TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science, Institute for Systems Architecture, Chair of Computer Networks

MASTER THESIS

Mining Product Opinions and Reviews on the Web

Felipe Jordão Almeida Prado Mattosinho
Born: 02.06.1983, Cruzeiro Brazil

Adviser: Dipl.- Medieninf Maximilian Walter
Advising Professor: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Date: 02. 07. 2010

Dedicated to the memory of my beloved mother and to my lovely family.

Acknowledgements

I would like to express my most sincere thanks to the Computer Networks group which provided me always a nice work environment, with help and support. A special thanks for Josef Spillner, Dr.-Ing. Daniel Schuster, Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill and Dr.-Ing.Thomas Springer. A special thanks to my advisor Dipl.Inf Maximillian Walther for being always ready to help and to support my work with an incredible patience. For all the advices, help and constant support of Dr.Prof. Maurilio Coutinho and PD Dr. Horst Lazarek, for being great friends and incredible human beings.

Contents

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**

1.1 Motivation 2

1.2 Summary of Chapters 3

2 Basics **7**

2.1 Web Search Engines 7

2.1.1 Web Crawlers 9

2.1.2 Stemming, Lemmanisation and Stopword Removal 11

2.1.3 Inverted Indexes 12

2.1.4 Ranking Algorithms 12

2.2 Information Extraction 13

2.2.1 Data Structure Transformation in a Web Document Generation Process 14

2.2.2 Web Scraping 16

2.2.3 Natural Language Processing 18

2.3 Data Mining 21

2.3.1 Association Rules 21

2.3.2 Web Mining: Data Mining the Web 23

2.3.3 Opinion Mining 25

3 Related Work **27**

3.1 Defining Opinion Components in a Opinion Mining Context 28

3.2 Opinion Mining System Architecture Overview 30

3.2.1 Part-of-Speech Tagging 31

3.3 Feature Identification 32

3.3.1 Frequent Features Identification 33

3.3.2 Infrequent Features Identification 33

3.4 Opinion Sentiment Analysis 34

3.4.1 Identification of Word Sentiment 35

3.4.2 Determining Sentiment of Opinions at the Sentence Level 36

3.4.3 Determining Opinion Sentiment at the Feature Level 37

4 Requirements **41**

4.1 Intended Audience 41

4.2 Use Case Analysis 42

4.2.1 Use Case Description 43

4.3 Functional Requirements 44

4.3.1 General Operation 44

4.3.2 Setting System Configuration Parameters 45

4.4 Non-Functional Requirements 46

5 Design **49**

5.1 General Architecture 49

5.2 System Management Module 51

5.3 Opinion Retrieval Module 53

5.3.1 Retrieving Opinion from Sources 54

5.3.2 Opinion Retrieval Jobs 55

5.4 POS Tagging Module 56

5.5 Opinion Mining Module 58

5.5.1 Opinion Model 58

5.5.2 Automatic Feature Identification 59

5.5.3 Word Sentiment Classification 60

6 Implementation **63**
6.1 Technologies Overview . . . 63
6.1.1 Core Technologies . . . 64
6.1.2 Software Components . . . 64
6.2 User Interface . . . 66
6.2.1 Web-based User Interface . . . 66
6.2.2 Web Services User Interface . . . 66
6.3 System Management Module . . . 67
6.3.1 Request Management . . . 67
6.3.2 Job Scheduling . . . 69
6.3.3 Running Jobs . . . 69
6.4 Opinion Retrieval Module . . . 70
6.4.1 Amazon Web Services . . . 70
6.4.2 CNET Web Scraping . . . 71
6.5 POS Tag Module . . . 74
6.6 Opinion Mining Module . . . 76
6.6.1 Feature-based Summary Generation . . . 78

7 Evaluation **83**
7.1 Test Environment . . . 83
7.2 Effectiveness of Opinion Sentiment Classification . . . 84
7.2.1 Considerations . . . 87
7.3 Effectiveness of Automatic Feature Identification . . . 87
7.3.1 Considerations . . . 88
7.4 System Efficiency . . . 88
7.4.1 Considerations . . . 89

8 Conclusion **91**
8.1 Future Work . . . 91

References **93**

List of Figures

2.1 A search engine system during a search operation 9
2.2 A web crawler system in detail 10
2.3 A CGI script dynamically generating web documents 15
2.4 A web document from three different perspectives 17
2.5 A web scraping agent gathering information from web pages 18
2.6 A web scraping agent may issue several requests to access even a small amount of information 19
2.7 Web Mining 24

3.1 Opinions highlight strenghts and weaknesses of attributes of an OuD or its sub-components. 29
3.2 Architecure of an opinion mining system 30
3.3 Infrequent feature extraction 33
3.4 Users opinions for a digital camera 35
3.5 Bipolar adjective structure.The solid arrows mean "similar", and dashed one "antonym"[15] 38
3.6 Predicting the orientation of opinion sentences [15] 38
3.7 Predicting the orientation of opinions on product features [9] 39

