as a seed word. Therefore, what we get is a simple *pair* consisting of the words *game* and *good*. The general case which covers this simple pair extraction of words **A** and **B** is shown in Figure 5-5.

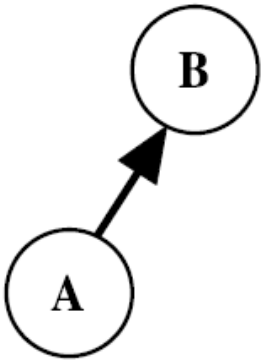

Figure 5-5 Direct dependency

2) Triplet extraction:

Further, let's take a more complete sentence "*the ringtone included is awesome*". As created by the preprocessing module, the syntactic tree for this sentence looks like Figure 5-6.

```
[
-> include-VBP (root)
  -> ringtone-NN (nsubj)
    -> the-DT (det)
  -> awesome-JJ (dep)
    -> be-VB (cop)
]
```

Figure 5-6 Syntactic tree CoreNLP representation example 2

The main point which is to be made here is that, looking at Figure 5-6, we can see that *include* (the lemma of *included*) is the root for both *ringtone* and *awesome*. There is no direct dependency between the target (*ringtone*) and the opinion word (*awesome*) however they are linked through this third *helping* word exemplified and denoted by **H** in Figure 5-7.

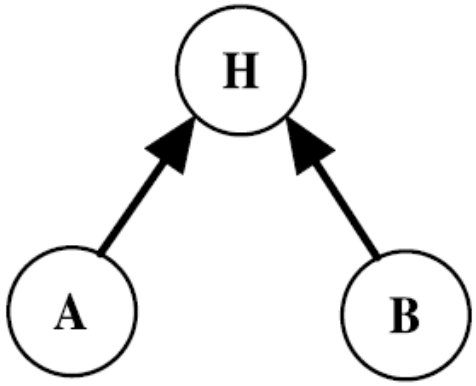

Figure 5-7 Indirect Dependency

As opposed to Figure 5-5, there is an auxiliary *helper* word which in terms creates a *triplet* instead of a simple *pair*. Thus, the extracted triplet will consist of the words *ringtone*, *included* and *awesome*.
More details on how this service is implemented will be discussed in section 5.3.2.

Experimentation parameters

Because they are thoroughly examined from a theoretical standpoint in section 4.5 only their implementation will be further discussed and can be found in section 0.

5.3. Implementation details of the Preprocessing and Extraction modules

In this chapter we will cover some important implementation-specific details of the modules briefly described in the previous sub-chapters. We will cover code decisions and the look and feel of the code architecture and design. Moreover, we will try to analyze the difficulties which were encountered when trying to make the sometimes difficult step of building something more concrete from something more abstract. The transition from abstract concept to code is not always easy but the clearest solution (almost always the easiest one to explain) is usually best. Let us first talk about the domain model as it is the most logical way to start.

Figure 5-8 shows the class diagram of the domain model classes.

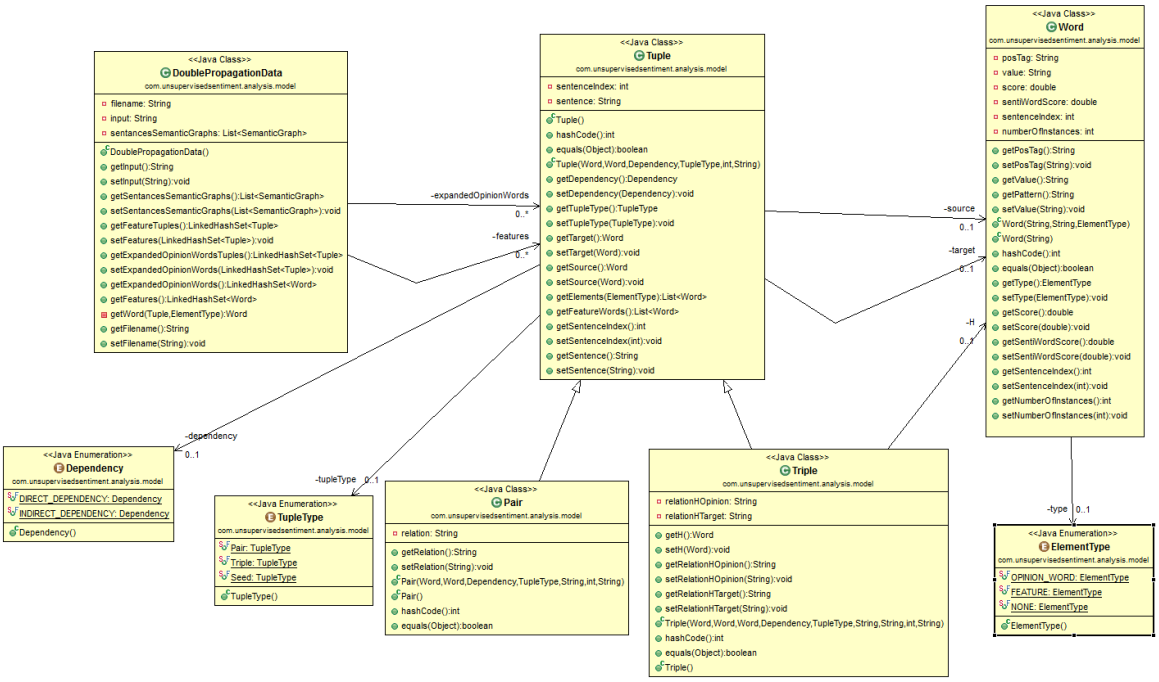

Figure 5-8 Domain model class diagram

The center point of these classes is the *Tuple* class. The *TupleType* enum basically tells us that a tuple can be either a *pair*, a *triple* or a *seed*. The Tuple object can also have a *direct* or *indirect* dependency depending on how it was extracted. Each tuple object has a *source* and a *destination* which are both of type *Word*. Two other classes extend the class Tuple. The class *Pair* has an extra *relation* attribute which basically describes the relation type between the two words inside the tuple. The *Triple* class however has an extra word and thus contains two relations, one between the source and *H*, the new word and the other between the target and *H***.** The *Word* class basically represents a word with an associated part of speech tag (*posTag*) and a value which is the actual value of that word (other attributes are somewhat irrelevant at this point). The last class is *DoublePropagationData* class which has two lists, *expandedWords* and *features* which are all containing tuples. These are main building blocks of the double propagation algorithm described in chapter 4 of this paper. Along with several other irrelevant classes, the above described ones are all located in the *com.unsupervisedsentiment.analysis.model* package.

Now that we have a general overview of the domain model, we can pass to some detailed code – related aspects of the application.

5.3.1. Preprocessing module implementation details

This subchapter will try and create a more in depth description of the preprocessing module of the application. This includes design and implementation specific decisions and of course some code snippets whenever the need arises.

Most of the preprocessing and input / output related classes are located in either the *com.unsupervisedseniment.analysis.IO* or *com.unsupervisedseniment.analysis.preprocess* packages. As a general guideline, for

cleaner code and in order to convey a finer modularity of the application we built *wrapper* classes for every entity which we either read from or wrote to a file. This is similar to the practices used in web applications which create DTOs (*data transfer objects*) which facilitate the transport of information from the frontend to the backend (and vice versa) without actually exposing the database entity to the frontend as this may lead to security related issues, lack of modularity and a poor code quality. There are four main services which are to be discussed, three of them residing in the *IO* package and the forth one in the *preprocess* package.

As an initial design decision, all services were made singleton as there is no use in having more than one instance of each service anywhere in the application. This decision brought a lightweight feel to the services. Everything related to any sort of typical data structures (lists, trees, hash maps etc.) is taken from the Java Collection Framework. The decision to not implement them by hand was quite an obvious one as the ones provided by the JCF are universally used and accepted as being quite efficient. Also, the time which would have been required to implement all these data structures by hand would have been far too great and would have slowed down the development process by a few days, if not weeks.

Input / Output services

The input and output services are both represented by a different class and they both reside in the *IO* package. As mentioned above, they are both singletons. One thing worth mentioning here is that both services are extensively using the *Config* class which is to be described in section 5.3.2 of this paper. The reason for this is that they both need directory or file paths in order to retrieve or write data from or to those specified paths.

The *InputService* class covers all the retrieval methods of the system. As seen in Figure 5-2, it should cover the retrieval of the seed words and document corpus. It has two public methods *getSeedWordsFromFile* and *getTextFromFile* which do this exact thing. The first method extracts seed words from both the positive and negative sample files. It also is responsible for the seed word shuffling which occurs when the number of seed words specified in the *config* file is smaller than the total number of seed words which are found in the two positive or negative files. This is not done by hand but with the use of Java Collection Framework's *Collections.shuffle(<collection>)* method. The second method, *getTextFromFile*, retrieves text from each file path which is given as a chunk however, there are a few helper methods which try and parse any additional metdata which might be present in that file (in a specific format) like author, source etc. Overall a very lightweight and expandable service which can be modified to fit new requirements easily.

The *OutputService* class is responsible for all the file writing which is to be done by the system. It has four public methods which cover all the current output needs of the application. The *writeOuput* method takes as input a list of *OutputWrappers* and writes the tuples or triples which are contained in those wrappers to a file. The *writeEvaluationModels* method is part of the evaluation-cache system which stores the evaluation models to files for fast later access. *WriteToEvaluationMetadata* creates (or appends to) a new csv which will contain all the metadata results of a system run. These results may include (but are not limited to) execution time, values of thresholds used, number of seed words etc. The *writeMapToDetailedReportsFile* creates an iteration-

bound map which basically contains almost all the data which is written in the previous methods' csv, however a bit more detailed. This last method was created in order to see iteration-dependent results and to analyze them accordingly.

All in all, both the *InputService* and the *OuputService* classes are lightweight and can easily be extended and modified at will, leaving room for further improvements.

Preprocessing service

The preprocessing service (not to be confused with the top-level *preprocessing module*) methods are responsible for any text-level work which is applied to the input given to the system. This includes tokenizing, sentence splitting, part of speech tagging, lemmatizing and last but not least, parsing.