4.1 Users of the system with respect to their roles. 42
4.2 OMS use case diagram. 43

5.1 The Opinion Mining System Architecture 50
5.2 The SMM assigns jobs for workers 52
5.3 Web Scraping agent extracting reviews of Nikon P90 on CNET web site. 57
5.4 A composition model for opinions 59

LIST OF FIGURES

5.5 Representation of the Wordnet word orientation search algorithm 60
5.6 Flowchart representing the WordNet word orientation search algorithm . 62

6.1 Web-based user interface 66
6.2 A XML message generated by the Web Service Interface 67
6.3 Jobs stored persistently in the database 69
6.4 The ORM UML class diagram 70
6.5 Web-based user interface 73
6.6 A CNET review presentation and the HTML code 75
6.7 Opinion Mining Module UML Class diagram 77
6.8 A CSV file containing words and their orientations 79
6.9 Sentences with negation words 82

7.1 Effectiveness of opinion sentiment classification 85
7.2 Effectiveness of the automatic feature identification algorithm 88
7.3 Time needed for code execution under different mining scenarios 89
7.4 Memory consumed by the mining algorithm for different mining scenarios 90

List of Tables

4.1 Stakeholders of the system and their goals. 42

5.1 Job description for product “Nikon P90” on CNET 55
5.2 CNET’s URLs to Nikon P90 camera reviews 56

6.1 POECS configuration parameters 68
6.2 Job description for product “Nikon P90” on CNET 72

7.1 Sample data for evaluation tests 84
7.2 Effectiveness of sentiment classification 85

LIST OF TABLES

Chapter 1

Introduction

Since its creation, the World Wide Web has undergone a fast change, when common users started to join the network. Meanwhile, several new technologies appeared to enhance computers and networks, operational costs related to web sites and the price of computer hardware decreased dramatically, which all together contributed to a mass popularization of the Internet.

Also, an amazing number of new development tools, applications and frameworks appeared, like the so called CMS (Content Management System), which is a framework that allows agile web site development. These ready-to-use systems have all the facilities necessary for easy installation, content publication and edition, enabling a common user to publish online content without being a computer expert or a programmer. Because of these facilities, more forums, blogs and specialized web sites have been developed, increasing dramatically the number of content generated by ordinary users. With the new concepts introduced by Web 2.0, more value started to be given to the active participation of users and communities, especially through knowledge contribution of each member to enrich global information.

As a consequence of this new order, providing a public space for community discussions became almost a pattern for new web applications. To meet the new demand, most existing web sites like social networks, web magazines, newspapers and e-commerce had to change their systems to comply with new design standards (introduced by Web 2.0), which among other things provides a common space allowing interaction of users through the exchange of opinions and experiences. Since then, a huge amount of data has been produced by users, a valuable content that can be extremely useful both as

a complementary, but also in many cases as a primary and unique source of information. Because of the nature of this content, which is unstructured data in form of free text, the recovery and extraction of meaningful information depends on specialized techniques. This work explores the use of such techniques focused on the analysis of user-generated content in the e-commerce context. Hence it will specifically analyze *consumers opinions* or *customers opinions*, also called *users reviews*. Even though this work narrows the subject domain, nearly the same concepts can be applied and extended to any other domain which is likely to admit opinions.

1.1 Motivation

The amount of available information nowadays makes the actual system each day more concerned about how to handle information overload, ensuring that the user will have access to the best sources with the least effort. On recent years, special attention has been giving also to the amount of produced user-generated content. The e-commerce sector is one of the most affected by the amount of data produced by customers, which increased dramatically during the phase known as Web 2.0. Customers opinions represent a valuable unique type of information which should not be mistreated or ignored by the research community. Thus, this work emphasizes the need of special mechanisms that aims to provide the community better ways to take full advantage from this data.

From the customer perspective, considering others opinions before purchasing a product is a common behavior long before the existence of Internet. In the era of the digital world, the difference is that a customer has access to thousands of opinions, which greatly improves decision making. Basically, customers want to find the best for the lowest price. In other words, they search for products that best fulfill their needs inside a price range that they are willing to pay.

It is important to emphasize that the benefit of analyzing other opinions, comes from their neutral nature, which are usually not linked to an organization or company. They represent the voice of ordinary consumers, and that differs greatly from ads (advertisements are biased and tend to favor the product, emphasizing the positives aspects and concealing the negatives ones).

From the e-commerce perspective, receiving consumers feedback can greatly improve its strategies in order to increase profits of the sector. For example, an online

shop can place smart ads by measuring the level of satisfaction of consumers for a given product. For instance, if a product has a low level of satisfaction, a smart strategy would be placing a competitor ad inside this page.