All the methods used for these operations are located in the *NLPService* class. We have used the ***Stanford Core Natural Language Processing*** tool (short StanfordNLP) for the above mentioned operations. Stanford CoreNLP provides a set of natural language analysis tools which can take raw English language text input and give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, and indicate which noun phrases refer to the same entities. Stanford CoreNLP is an integrated framework, which make it very easy to apply a bunch of language analysis tools to a piece of text. Starting from plain text, you can run all the tools on it with just two lines of code. Its analyses provide the foundational building blocks for higher-level and domain-specific text understanding applications. CoreNLP has a reported accuracy of over 97% and this was the main reason why we chose to use it over any other tool. Some alternatives include *minipar* for parsing or *porter* tools however, CoreNLP integrates everything from lemmatizing and splitting to parsing in one single tool which makes it a lot easier to use and thus offers more consistency and less work on integrating different tools. At first, the NLP pipeline must be initialized and this is done in the constructor of the *NLPService* class, as shown in Figure 5-9.

```
private NLPService() {
    final Properties props = new Properties();
    props.put("annotators", StanfordCoreNLP.STANFORD_TOKENIZE + ","
            + StanfordCoreNLP.STANFORD_SSPLIT + ","
            + StanfordCoreNLP.STANFORD_POS + ","
            + StanfordCoreNLP.STANFORD_LEMMA + ","
            + StanfordCoreNLP.STANFORD_PARSE);
    coreNlp = new StanfordCoreNLP(props);
}
```

Figure 5-9 NLP Service code snippet 1

There is only public method in this class, all the others being the private innards of the pipeline. *CreateSemanticGraphsListForSentences* is the public method from which the whole pipeline starts. It takes as input a string which is a chuck of text. A series of methods are then called which splits this string into sentences, annotates them, lemmatizes them and POS tags them accordingly. After that, as shown in Figure 5-10, for each object from the *annotatedSentences* list which is a list of sentences created by the above mentioned steps, a *CoreMap* object (*CoreNLP* specific class) is extracted and using its provided *.get* method and passing the

CollapsedCCProcessedDependenciesAnnotation parameter a semantic graph is generated and added to the semantic graphs list.

```
for (final CoreMap sentence : annotatedSentences) {
    final SemanticGraph graph = sentence
            .get(CollapsedCCProcessedDependenciesAnnotation.class);
    semanticGraphs.add(graph);
}
```

Figure 5-10 NLP Service code snippet 2

This list is returned from the method and thus ending the whole pipeline.

Cache service

The *CacheService* class contains all the methods which are responsible for caching. Currently, only two application elements are cached, the syntactic trees (or semantic graphs) and the evaluation models corresponding to each input file.

There are four public methods in this class. The first one, *existsObjectsForFile* simply checks whether or not for an input file we have any cached data. This method was created because cached data is always associated with an input file as in, we do not cache syntactic trees separately but each input file with all its containing sentences' trees are stored in one associated cached file which has its name derived from the input file. So for example, if the input file name is *file_01* then the semantic trees will be stored in *file_01_semantic_trees* and the evaluation models in *file_01_evaluation_models*. The directory in which all these are to be stored is taken from the configuration file which is to be discussed in section 5.3.2.

As only two types of entities which are cached (the semantic graphs and evaluation models), there are two public methods for this class which have equivalent behavior, one for the semantic graphs and one for the evaluation models. The method *getOrCreateSemanticGraphForFile(directory, filename, input)* first checks whether or not that directory contains the specific semantic graphs cached for that specific filename. If it does, it simply returns those cached semantic graphs, however if it does not, it makes a call to the *NLPService* which is described in the previous subchapter and creates the semantic graphs for the provided input argument after which it returns them. The same idea is used for the evaluation models.

Generics are heavily used in the cache service as they provide an elegant way to deal with saving and loading different type of objects to files. Basically, if you are to use an *ObjectBuffer* in order to save or load lists of objects (which implements serializable) from a file, you are most probably going to need to use generics. Creating a generic save or load method will allow you to do these operations on lists which contain any elements without having to casting or other similar procedures. Figure 5-11 shows how a generic method is declared.

```
private <T> List<T> getObjectsFromFile(final String directory, final String filename, final String type)
```

Figure 5-11 Cache service code snippet

When calling such a method, you would have to use a code similar to the one in Figure 5-12. Notice the generic arguments being used at the beginning.

```
return this.<EvaluationModel> getObjectsFromFile(storedEvaluationModelsDirectory, input.getFilename(),
        evaluationModelType);
```

Figure 5-12 Evaluation model code snippet

Now there are 2 alternatives to doing this, both less elegant and one of them terribly redundant. The first *not so bad* alternative would be using the Object superclass for the containing elements of the list. However, this is somewhat of a bad practice and should usually be avoided. The other alternative would have been to use different methods for writing the two lists to file. This is even more cumbersome and it involves quite a hefty amount of code duplication and thus should be avoided at all costs.

The lightweight design and implementation of this module offers great opportunity of extending its functionality and improving it even further. A few ways of extending it will be discussed in section 8.2 in more detail.

5.3.2. Extraction module implementation details

Similar to the preprocessing module, the extraction module is based around several services (or service classes). These services are using the same architectural patterns as the ones described in the previous chapter. They are singletons and are as specific as possible as to reduce the number of lines of code per class. In this chapter we will go through the implementation specific details of the target extraction service and the experimentation parameters. As seen in Figure 5-3, the first part, the target extraction service is part of the bigger sub-module called *extraction service* and is the counterpart of the *opinion word extraction service*. The second subchapter, 0, covers not some procedure in its own right but the parameters which makes this system a multi-dimensional tunable machine with some parameters not being relevant to the extraction module.

Target extraction service

As stated above, the *target extraction service* is part of the *extraction module* in which we can also find the *opinion extraction service.* The *opinion extraction service* will be thoroughly detailed in [27]. In this subchapter, only the target extraction service will be covered and detailed.

All the code related to this service can be found in the *TargetExtractorService* class and the *Helpers* class. The first one can be found in the com.unsupervisedsentiment.analysis.modules.targetextraction package and the *Helpers* class can be found in the com.unsupervisedsentiment.analysis.modules.doublepropagation package. The *Helpers* class is subject to refactoring and may be broken into several classes as to reduce its length and improve code readability. Any such changes will be documented on the GIT commits such that the user might know about where the code is placed.

There are just two methods in the *TargetExtractionService* both representing a subset of the rules described in chapter 4. Figure 5-13 shows a general block diagram of

the extraction mechanism using these rules. We will concentrate on the upper most two rules which correspond to the target extraction.

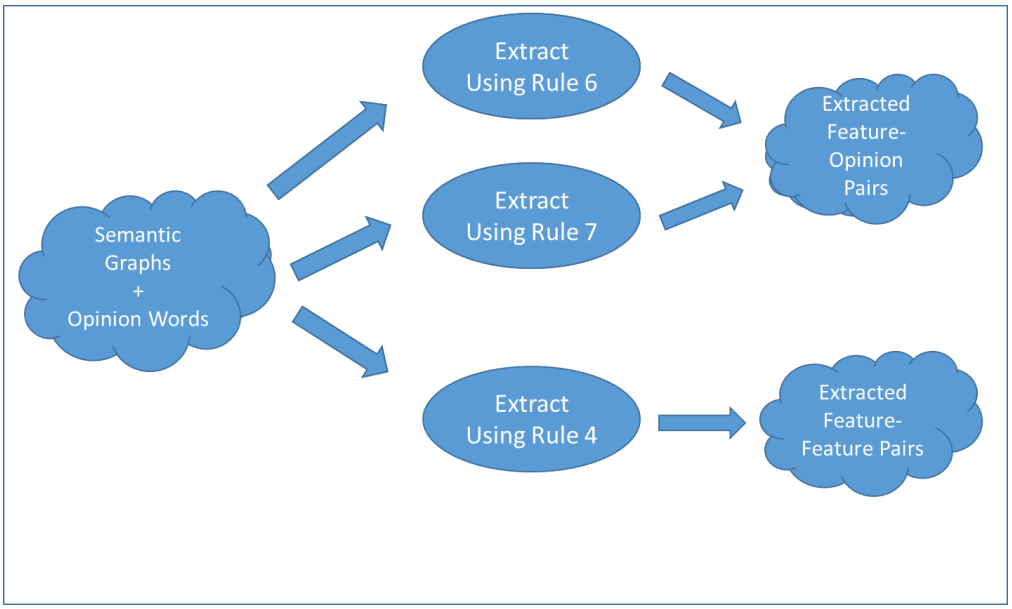

Figure 5-13 Extraction rules diagram

Notice how all the rules require semantic graphs as input. These semantic graphs contain detailed information about a sentence and its' containing words, information like each words' POS tag, the relations between words etc. In order to make rule adding an easier process we have created a method for each of these rules. As seen in Figure 5-14, there are a few extra arguments which are present in the methods' definition. The set of opinion words simply contains all the already-extracted opinion words which might propagate further down the algorithms to ether extract a target (our case) or another opinion word. The set of Tuples (*existionFeatures*) contains already extracted features (or targets) as tuples. That means that they do not come *alone* but those targets are always linked to whatever word helped extract them or vice versa and the relation between them. The *semanticGraphIndex* is only used for evaluation and testing and will not be discussed here.

```
@Override
public Set<Tuple> extractTargetsUsingR1(final SemanticGraph semanticGraph,
        final Set<Word> opinionWords, final Set<Tuple> existingFeatures,
        final int semanticGraphIndex) {
```

Figure 5-14 Extraction service code snippet

The extraction itself (code which is to be found in these methods) is more thoroughly detailed in [27] and will thus not be presented in this paper.

Experimentation parameters

This subchapter will cover the internal representation of all the experimentation parameters which can be tweaked in the system. They will be structured a bit different than chapter 5.2.2 as a few of them may belong to the same class. The first part will cover

the infamous configuration file (config.xml and its java counterpart) whereas the second will tackle the representation of the POS tags and syntactic relations.