It is common to find products with thousands of opinions, thus it could be a hard task for a customer to analyze all of them. Also, it could be a very tiresome work to find opinions about just some features from a product, usually a requirement for an experienced customer.

An important difference makes the actual ranking mechanisms not so efficient to depict the information represented by opinions. This difference is mainly due to nature of textual information in the world. These information are either facts or opinions. The actual search systems are focused on facts (e.g ranking mechanisms used by search engine). One fact is usually equal to all other same facts. An opinion however is a belief or judgement of a subject. Therefore, one opinion from an object under discussion (OuD) is usually different from multiple opinions for the same OuD. In this sense, a summarization mechanism portraits better the reality of opinions and thus provides better ways for users to draw conclusion out of them.

This work presents ways for locating, extracting, classifying and summarizing opinions or reviews on the Internet. The proposed framework will combine several techniques to extract valuable information out of natural language text (user-generated content), in order to provide enrichment of the experience of users by taking advantage of the available content in a more intelligent and organized way. As a consequence of the employed techniques, data can be structured, this will also provide a necessary bridge for many applications to be able to fully interact with others in a Web 3.0 context.

1.2 Summary of Chapters

Including this chapter, this work is divided into eight parts as following:

Chapter 2 - Basics

Opinion Mining is a new subject and stays at the crossroads of Information Retrieval, Information Extraction and Web Mining. This chapter will provide a basic background about the mentioned areas explore, presenting technologies and concepts which will be used throughout this work.

Chapter 3 - Related Work

Related work will investigate some researches in the Opinion Mining field. These works mainly aim areas such as sentiment analysis and feature identification. This chapter will demonstrate different strategies used for different works to cope with sentiment analysis and feature identification in opinions, exposing the pros and cons of the employed methods.

Chapter 4 - Requirements

This chapter will address the audience expectations with an opinion mining system. From the system perspective, some non-functional requirements must be addressed with care to guarantee that the system will meet all the functionalities specified for proposed opinion mining system.

Chapter 5 - Design

Design presents the overall architecture of the opinion mining system and details about each subsystem task. The details about each basic subsystem are also introduced. The System Management Module which is responsible for handling communication and requests between the subsystems. The Opinion Retrieval Module in charge of retrieving opinions from the Internet. The POS Tagging Module which tags opinions with their part-of-speech for later processment and finally the Opinion Mining Module which produces a high-quality summary out of the analyzed opinions. This chapter provides all the necessary knowledge for the realization of the system on chapter 6.

Chapter 6 - Implementation

This chapter presents the technologies involved on the development of a platform called POECS (Platform for Opinion Extraction, Classification and Summarization). Also, all the aspects relevant to the chosen technology will be discussed, as the classical object-oriented analysis and code demonstration.

Chapter 7 - Evaluation

This chapter shows all the results achieved with POECS. The evaluation of the system will be focused on accuracy of feature identification and the number of correctly

classified opinion orientation.

Chapter 8 - Conclusion

The last chapter will discuss the results achieved with the system and the biggest shortcomings found during the development. Also, it will highlight the new directions expected for future works, where more efforts should be put on.

Chapter 2

Basics

This chapter presents basic technologies and tools used by Opinion Mining, a specialized area of the so called Web Mining (Data mining applied to the Web). Web Mining stays at the crossroads of Information Retrieval, Information Extraction and Data Mining. Both Information Retrieval and Information Extraction play important roles to locate and extract valuable information out of unstructured data, before it is suitable to be processed by data mining applications. Exploring better these techniques is extremely necessary to cope with the amount of available data in the Information Overload Era. Also, with the Web being increasingly oriented towards the importance of semantics and integration of information, these areas of study become very important to address the new future trends of the Web. This chapter is divided as follows: Section 2.1 gives an overview of search engines and provides basic explanations about its components. Section 2.2 presents Information Extraction tools and techniques. Finally, section 2.3 introduces Web Mining, a new challenging area of study with a great importance nowadays due to the amazing growth of the Internet in the recent years.

2.1 Web Search Engines

Information Retrieval (IR) is a field of study concerned with the retrieval of documents from a collection of other documents (relevant and non-relevant), usually based on keyword searches. With the Internet expansion, Information Retrieval got a very special focus, as search engines became a dominant way to find information on the Web. Nowadays, because of their importance, search engines are the most representative

name in Information Retrieval and therefore this section will treat the technologies, goals and development issues involved in their design decisions.

One of the reasons why some search engines are so successful on the Internet are their commitment to quality services, especially with regard to the processing speed of users queries. In today's Internet, with billions of available pages, the lack of mechanisms that provide a response in a short amount of time would leave the system incompatible with the standards of the modern times (more data must be processed with even more stringent constraints with respect to time). The main names behind search engines (e.g, Google, Yahoo, Bing, etc.) achieve this high standard of service mainly due to their indexing technique combined with high-end infrastructure comprised of several hundreds of clusters highly optimized for jobs demanding large capacity of computational work. These systems are very scalable and are able to deliver high quality services even with millions of users accessing their systems simultaneously. Moreover, ranking algorithms (e.g, Google's PageRank), are able to sort the most important documents related to a search. Without the help of ranking algorithms a user would have no clue about where to start searching for a desired information among several other equal possible documents. A ranking algorithm provides a hierarchical level of more important documents, thereby providing an initial clue to the user about where the desired information is more likely to be.

During a *search operation* the following interactions are performed, as shown in figure 2.1: (1) A query is submitted by the user. (2) The user query is checked to ensure that it is ready to be used by the retrieval system. This could be achieved through simple tasks such as removing *"stopwords"*, reducing the words to roots (stemming) and checking the spelling. (3) The query is checked against the available indexes in order to retrieve the documents that contain some of the query terms. Afterwards, a ranking algorithm is applied to the set of found documents which are finally presented to the user (the most relevant documents appear at the beginning of this list). (4) The user receives the response and accesses the matching documents from a result list.

The aforementioned steps show the search engine in its *search operation* phase, directly serving a user request. However, the main tasks must be performed in advance, as crawling web pages, their subsequent indexing and ranking calculation. The following subsections will present each internal subsystem of a web search engine and their respective tasks.

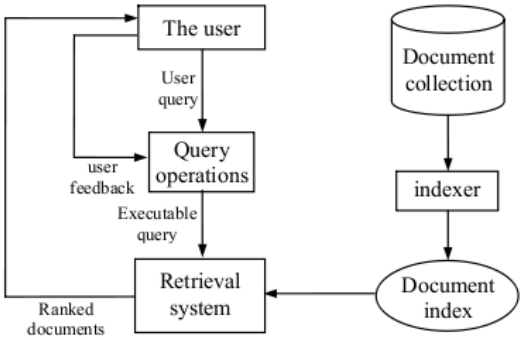

Figure 2.1: A search engine system during a search operation - An user issues a query which is first checked before it is forwarded and compared to documents indexes [5]

2.1.1 Web Crawlers

Search engines rely on computer programs called Web Crawlers (also called Web robots or Web spiders), to traverse web pages by following hyperlinks and storing web documents that are later indexed to optimize the search process. A web crawler is probably the most important and complex component of an online search engine.

Web crawlers have two important issues to address: First is to use a good crawling strategy (which includes the algorithm strategy for traversing new web pages) and intelligent mechanisms to optimize the process of re-crawling. Second, because this task is computational intensive, the system must be able to deal with many different scenarios under different circumstances (hardware failure, server problems, errors while parsing documents) while still maximizing the work to ensure that the maximum advantage is taken out of the available resources (such as limited network bandwidth, computer memory, cpu, etc).

A web crawler system [32], may have some of the following components, as presented on 2.2:

A crawl manager [32] is responsible for forwarding URL requests to the *downloaders*. Typically the crawling operation starts with a seed of hyperlinks, and as their are visited by crawler the new identified hyperlinks are added to the list of URLs to visit (also called the *crawl frontier*). Also, this component has the task to enforce the rules imposed by robots.txt[1], provided by web server administrators. *Downloaders* are responsible

[1]Robots.txt is a file used to enforce rules that web crawlers should follow when exploring the links of a web site. Normally a web crawler checks this file content to ensure that it is authorized to visit a

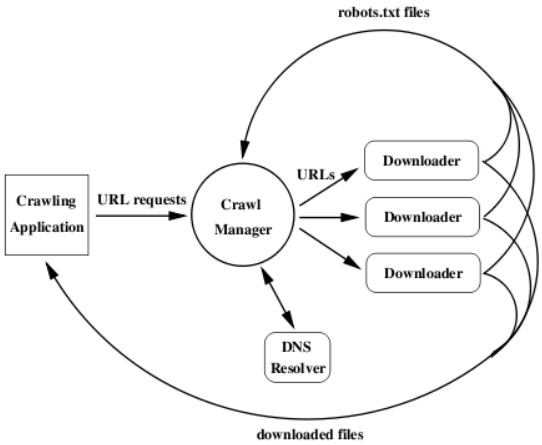

Figure 2.2: A web crawler system in detail - A Web Crawler is a very complex system. Each subsystem has to cope with many different failure scenarios and still be able to scale with limited resources.

for opening connections with different web servers and polling these connections for arriving data. In [**?**] it was reported that the developed search engine would receive hundreds of pages per second through the downloaders, thus writing them to disk in one disk operation. In the architecture of [32] a schema with localized DNS resolver in order to speed up the resolving process as shown in figure 2.2.