Configuration file

Initially, the configuration file was thought of as a *path keeper*, a way to externalize paths such that they will not conflict with one another on different machines. For example, the input files directory on one machine may be located in *D:\Projects\OpinionMining\Input* whereas another machine may have them located in *C:\Code\OpinionMining\Input*. Now dealing with this at code level using any code versioning system is a pain. You will only end up having either conflicts or things which have to be changed each time you do a pull or update. So what we did was to place these paths in a file which is not subject to versioning however it has to be present in the root of the project. Therefore, the *Config.xml* file was created. This file has to be present in the root of the project because it will end up being parsed and its' content transferred to an associated java file at the systems' startup, as we will see a bit later on. There is an existing model of such a *Config.xml* being versioned but in a separate file so that the user can simply copy it in the root and change the paths which are inside. But this is only how it first started and how it was first thought of, as a *path keeper*. Later on, we realized that this file can come in handy with plenty of different other configuration-related settings like the ones we are going to talk here. Figure 5-15 shows a snippet of the contents of the *config.xml* file. Table 5 shows descriptions and examples of all the elements in the configuration file.

```
<evaluationMetadataFile>
    C:\Users\Alex\Desktop\Research\Project\Output\EvaluationMetadata.csv
</evaluationMetadataFile>
<detailedEvaluationMetadataFile>
    C:\Users\Alex\Desktop\Research\Project\Output\DetailedEvaluationMetadata.txt
</detailedEvaluationMetadataFile>
<numberOfSeeds>
    MAX
</numberOfSeeds>
<seedType>
    OW
</seedType>
<polarityThreshold>
    0.01
</polarityThreshold>
<scoringThreshold>
    0.3
</scoringThreshold>
<targetFrequencyThreshold>
    2
</targetFrequencyThreshold>
```

Figure 5-15 Configuration file

Table 5

Element name	Description	Example
inputDirectory	The directory where the input files must be placed	C:\Code\Input
ouputDirectory	The directory where the output files will be created at the end of the process	C\Code\Ouput
SWNPath	The full path (filename included) to the *sentiwordnet* lexicon	C\Code\SWN\swn.txt
positiveSeedWordsFile	The full path (filename included) to where the positive seed words is located	C\Code\Seed\pos.txt
negativeSeedWordsFile	The full path (filename included) to where the negative seed words is located	C\Code\Seed\neg.txt
storedSemanticGraphsDirectory	The folder where the semantic graphs cache files will be placed	C\Code\Cache\SemG
evaluationModelsDirectory	The folder where the evaluation models cache files will be placed	C\Code\Cache\EvalM
evaluaionMetadataFile	The full path (filename included) to the evaluation metadata file	C\Code\Output\Eval
detailedEvaluationMetadataFile	The full path (filename included) to the detailed evaluation metadata file	C\Code\Output\DetEv
numberOfSeeds	The number of seed words to be used from the total present (use any number for a fixed value or MAX for all of them)	2 412 MAX
seedType	The seed word type (not used at the moment)	OW, T
polarityThreshold	The polarity threshold which is to be used when rejecting opinion words based on their objectivity	0.0 – 1.0
scoringThreshold	The scoring threshold used as a *fault tolerance* mechanism for polarity evaluation	0.0 – 1.0
targetFrequencyThreshold	The threshold which is responsible for target frequency pruning, described in chapters 0 and 6.2	0 – *infinity* (recommended max value = 5)

Now to talk a bit about how the information is transferred form the configuration file to its java counterpart. Using the *javax.xml.** packages and a class called *Config* which actually contains all the above mentioned configuration paths, thresholds etc. we created a small chunk of code which runs each time the system is started and automatically parses the *config.xml* and transfers its' data to the java class. Code example in Figure 5-16.

```
JAXBContext jaxbContext;
try {
    jaxbContext = JAXBContext.newInstance(Config.class);
    final Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
    return (Config) jaxbUnmarshaller.unmarshal(new File("config.xml"));
} catch (JAXBException e) {
    e.printStackTrace();
    return null;
}
```

Figure 5-16 Configuration file code snippet

Each time a value is needed, the *Config* class (singleton) instance can be accessed at runtime and it will provide the value from the *.xml* file which was parsed. Thus we covered the *number of seed words*, and all three thresholds. Next, we will discuss the internal representation of the POS tags and syntactic relations.

POS tags and syntactic relations

We knew these tags and relations will be changed often enough to cause problems if not correctly implemented therefore much time was invested to offer this part a very extendable aspect. We decided to use *enums* extensively and thus achieved the goal of creating a lightweight container of POS tags and syntactic relations.

Because we did not want to keep everything in one enum, as mentioned in chapter 5.2.2, we subdivided both the relations and the POS tags into more classes which all had more enums. All POS and syntactic relation related classes can be found in the *com.unsupervisedsentiment.analysis.core.constants.relations* package.

Let us first take a look at the *GeneralPosRelationEnum* class which will be the superclass of all the POS and relations container classes. It is an abstract class containing 2 abstract methods and two concrete ones. The *contains* abstract method will return true or false based on whether or not a string which is given as argument is contained within any of the enums in the implementing class. *The getContainingEnum* abstract method returns the enum in which a given string argument can be found from all the enums in the implementing class. A helper concrete method is *isInEnum* which with the help of generics checks whether or not a value is contained inside an enum (pretty nifty implementation). Figure 5-17 shows an example of implementation of these methods for the *Pos_NNRel* class.

```
@Override
public Class<? extends Enum<?>> getContainingEnum(final String word) {
    if (super.isInEnum(word, NN.class)) {
        return NN.class;
    }
    return null;
}

@Override
public boolean contains(final String word) {
    return super.isInEnum(word, NN.class);
};
```

Figure 5-17 Relations code snippet

6. Testing and Validation

The main goals of the system are to achieve good precision and recall, domain independence through the use of generic seed words and a good runtime with using as few seed words as possible. The evaluation covers a wide spectrum of tests in different directions like target, opinion word extraction performance, polarity assignment performance and the influence of different parameters, thresholds, seed word number of quality and grammatical relations along with different POS usages. This section covers the tests related to the opinion extraction, parameter and threshold, seed word influence and caching. We should keep in mind that, even though the sole purpose of the system was to create a multi-dimensional tunable machine for opinion extraction and sentiment analysis, there are only so many configurations which can be shown in one paper. Many more can be found in [27] and [28].

There are a few conventions which are used thoroughly throughout this chapter and are thus worth mentioning here. We use ***T*** for *targets* and ***OW*** for *opinion words*. The dataset which contains the positive and negative *seed words* has exactly 6789 words, henceforth, for clarity purposes, we shall use ***all seeds*** instead of the more ambiguous number *6789*. Moreover, whenever ***2 seeds*** is encountered, the reader must know that in order to have a representative of each class (positive and negative), these two seed words are *good* and *bad*.

There are four subchapters all targeting specific parameters and use-cases of the system. Once again, they do not cover the whole parameterization range and all the different configuration settings of the system but are only a subset of these.

6.1. Evaluation of Preprocessing and Caching

Let us start with a quick reminder of what *preprocessing* and *caching* steps are in the context of this system. The *preprocessing* step consists of all the actions which, one way or another, manipulate the initial input in order to transform it into a form which is needed for the later (*processing*) step. *Caching* is simply storing the result (in this case, the result of that preprocessing) in order to speed-up any later re-runs of the system. For more information on these, please refer to chapter 4.

6.1.1. Preprocessing

In Figure 6-1 we can see the total preprocessing time for all the available input files. The file corpus is the same as the one used in [19] and it comprises of 5 different files containing hundreds of reviews from different types of products (camera, mp3 player etc.).We also have a comparison between the two cases, one in which the preprocessing result was not cached and the other in which it was. The time difference is roughly at the scale of 1:2000 which basically means that if cached, a file which was preprocessed in roughly 1 minute will have the result of its preprocessing available in 3ms. As stated in the previous chapters, this might not improve overall performance in the rare case in which the user only runs the system once for every file, regardless of the configuration. However, this was not the case when developing the system. We had to

test files hundreds of times with different configurations. The great advantage of having such a low preprocessing time after caching is that it is independent of any configuration change, meaning it will run just as fast and will not require re-caching after a configuration setting is modified.

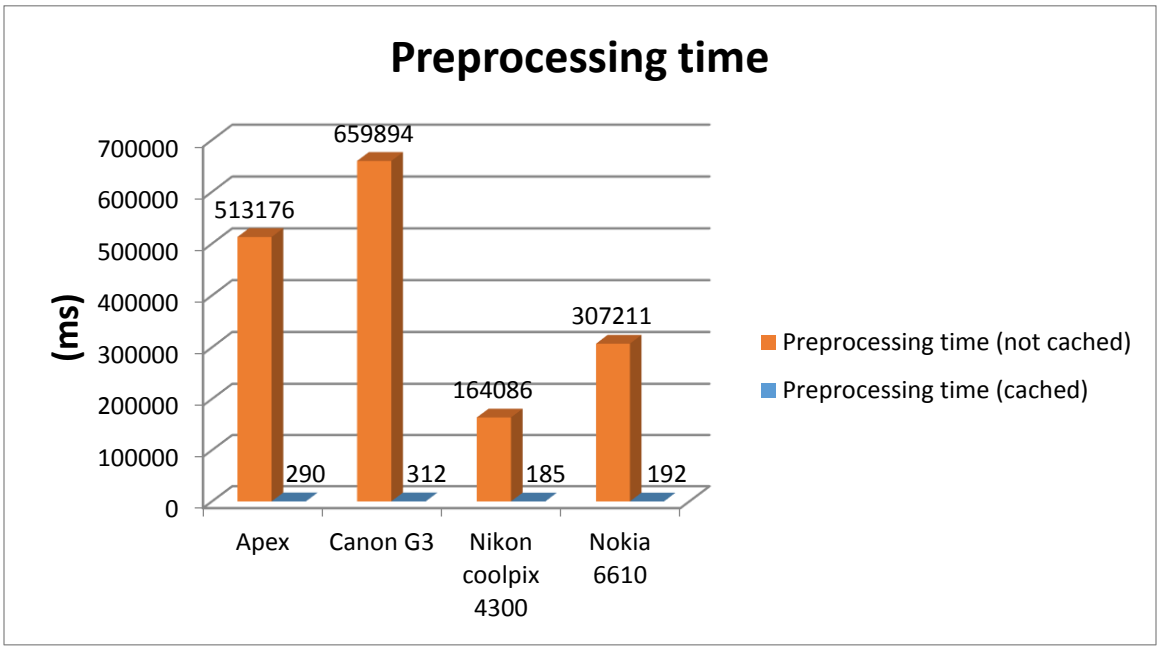

Figure 6-1 Preprocessing time comparison

6.1.2. Caching

Caching can affect both the preprocessing stage of the system but has a great effect on the total runtime as well. The creating of semantic graphs which come as the output from the first stage of the pipeline is very time consuming. The reason is quite simple. As mentioned in the previous chapters, the semantic graph (or *syntactic tree* as it is also referred to in literature) is basically a tree which represents one (and only one) sentence. The vertexes are the words from which that sentence is comprised of and the edges are the syntactic relations between those words (see example in chapter 4). Now considering that on average, an online review has roughly four sentences and a product has approximately 1000 such reviews, that gives us (with some degree of error) 4000 sentences which have to be both POS tagged, lemmatized and from which 4000 syntactic trees must be generated. In order to test our configurations on the same corpus of data we of course had to somehow save those thousands of trees in order to gain precious time by not recreating them over and over again. Figure 6-2 shows the great effect which *caching* had on the overall runtime of the system.