Crawling Methods

Web Crawlers may differ from each other in the way they crawl web pages. This is mainly related to the final application that the web crawling system will serve. Two examples of crawling strategies are:

Broad Breadth-First Crawler

A crawler can select a small set of web pages, as a seed list, and follow their links using a breadth-first like algorithm. Real search engines employ a series of other techniques to improve the crawling algorithm, like pruning crawls inside a web page or crawling better ranked web pages first. With this strategies all the links are followed and therefore there is no restriction between the subjects covered by the website.

certain section of a web page.

Topical Crawlers

Topical crawlers, also known as focused crawlers, attempt to crawl only specific pages. These could be pages of a particular subject or in a specific language, image, mp3 files or computer science research papers. The goal of this crawlers is to find as many as possible pages of interest without using a lot of bandwidth.

2.1.2 Stemming, Lemmanisation and Stopword Removal

To optimize the search process and to maximize storage capacity, recent crawled web pages are preprocessed before indexes can be built and the pages safely stored. Stemming is a process that reduces words by removing suffixes, thereby mapping them to the same root stem. An example of words (left side) and their corresponding stems (right side):

Example 1:

I. tests, testing, tested → test

II. cooking, cooked, cooks → cook

Another advantage of reducing words to a root, is to increase the hit rate of similar terms. Therefore not only identical words would be retrieved, but also their corresponding stems. Another similar technique is called *Lemmanisation*. The important difference between Stemming and Lemmanisation is that the later reduces words to their roots, using vocabulary and morphological analysis of words, therefore removing inflections ending and returning back to the dictionary form of a word, also known as *lemma*. Stemming is a much simpler heuristic process, which chops the end of words in the hope to perform these operation correctly in most of the times. This process however often also removes derivational affixes, causing undesirable results. Lemmanisation uses a more sophisticated process and achieves better results through more complex algorithms, but it also performs slowly. For this reason, many applications where accuracy is not a strong requirement, tend to use stemming as it performs much faster.

Stopword removal removes words that may have little lexical meaning, or may not change the semantics of a text or a query, which are usually used to connect words or

to express grammatical relationship with other sentences inside a sentence. They are usually articles, prepositions, pronouns, etc. Common English stopwords include: *the, on, of, with, a, about, what, when, where, that, this, by, be.*

2.1.3 Inverted Indexes

A search engine system might have to search for billions of documents. Searching all of them for specific terms (from a given user query), would take a great amount of time. To help search engines performing the search in an acceptable amount of time, the retrieval system uses data structured called indices. The best indexing scheme and most widely used for web search engines is the inverted index. An inverted index is a data structure comprised by a term and all the documents that contain this term. Besides its name, the inverted index works exactly like a "normal book index" , therefore one should ignore the word *inverted* to avoid misinterpretation. Suppose three documents *D1*, *D2* and *D3*, with the following content:

Example 2:

D1 → A nice digital camera

D2 → A bad analog camera

D3 → A camera with nice features

An inverted index with the following documents would look like:

I. **nice**: [D1,D3]

II. **digital**: [D1]

III. **camera**: [D1,D2,D3]

IV. **bad**: [D2]

V. **analog**: [D2]

VI. **features**: [D3]

2.1.4 Ranking Algorithms

Ranking a search result is a necessary task to improve the user experience. With the amount of online documents it is almost impossible for a user to check every single document to testify its relevance. Also, ranking algorithms help to defeat web spamming, a practice not just harmful to users experiences but also to the information retrieval

business. As a document is treated as a *bag of words* by the search engine, a page owner can repeat many words in his page, to make it relevant for a great number of queries.

One of the most important ranking algorithms, in web search is Google's PageRank, which usually uses the concept of *prestige* to sort relevant documents. The idea is that web pages that are referenced by many other web pages (through hyperlinks) is likely to be an important web page. Therefore, the more in-links that a page has, the more important it is. However, the prestige score is not just limited by the number of links that point to a web page. The algorithm also takes into account the prestige of another pages. Out-links from more important pages (with more prestige) have a higher weight on the PageRank score. Therefore, the importance of page i (i*'s* PageRank score) is determined by summing up the PageRank scores of all pages that point to i divided by the number of their out-links. Google lists the pages with higher rank scores first on their result list.

2.2 Information Extraction

Information Extraction (IE) is a sub-discipline of artificial intelligence that aims to extract valuable information out of unstructured data. An extraction information system usually is focused on entities or objects identification (people, places, companies, etc) and extraction rules are usually but not necessarily domain specific. Unstructured data can have many different forms, such as videos, images, audio and text. The first information extraction systems were mainly focused on text, and still nowadays this is the most explored type of data by the research community and commercial frameworks. For this reason, the examples and further explanations on this section will be focused on texts.

The goal of IE is to identify useful parts out of raw data (unstructured data) and extract them to finally build more valuable information through semantic classification. The result may be suitable for other information processing tasks, such as IR and Data Mining.

There is classically a difference between the goals of IR and IE, but in the real world they should be seen as close complementary activities. to improve their precision and accuracy. The need of this cooperation is well summarized by the following citation:

> "[...] the classical information retrieval paradigm is no longer preferable. This paradigm has found its roots in the example of the traditional library. One is helped in finding potentially relevant books and documents, but the books and documents are still consulted by the humans. When the library is becoming very large and the pile of potentially relevant books is immensely high, humans want more advanced information technology to assist them in their information search. We think that information extraction technology plays an important role in such a development." (Marie-Francine Moens) [13]

It is important to highlight that the term *extraction* means that the target information is explicitly available (in texts for instance this information is readable). The need for this explanation is to differentiate between statistical data mining techniques which *deduce* information out of available data. The fact that both techniques can *build knowledge* from a given data set does not mean that they can be used interchangeably. Actually, information extraction is a necessary pre-processing step to structure data before a statistical data mining algorithm can build knowledge (at this point still *hidden*) from it.

This section will focus on the application of IE techniques turned to the web. Next subsection 2.2.1 will show how data web documents are generated and how this process affects the data structure. Subsection 2.2.2 and 2.2.3 will show how IE can help the recover of information from the generated web documents.

2.2.1 Data Structure Transformation in a Web Document Generation Process

The generation of web documents may involve many different types of data structures along the process. In the so-called static web documents, an HTML document will hold the same information (content), no matter which client is requesting the page, or in which context this page is being called. All the information is enclosed between HTML tags, whose main function is to markup parts to provide structural semantics for text (paragraphs, lists, headings, etc).

With dynamic web pages, pages are generated by a script server, and they usually change as a response from different clients under different scenarios. They are docu-

ments generated per request, and an example would be an e-commerce which shows products to customers based on keyword searches. Different keywords return a different list of products and a common practice is letting the script server query a database, which will return data in accordance with the request. The static part serves as a template for the structured data that comes from the database. Usually, the same template is reused by other similar data in the database. The new generated document will hold the structured data which will be part of the document as a normal text. This mechanism can be better noticed on figure 2.3. At this point it is not anymore possible to retrieve the structured information using for example a specialized structured language such as SQL.

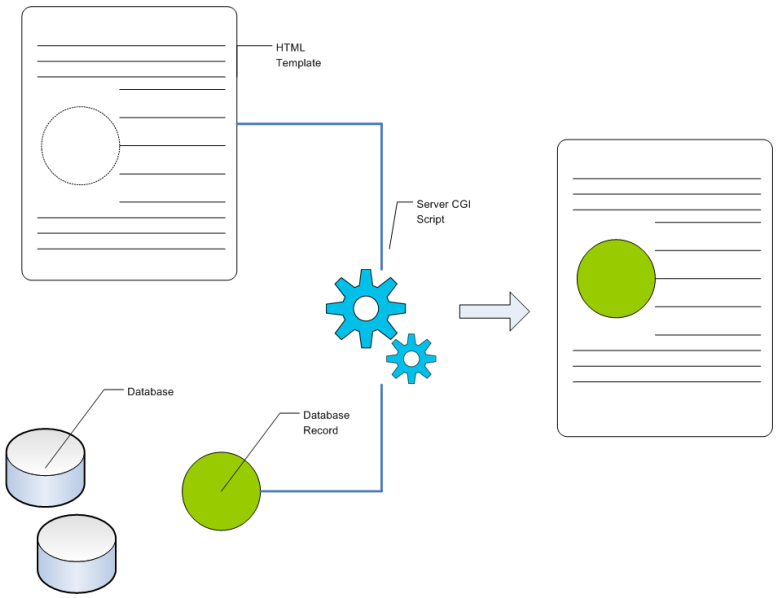

Figure 2.3: A CGI script dynamically generating web documents - A CGI script also known as server side script builds a HTML template togheter with structured data (from the database) to form a web page

A web document has normally several parts where each of them are labeled with HTML annotations(<*div*>, <*title*>, <*body*>). Because of this property of labeling different parts of the document in accordance with the information they hold, a web document is a type of a semi-structured document (as it still preserves some kind of structure, when compared with a plain text document, for example). Once structured data becomes part of a semi-structured document, the structural properties are lost

and therefore to retrieve the desired information, special techniques have to be used, which are going to be presented on the next section. Also, an important caveat here is that talking about structural properties of a data depends on the granularity being analyzed.