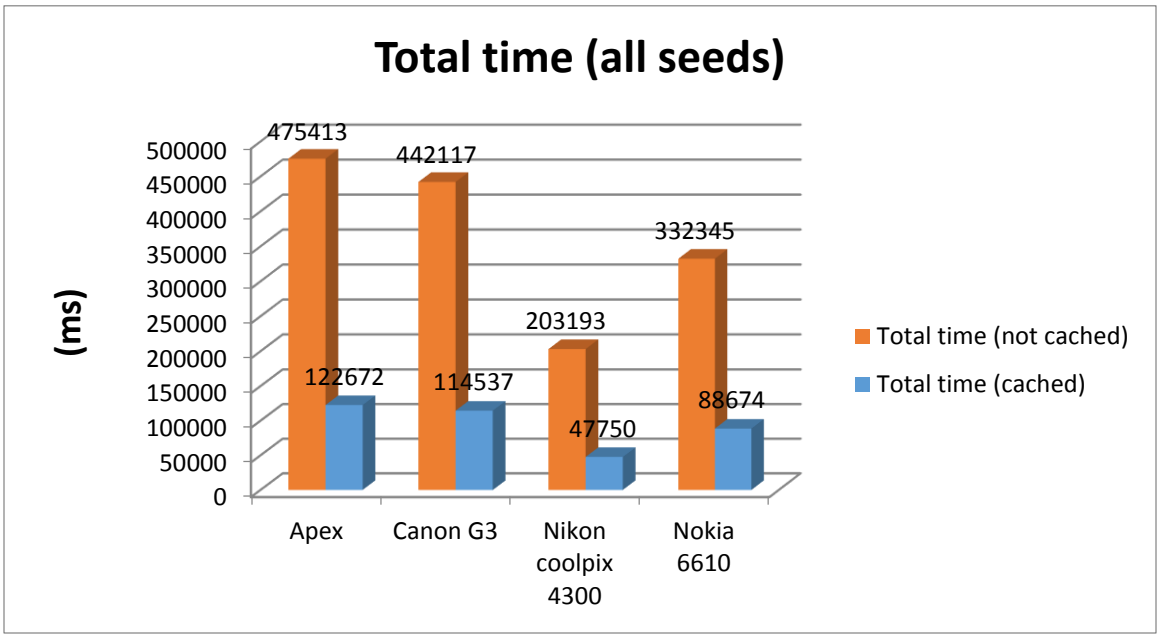

Figure 6-2 Cached influence on runtime

The average total processing time is thus reduced to approximately 15 – 20% of its initial time. Now to put this into perspective, let's assume a total runtime of 20 minutes without the use of caching. That would yield a speedup of roughly 15 minutes if caching was used. Running the system 100 times would mean 1500 minutes (25 hours) wasted on waiting for the process to finish. Instead of it only lasting a total of 500 minutes (under 10 hours) it lasted a cumbersome 2000 minutes (33+ hours).

6.2. Target frequency threshold results

A relevant question which showed up while investigating the problem of opinion extraction, and more precise, the problem of *target* extraction was "*is the number of times a target appears in a corpus relevant to the result?*" and more precisely, "*can we somehow use that as a filter in order to improve our precision?*"

The *target frequency threshold* (denoted as *TFT*) is nothing more than a lower limit which is imposed on target extraction. If a target is extracted less that that *TFT* times, it is removed and is not counted / taken into account when computing the scores. To further create a more detailed picture of how this affects our score, let's take a quick look at a few example sentences which are to be mockly processed. *The camera was perfect. The lens was however a bit scratched but it did not affect the great camera. The simple fact that the scratched lens didn't affect the awesome camera made me buy the whole thing.* The word *camera* appears as three times as a target in the previous sentences. The word *lens* however, only two times. Now for example purposes, let's consider the *TFT* as being 3. In this way, only *camera* is extracted as a target even though *lens* also appears two times. This is not very useful in this scenario in which it should have extracted both *lens* and *camera* as they are both important in the context of a digital camera review. However, in the case of 1000 reviews on a washing machine, the word

lens might accidentally show up only once because the user wrongfully chose the product on which he posted his review. In this scenario, trimming the targets by their frequency is actually useful. The purpose of these experiments was to somehow find a good tradeoff between the different values of this threshold or whether or not it is relevant in any way to our results. As we will soon find, not only is it relevant but it can actually bend the results towards having a much higher precision with the cost of a lower recall.
Opinion word extraction is not relevant in this chapter as it is not at all influenced by the *TFT*. The pruning based on the threshold is done after the double propagation algorithm is executed thus after all the opinion words are extracted but before the evaluation takes place. Target related results are therefore influenced by this threshold.

Let's take a look on how different values of this threshold affect both precision and recall. In Figure 6-3 we have the evolution of the target extraction precision and recall for 2 seeds.

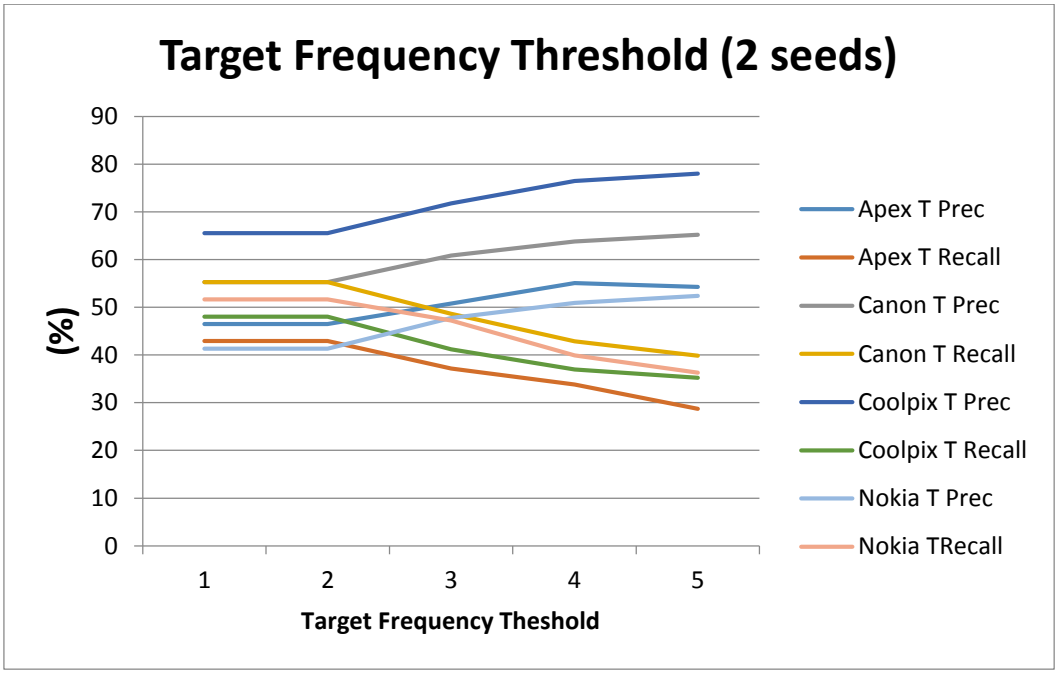

Figure 6-3 Target Frequency Threshold influence for 2 seed words

One very peculiar result comes from the fact that setting the *TFT* to 2 influences neither the precision nor the recall. From this we can conclude that in the *majority of reviews, the important targets appear indeed at least two times*. Of course, this is relative to how the text was annotated but for clarity purposes, we shall not discuss the problem of faulty annotation here. We see a linear growth in precision and as a result, a linear decrease in recall. This is a quite obvious and predictable result. While we extract less and less targets because of that threshold, the number of *good* extracted targets relative to their total number increases, thus increasing the precision.

Now let us take a look on how this threshold affected results when using all the seeds instead of just 2. Figure 6-4 shows exactly this.

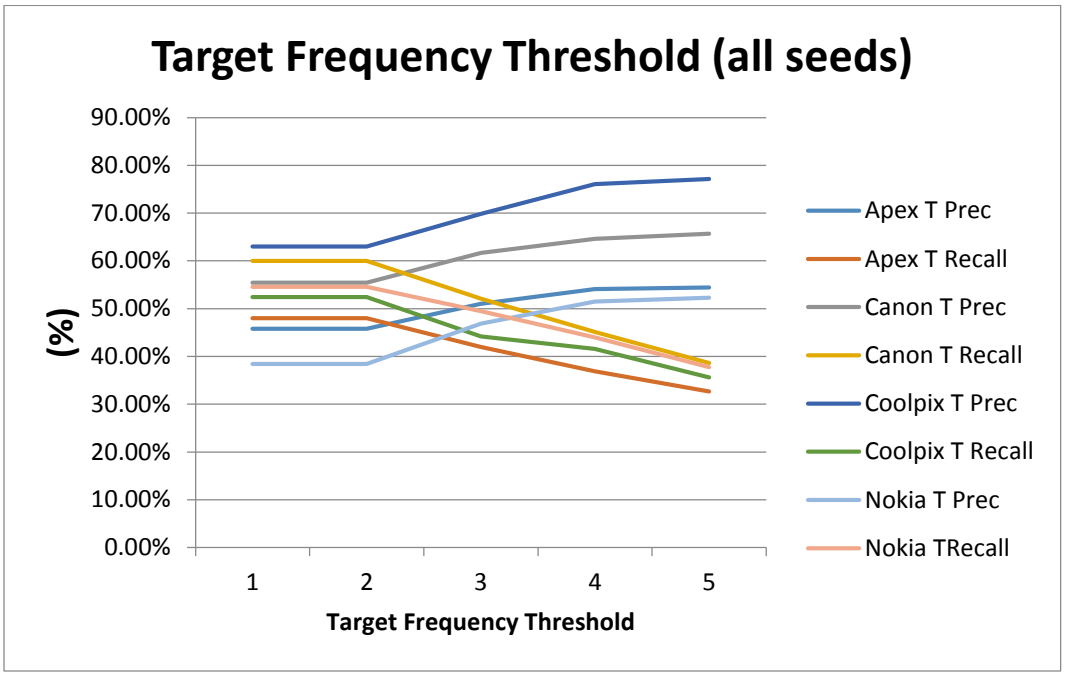

Figure 6-4 Target Frequency Threshold influence for all seed words

As we will see in the next subchapter, using two seed words instead of all of them does decrease the runtime of the system by quite a bit however it comes at a price: the recall will inevitably be a bit lower. This is clearly visible in Figure 6-4 in comparison to Figure 6-3. In the first one, we clearly see that the recall never drops below the 30% mark not even in the case of the *Apex* review set and stays above it for all the rest whereas in the later, the recalls from all review sets drop below 40% mark by quite a little more. The precision is quite similar however a bit lower when using all the seeds instead of just two of them. This is mainly because using so many seeds can (and in most cases will) introduce more noise in the system, noise which if not treated accordingly will end up negatively influencing the results.

6.3. Iteration centered results

As stated in the previous chapters, at the core of the double propagation algorithm we have an iterative process. We thought it useful to have a snapshot of the extraction results at the end of each iteration to see how fast (or slow) the algorithm converges and to analyze the precision and recall relative to that convergence. We were interested in finding out which of the two separately analyzed entities (*targets* or *opinion words*) converge faster and in which case. Also, another point of interest was the number of seeds and how it affects convergence on the two extracted entities. Nevertheless, the

results proved valuable and returned great insight on what is going on after each iteration. In the first subchapter we will concentrate on the results which come from using only two seed words and in the later subchapter we will compare them to the case in which 6789 seed words are used.

6.3.1. Two seed words iteration analysis

In Figure 6-5 we see how the *Opinion Word* extraction converges after each iteration. A very curious finding which resulted from both Figure 6-5 and Figure 6-6 is that even though only two seed words are used, the number of iterations needed for the algorithm to converge is very, very small!

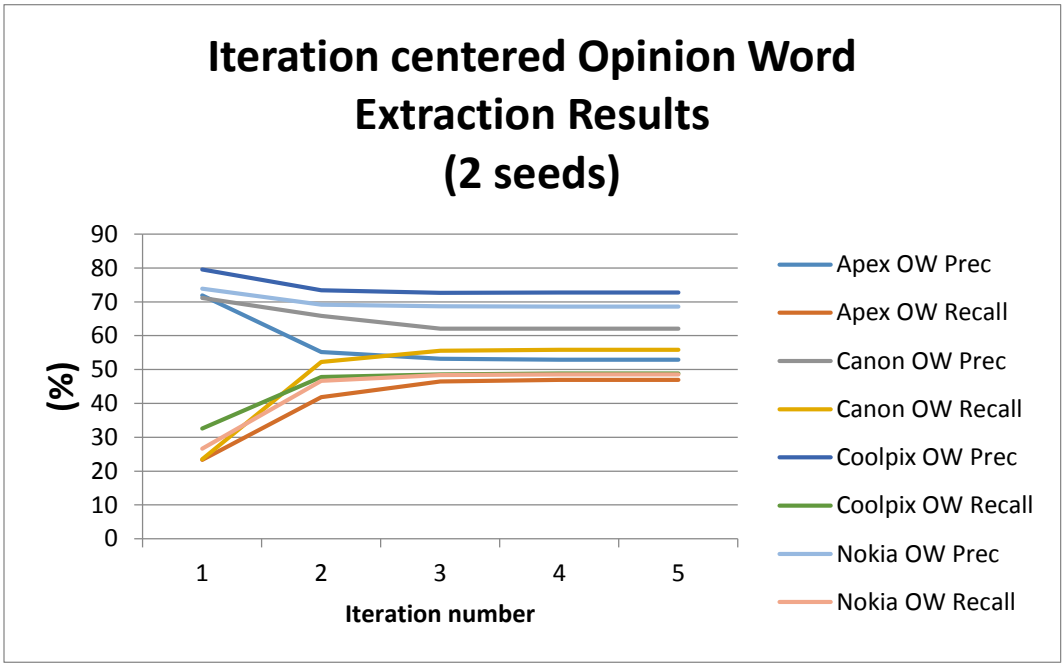

Figure 6-5 Iteration centered results for opinion word extraction (2 seeds)

We expected a rather bigger number however were surprised to find out that only 5 iterations are needed for every file. Peculiar and terribly unintuitive result. Digging deeper into Figure 6-5 we see a few more intuitive results. Starting from a precision ranging from 70 – 80% and from a substantially lower recall, as the iteration number increases, the values tend to normalize: as recall increases, precision decreases but at a much slower rate.

Another surprising result was the abrupt increase in recall after only 2 or 3 iterations, taking into account the fact that only 2 seed words were being used. Again, this unintuitive result gave great insight and helped us spot bottlenecks and further improve extraction.

Figure 6-6 shows the same 2 seeds extraction results but for the targets instead of the opinion words.

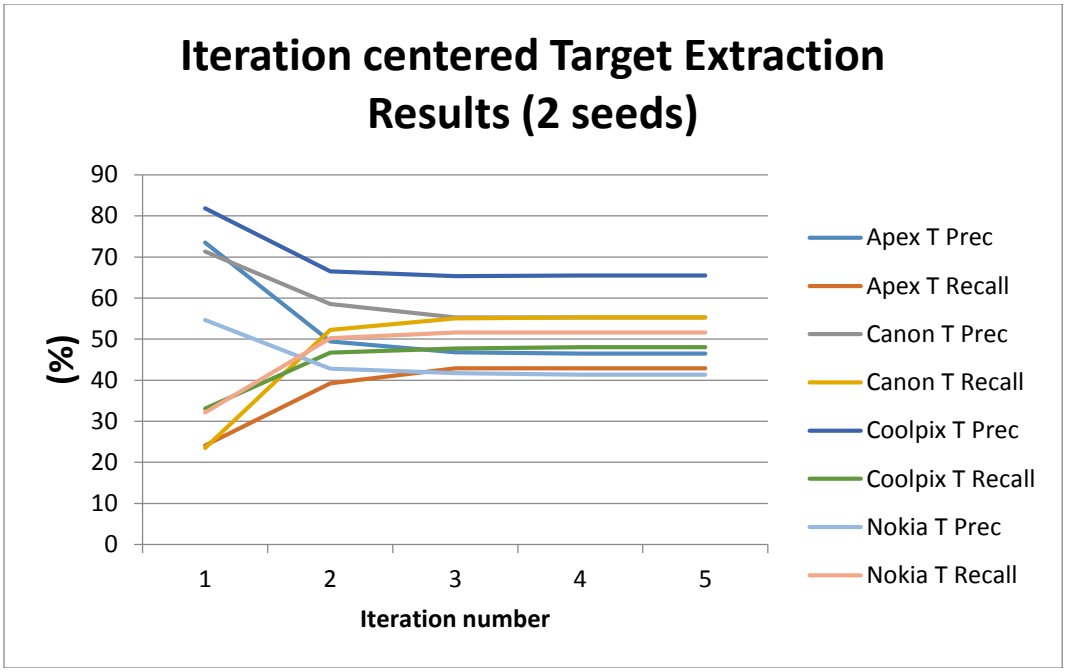

Figure 6-6 Iteration centered results for target extraction (all seeds)

The results are very similar and we observe no major change in convergence from the Opinion Word. Both precision and recall have roughly the same convergence rates.

6.3.2. All seed words iteration analysis

A quick reminder for the reader, the number of seed words used in this scenario is 6789 and the source [29]. As expected, convergence is faster, but disproportional to the number of seed words used compared to the previous case where only two are needed. In Figure 6-8 we see Opinion Word convergence rate relative to the number of iterations. As expected, the number of iterations needed is slightly smaller however, considering the fact that the number of iterations needed for convergence in the case of two seed words was already small, is it a good tradeoff? Comparing the processing time for the two scenarios (2 seed words and all seed words) from Figure 6-7 we can conclude that this is indeed not an advantageous tradeoff. In the case when only two seeds are used, the process takes roughly 30 times less than it takes with using 6789. A more thorough examination on the impact of using less seed words has on the overall performance of the system can be found in [27]. Nevertheless, the non-cached results yield similar runtimes as the cached ones and that table can be found in the Appendix 1.