A web document with a text enclosed by a $<div>$ tag could be viewed from two different perspectives concerning the granularity of the desired information. The document itself is semi-structured, but the text inside the div tag is completely unstructured. Therefore, an attempt to retrieve the whole text would be pretty straightforward as the $<div>$ tag in this case would work like a hook for the extraction algorithm. The markup tags provide a very efficient way to narrow the possible locations of a target information inside the whole document. It should be clear that if one is willing to recognize entities inside this text, then specialized extraction techniques would have to come in action such as the ones explored by Natural Language Processing (NLP). The subsection 2.2.3 will discuss NLP later with details.

2.2.2 Web Scraping

Web scraping is a technique used to harvest information from web pages based on script routines. Web pages are documents written in Hypertext Markup Language (HTML), and more recently XHTML which is based on XML. Web documents are represented by a tree structured called the Document Object Model, or simply the DOM tree and the goal of HTML is to specify the format of text displayed by Web browsers as shown in figure 2.4.

Since web scraping uses data from other online sources, it is important to consider the deliberately use of this technique, as the content from one web site is usually copied and republished by another one. Usually is sufficient to read their terms of service to ensure that data reuse is allowed, or to simply ask permission for that.

From the operation perspective, a web scraping resembles in many ways a manual copy and paste task. The difference here is that this job is done in a organized and automatic way, by a virtual computer agent. When an agent is following each link of a web page, it is actually performing the same operation that a human being would normally do when interacting with a web site. This agent can follow links (by issuing HTTP GET requests) and submit forms (through HTTP POST), browsing through many different web pages. While a computer would perform manual tasks at the speed

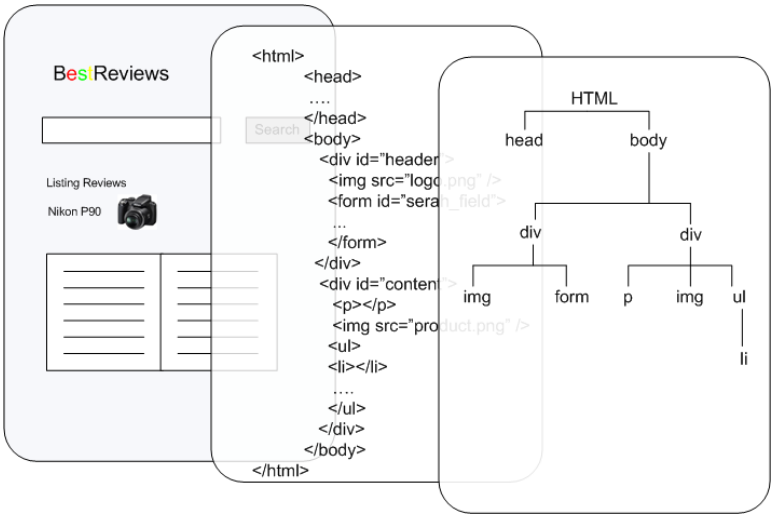

Figure 2.4: A web document from three different perspectives - From left to right: The document presentation, the HTML code and the DOM

of a computer instruction, a human would have to think, grab the mouse, point to the link and finally click on it. Now the benefit seems clear when a user has to click on several links before getting to the actual desired page. However, not surprisingly, the benefit of issuing requests at a script speed, also brings a problem. If one uses web scraping without a policy for limiting requests, the requested server may find that someone is trying a Denial-of-Service attack, due to the great amount of requests triggered in a short period of time.

After retrieving the target web document, the parser follows user-specified paths inside the document to retrieve the desired information. These paths are specified by CSS selectors or XPATHs. They use both relative and absolute paths (based on the DOM tree) to point the parser to a specific element inside a web document. After locating the desired information, normally web scraping operations uses also regular expressions to narrow or prune the located information, in order to retrieve data with an user-specified granular size. This process is illustrated in figure 2.5.

A big shortcoming of web scraping, is the difficulty to generalize extraction scripts. The script is usually tied up to the DOM model of a given page, therefore the dependence introduced by XPATHs or CSS Selectors, make it not easily reusable through different web sites. Also, considering performance web scraping may not be an optimal solution for retrieving information, specially when using it in larger scales or for