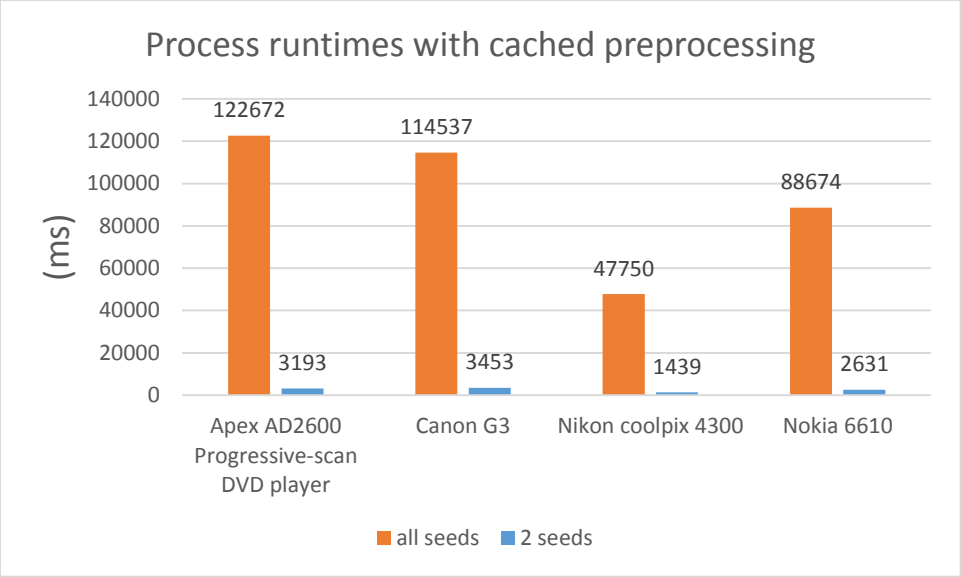

Figure 6-7 Process runtimes with cached preprocessing

Examining Figure 6-8, we can see some files requiring only three iterations for convergence but the majority of them needing at least 4. This is one less than what we have found by using two seed words, as seen in Figure 6-5 and Figure 6-6.

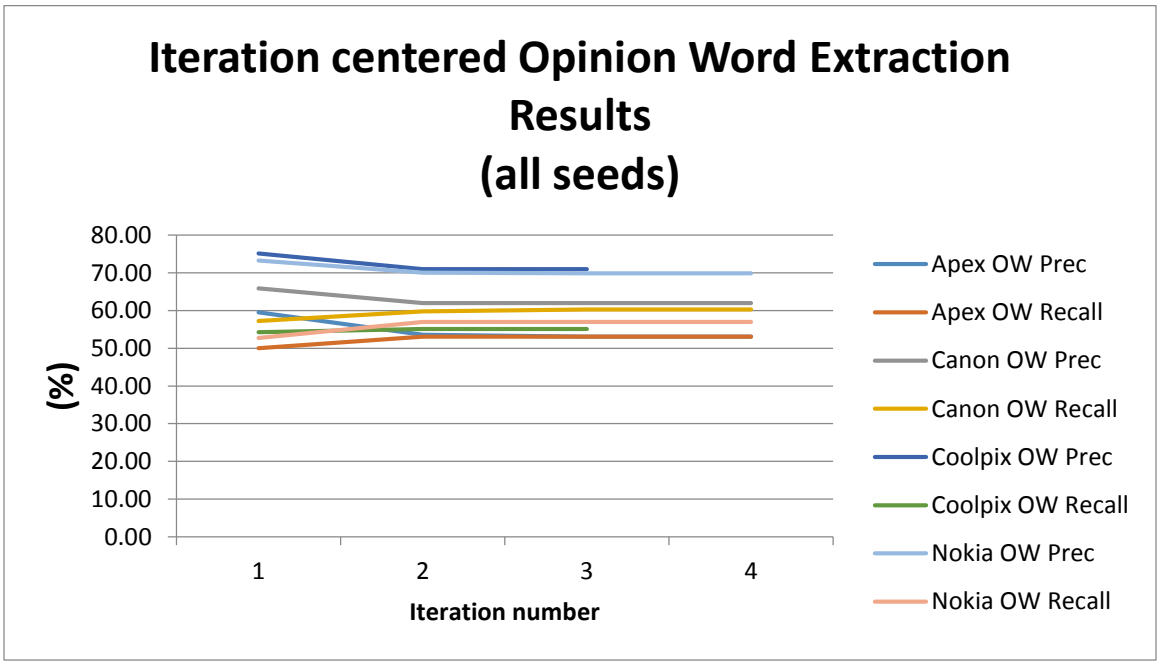

Figure 6-8 Iteration centered results for opinion word extraction (all seeds)

Convergence from the Opinion Word extraction results is very similar to the Target extraction results. There is no noticeable difference between the last 3 iterations in each case as shown in both Figure 6-8 and Figure 6-9.

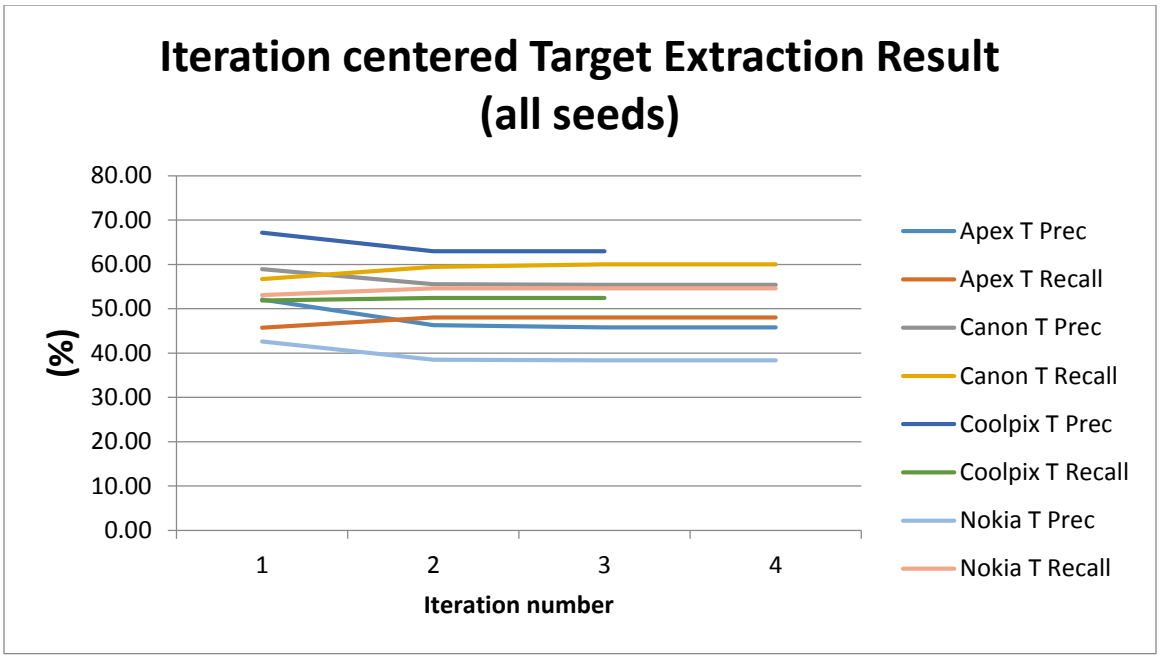

Figure 6-9 Iteration centered results for opinion word extraction (all seeds)

7. User's manual

In this chapter, a step-by-step guide to installing both the application and all of its prerequisites will be presented. It starts off with a short description of how the system was thought of followed by a guide to installing it and its requirements and a main success flow.

7.1. Generalities

The system was from the get-go designed with *Java* in mind. We decided to use this over any other language mainly because of its ease of integration with other tools which we have also used thoroughly throughout the development process. GIT played a key part in this process as it was our code versioning system of choice. We chose Maven as a dependency management tool in order to keep a *portable* feel to the application. As an IDE, we chose Eclipse as it's free and offers great support in integrating everything seamlessly.

7.2. Prerequisites

This subchapter will present all the things which have to be installed or simply downloaded in order to run the system. This will include the Java JDK and Eclipse IDE.

7.2.1. Java JDK

Even though most computers have the Java Runtime Environment already installed, it's important to download and install the Java Development Kit (JDK) instead. This is required for compiling and successfully running the application from the Eclipse IDE.

The download link can be found here (or in the Appendix for the hard copy of this paper). You are there required to choose the corresponding operating system (usually Windows 32/64). After successfully downloading and installing the Java JDK, we can move on to downloading the Eclipse IDE.

7.2.2. Eclipse IDE

The IDE used for writing, compiling and running the code/application is Eclipse. It can be fully integrated with Maven and GIT and it offers great intellisense for java and is overall viewed as being a very power IDE. It can be downloaded from here. There are plenty of packages to choose from, each one being slightly different however the differences are irrelevant to our needs so any of them can be chosen.

7.3. Additional requirements

In this subchapter we will go through all the additional settings which have to be made in order for the system to function correctly. These will include IDE settings, importing the project and GIT cloning.

7.3.1. GIT and GIT Cloning

For the whole development process the code versioning system used was GIT. The repository is public thus it can be cloned without any additional permissions. Of course, for pushing or branching of any sort which is not to be done locally, permissions from our side have to be granted. Please contact us for any such permissions.

First of all, one has to download and install GIT. The link can be found here. After this, even though the GIT bash can be used to pull the repository, a more elegant and intuitive way will be presented by using Tortoise GIT. Tortoise GIT is an extension to GIT which offers a more "*click and do*" user friendly approach to operating the GIT commands. It's an alternative to writing bash commands which can sometimes get tricky. The download link can be found here.

After installing these two in this exact succession, when right-clicking anywhere in the file system, one should get additional menu items as seen in Figure 7-1.

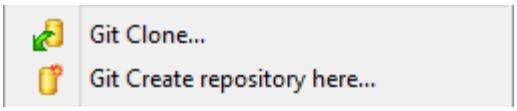

Figure 7-1 Git example 1

Clicking the first item, *Git Clone…* will result in a window very similar to the one in Figure 7-2. Simply copy the following text in the *URL* field and click ok.

https://github.com/dariussuciu/unsupervised-sentiment-analysis.git

A slight warning, a folder will be created inside the folder in which the *right click* was made.

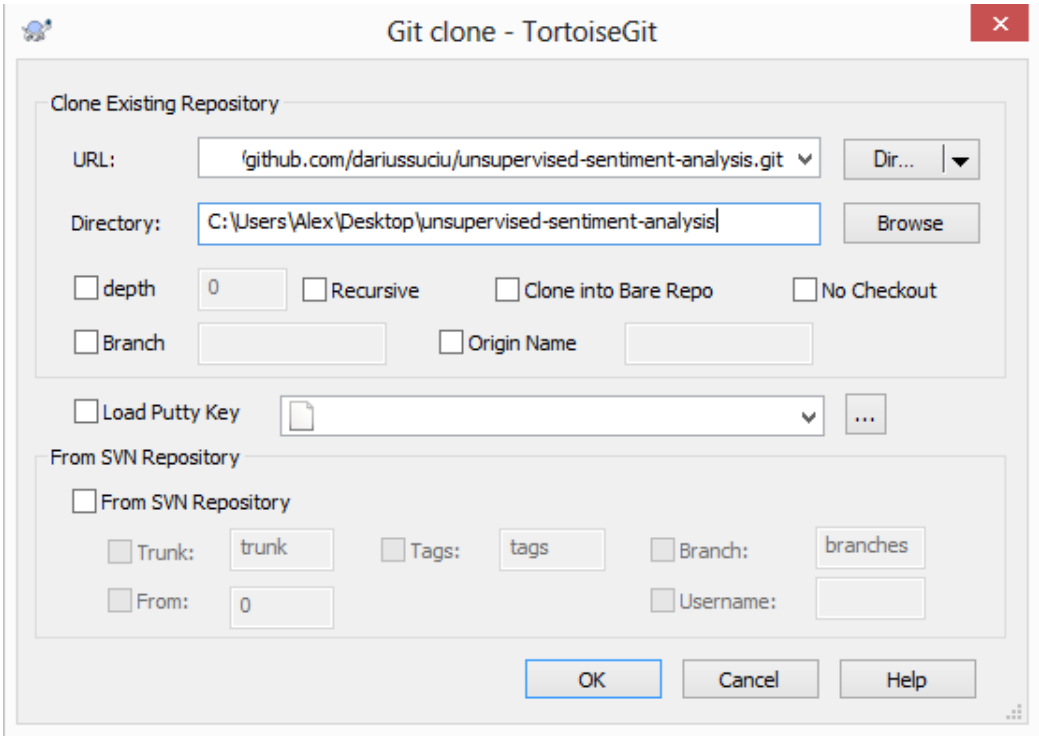

Figure 7-2 Git example 2

7.3.2. Maven in Eclipse

Not all versions of Eclipse come with Maven preinstalled therefore a short tutorial might come in handy. After starting Eclipse and arriving at the (most probably) new and empty Workspace, go to *Help* → *Install new software…* in the above menu-bar. In the *Work with* field from the dialog which appears, copy paste the following: http://download.eclipse.org/technology/m2e/releases. After checking the newly appeared checkbox, the dialog should look like in Figure 7-3.

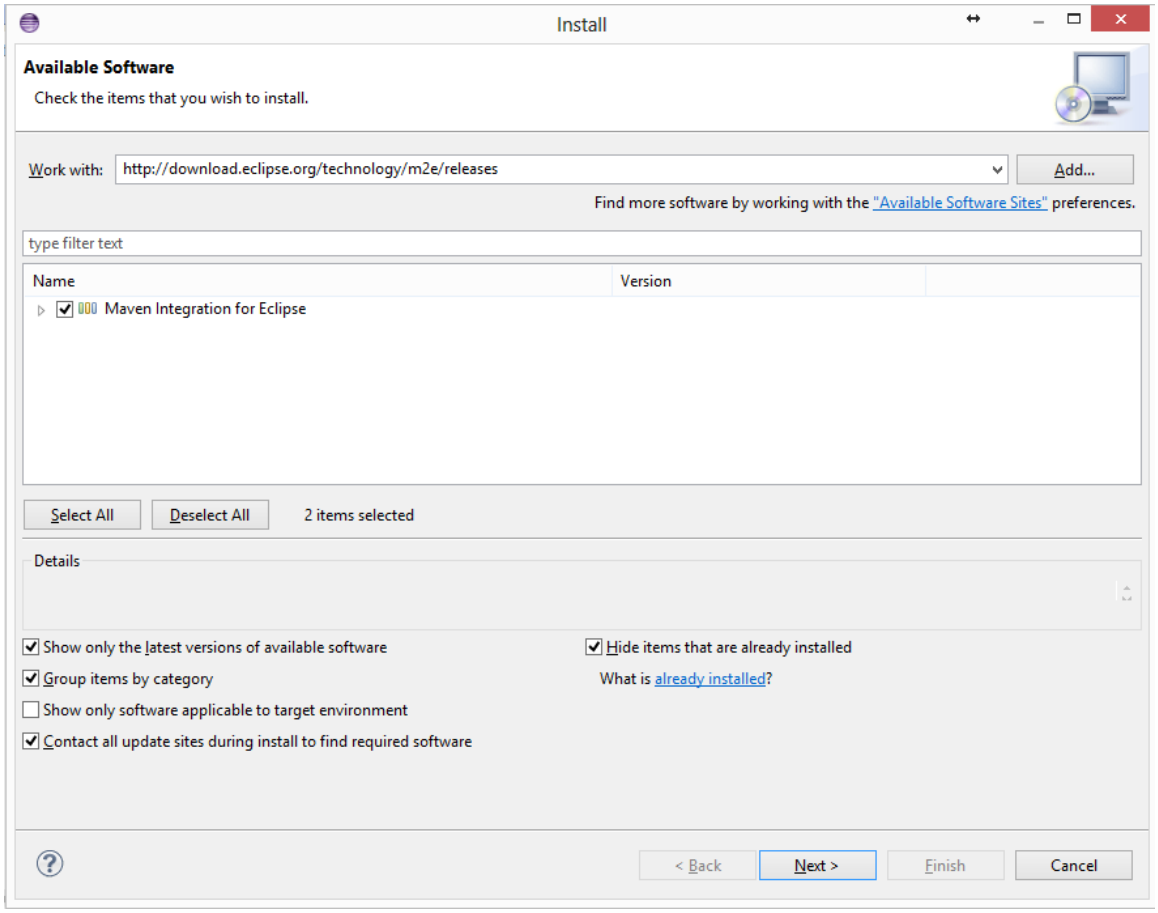

Figure 7-3 Maven example 1

Clicking Next, Next, accepting all the terms and conditions etc. will eventually install the Maven plugin for Eclipse.

7.3.3. Importing the project

Simple yet necessary operation. Right click anywhere in the left package-explorer view in Eclipse and click on *Import…*. Select the Maven dropdown list, existing Maven Projects and in the *Root Directory* field browse to the folder where the GIT Cloning was done. The dialog should appear as in Figure 7-4.

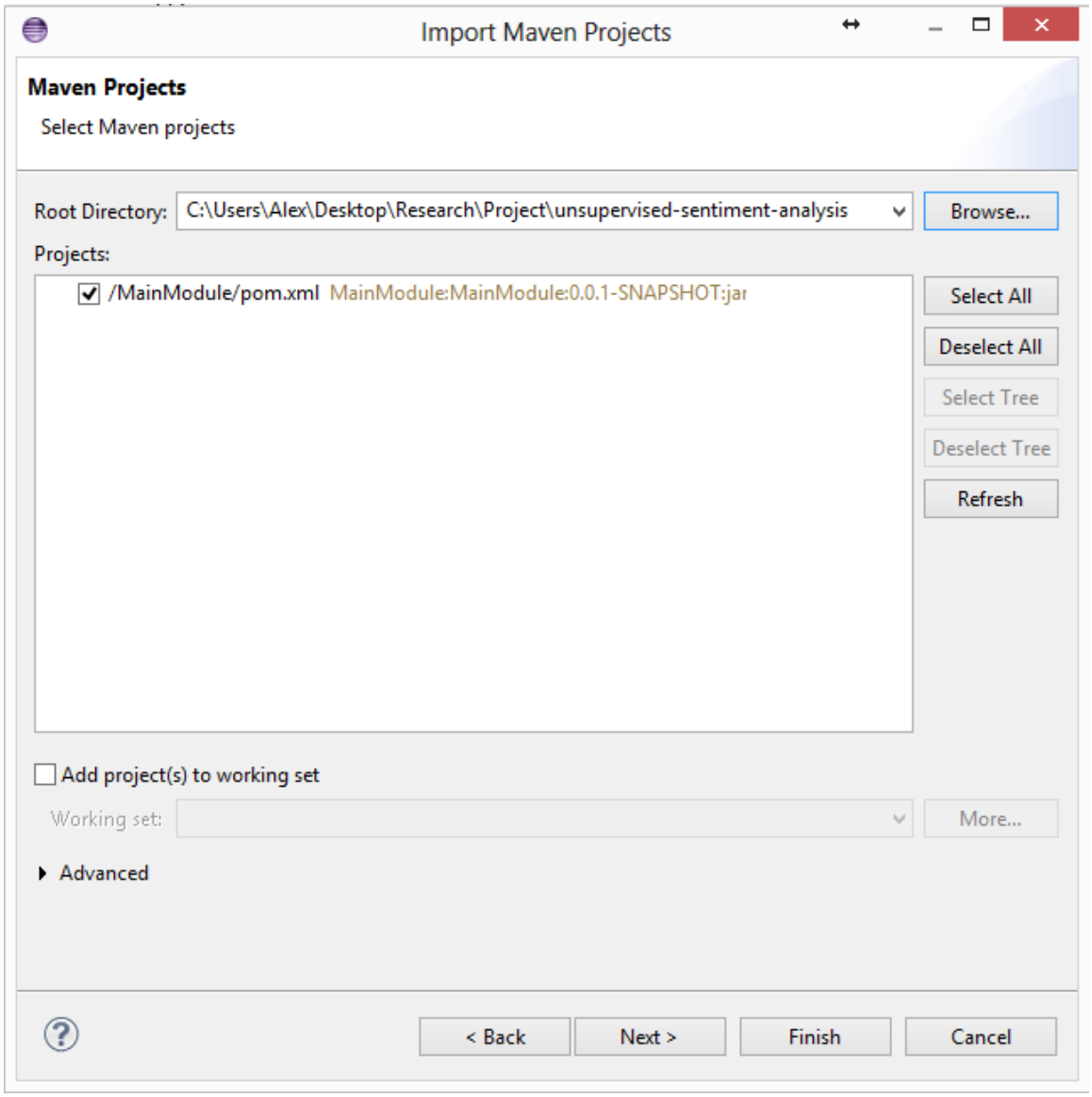

Figure 7-4 Project importing

After clicking *Finish* and waiting for all the processes to finish, the application should be imported fully into the workspace, including all additional maven dependencies which should automatically be downloaded.

7.4. Running and testing the project

This subchapter will cover all the different configurations which have to be initially changed before attempting to run the project and a quick guide on how to run it.

7.4.1. Configuration management and initial settings

The first thing that has to be done is to copy the configuration model which is present in one of the folders which was pulled from GIT. The folder name is *configModel* and the *config.xml* file inside should be copied to the *MainModule* folder. There are only a few mandatory modifications which have to be done in the *config.xml* file.

Please modify the <*inputDirectory*> and put the path to the folder …/ProjectRoot/TestReviews/Annotated.

Also modify the <SWNPath> to point to the ProjectRoot/SWN/SentiWordNet.txt file.

The file paths for the <negativeSeedWordsFile> and <positiveSeedWordsFile> must also be changed to the ones corresponding to the files in the ProjectRoot/SeedWords/*-words.txt.

The output folders can be chosen at will. The rest of the configuration parameters are not mandatorily changed however are thoroughly explained in chapter 5.3.2.

7.4.2. Running the project

After all the previous steps were finally done, the last step would be to run the application in all its glory. In the IDE package explorer, simply right click the project and select *Run As* → *Java Application.* The project will then start. As a small warning, the initial runtime is of about 1-2 hours depending on the processor power.

8. Conclusions

This thesis breaks down the problem of unsupervised opinion mining and tries to offer a viable and expandable solution to it. Starting from the presumption that *grammar* is the common denominator among all types of texts, whatever their domain is, we built a multi-dimensional tunable machine which tackles the problem of opinion mining not from a supervised and domain-dependent point of view, but rather from a domain-independent one. Its multi-dimensionality is provided by the many parameters which can be tweaked at will. From thresholds to seed word number and type, all these extra *knots* can be used to find some hidden underlying model which can in turn be utilized on other corpora as an extraction model. The project is easily maintainable and expandable, these being just two of the many non-functional attributes/requirements which were followed and implemented throughout the development process.

8.1. Main Contributions and analysis

Both levels of granularity, document and aspect level of opinion mining and sentiment analysis are handled by this approach. Using an opinion lexicon for both filtering objective opinion and polarity aggregation led to great results. The unsupervised nature of the approach, by default meant that it would be rather hard to obtain similar results to supervised approaches in terms of precision and recall. However, domain-independence was key here and as there is a lot of room for improvement, things can only get better. A thorough analysis of the parameters which make this system a multi-dimensional tunable machine can be found both in this thesis (chapter 6) and in [27] and [28].

Preprocessing and caching were of great importance as they created the backbone of the whole approach. The preprocessing module generated the fundamental element of extraction, the syntactic trees (or semantic graphs). These trees ended up being passed to both the extraction module and the polarity aggregation one and were thus of great importance. Caching made everything faster and easier as we had to run the same tests over and over again for the same files. It would have been a terribly cumbersome process to wait for the generation of those trees every time some minor parameter was changed and the system had to be run again.

As a general rule of thumb and as many may already know, precision is greatly influenced by recall and vice versa, meaning, when one goes up, the other is almost bound to go down and so on. Using our thresholds to either increase or decrease the recall we managed to increase the precision to a staggering 80-90% but at great cost of recall which dropped to under 40%. As grammatical relations and POSs are thoroughly exploited in this approach, a lot of time was invested into testing all different possibilities which might yield better (or worse) results. From adding certain relations to removing a few POSs, every bit of change led to some degree of difference in the results, thing which guided towards a better understanding of the underlying hidden models.

The greatest finding was however related to the number of seed words. Astonishing as it may seem, using only two seed words instead of a huge corpus of 6000+ led to almost the same results (precision/recall wise) but with a lot less processing

time required. As seen in chapter 6.1, using 2 seed words instead of 6000+ is ~30-40 times faster and the quality of the results is pretty much the same! The two seed words being a representative of each class (*good* – positive, *bad* – negative) led to the conclusion that this is indeed an unsupervised approach.

Overall, the system is robust and both easy and sort of fun to play with. It's interesting to see how mingling with the different parameters leads to different and sometimes surprising results. The results of the polarity aggregation module can be found in [28]. More details on the opinion extraction part can be found in [27].

8.2. Future development

As it is a very extendable system, lots of modules can be added to improve performance and the overall results. Our aim is to, step by step, improve precision and recall and exceed the results obtained by previous unsupervised opinion mining and sentiment analysis approaches.

The preprocessing module might be improved by using a database instead of the file system for storing and caching the generated syntactic trees and evaluation models. Also, a new parser like *minipar* might be tried out in order to compare the results with the current one, the StanfordCore Parser. The opinion extraction module must first be augmented with a negation detection module as negation is used quite often in any sort of text. The other big improvement to the extraction module would be a composite-noun aggregator which will take nouns formed from more than one word and aggregate them so that the extractor will only extract one. Also, we do not yet take into consideration sentences which have only an opinion word without a target. Nothing is extracted in this case and it will be a good improvement to change this. The sentence *Great! Awesome. Speechless* offers quite a bit of subjective information which however, for the moment, is not extracted in any way.

As a more *top* level/metadata related approach, a neural network which takes as input the different parameters chosen and the result of each system run could be built on top of the whole thing in order to see what the best choice of parameters for a large set of input files might be and to draw some conclusions from this.

9. Bibliography

1] N. Indurkhya and F. J. Damerau, Handbook of Natural Language Processing, Chapman and Hall/CRC, 2010.

2] "Stanford NLP Group," Stanford, [Online]. Available: http://www-nlp.stanford.edu/software/corenlp.shtml.

3] "NLTK Project," 2013. [Online]. Available: http://www.nltk.org/.

4] A. Esuli and F. Sebastiani, "SENTIWORDNET: A Publicly Available Lexical Resource," in *5th Conference on Language Resources and Evaluation (LREC'06)*, Pisa, 2006.

5] S. F. Sayeedunnissa, A. R. Hussain and M. A. Hameed, "Supervised Opinion Mining of Social Network Data Using a Bag-of-Words Approach on the Cloud," in *Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, 2012.

6] I. Smeureanu and C. Bucur, "Applying Supervised Opinion Mining Techniques on Online User Reviews," *Informatica Economică,* vol. 16, no. 2, 2012.

7] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification," *Informatica*, no. 31, 2007.

8] G.Vinodhini and RM.Chandrasekaran, "Sentiment Analysis and Opinion Mining: A Survey," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 6, 2012.

9] R.-S. Chiorean, M. Dinsoreanu, D. I. Faloba and R. Potolea, "Sentiment Polarity Identification using Machine Learning Techniques," in *ICCP*, 2013.

10] A. Balahur and M. Turchi, "Comparative Experiments for Multilingual Sentiment," *Computer Speech & Language*, vol. 28, no. 1, p. 56–75, 2014.

11] M. Anjaria and R. M. R. Guddeti, "Influence Factor Based Opinion Mining of Twitter Data Using Supervised Learning," *Sixth International Conference on Communication Systems and Networks (COMSNETS)*, vol. 1, pp. 1-8, 2014.

12] Z. Ghahramani, "Unsupervised Learning," *Lecture Notes in Computer Science - Advanced Lectures on Machine Learning*, vol. 3176, pp. 72-112, 2004.

13] B. O. Connor, R. Balasubramanyan, B. Routledge and N. Smith, "From tweets to polls: Linking text sentiment to public opinion time series.," in *ICWSM*, 2010.

14] J. Wiebe, T. Wilson and C. Cardie, "Annotating expressions of opinions and emotions in language," *Language Resources and Evaluation*, pp. 165-210, 2005.

15] T. Wilson, J. Wiebe and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *HLT and EMNLP*, 2005.

16] X. Hu, J. Tang, H. Gao and H. Liu, "Unsupervised Sentiment Analysis with Emotional Signals," in *22nd international conference on World Wide Web*, Geneva, 2013.

M. Taboada, J. Brooke, M. Tofiloski, K. Voll and M. Stede, "Lexicon-Based
17] Methods for Sentiment Analysis," *Computational Linguistics*, vol. 32, no. 2, pp. 267-307, 2011.

P. D. Turney, "Thumbs up or thumbs down?: semantic orientation applied to
18] unsupervised classification of reviews," in *40th Annual Meeting on Association for Computational Linguistics*, 2002.

G. Qiu, B. Liu, J. Bu and C. Chen, "Opinion Word Expansion and Target
19] Extraction through Double Propagation," *Computational Linguistics*, vol. 37, no. 1, pp. 9-27, 2011.

G. Qiu, B. Liu, J. Bu and C. Chen, "Expanding domain sentiment lexicon
20] through double propagation," in *21st international jont conference on Artifical intelligence*, 2009.

M. Sharifi, *Semi-supervised Extraction of Entity Aspects Using Topic
21] Models*, Pittsburgh, 2009.

V. A. S. Balan, S. Singaravelan and D.Murugan, "Combined Cluster Based
22] Ranking for Web Document Using Semantic Similarity," *IOSR Journal of Computer Engineering (IOSR-JCE)* , vol. 16, no. 1, p. 7, 2014.

D. Sharma and Prakash.R.Devale, "Approach for Transforming Monolingual
23] Text Corpus," *International Journal of Applied Information Systems (IJAIS)*, vol. 1, no. 9, p. 1, 2012.

L. Skorkovská, "Application of Lemmatization and Summarization Methods
24] in Topic Identification Module for Large Scale Language Modeling Data Filtering," in *15th International Conference, TSD 2012,*, Brno, 2012.

S. N. Group, "1," [Online]. Available: http://nlp.stanford.edu/IR-
25] book/html/htmledition/stemming-and-lemmatization-1.html.

N. Chhillar, N. Yadav and N. Jaiswal, "Parsing: Process of Analyzing with
26] the Rules of a Formal Grammar," *Journal Of Harmonized Research in Engineering*, vol. 1, no. 2, pp. 73-79, 2013.

D. Suciu, "Unsupervised aspect based opinion mining using Double
27] Propagation," 2014.

V. Itu, "Cross-domain polarity assignment by minimal interaction with
28] external resources," 2014.

"1," [Online]. Available: http://www.cs.uic.edu/~liub/FBS/sentiment-
29] analysis.html#datasets.

Appendix 1 – KDIR 2014 Article

Appendix 2 - Students' Conference 2014 Article