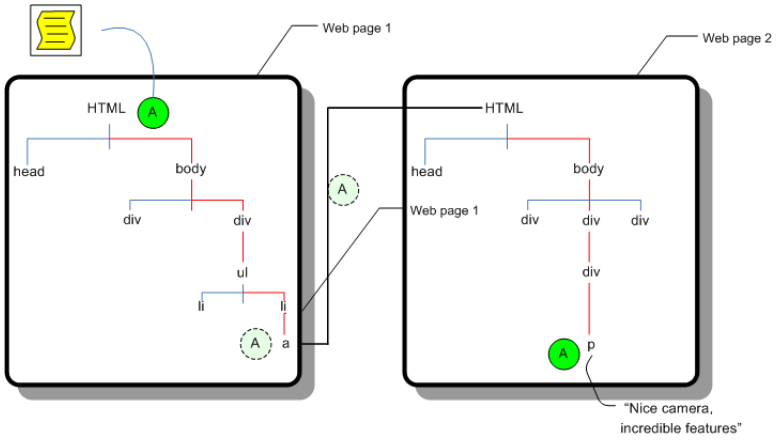

Figure 2.5: A web scraping agent gathering information from web pages - The dotted circle represents a web scraping agent traversing a DOM tree. The red lines are XPATHs to a desired element within the document. The agent reaches the hyperlink in web page 1 and proceeds to web page 2 until it finds the information enclosed by element p (paragraph)

commercial solutions. The request of an entire document when just a small part of it is actually necessary, makes it a very expensive process from the performance point of view as illustrated in figure 2.6. However, still with the mentioned shortcomings, web scraping can be a very powerful technique (an possibly the only public known method), when no other option for retrieving information on an user-specified size is available.

2.2.3 Natural Language Processing

Natural Language Processing (NLP), also known as computational linguistics, is a field of computer science that studies interactions of human languages with computers. The main goal of NLP is to enable effective human-machine communication, which could be either as spoken or written form. Here, only the written form will be addressed.

For many applications, is desirable to automatically process texts written in natural language. Computers can parse and automatically generate natural language texts, extract semantics from them and identify real world objects. As a consequence, many new applications could benefit from it, as well as existing ones which could become more human friendly. Some examples include search engines understanding natural language text queries and data information extraction applications which could interpret a large amount of text and store just the significant parts in a database. Next subsection

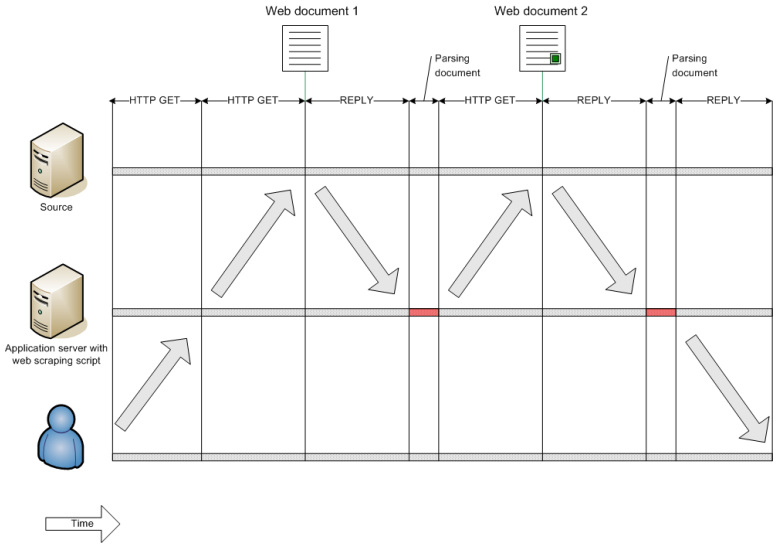

Figure 2.6: A web scraping agent may issue several requests to access even a small amount of information - Entire documents are returned no matter how small is the target information. This represents a great overhead when small information is required from multiple pages.

will present an important application of a NLP technique used in text mining called Part-of-Speech tagging.

Part-of-speech Tagging

One special application of natural language processing is determining the part of speech of each word in a sentence, known as part-of-speech (POS) tagging. The part-of-speech is a category used in linguistics that is defined by a syntactic or morphological behaviour of a word. The traditional English language grammar classifies parts-of-speech in the following categories: *verb*, *noun*, *adjective*, *adverb*, *pronoun*, *preposition*, *conjunction* and *interjection*. However some other sources admit more categories, like *articles* or even more based on variations of the aforementioned ones.

The reason why POS tagging is so important to information extraction is the fact that each category plays a specific role within a sentence. *Nouns* give names to objects, beings or entities from our world. An adjective qualifies or describes nouns. Also some adverbs can play pretty much the same role as an adjective, however under different circumstances, as they rarely modify a noun. The next example summarizes some

part-of-speech taken from opinions on the Internet:

I. *I have a good camera.*

II. *My camera is bad.*

III. *The autofocus function never works.*

IV. *I always have problems with the autofocus.*

V. *This camera comes with a recharger that works everywhere.*

In I and II, both good and bad qualifies the noun camera. In III, IV and V the words never, always and everywhere, also qualify the noun but different from adjectives they modify the verb. After classifying parts of a sentence, many algorithms of data mining can benefit from it, using statistical methods to determine the likelihood of a word to belong to a specific group.

By tokenizing a sentence with their part-of-speech a system can benefit from their morphological and syntactical behaviour, creating hooks for other algorithms to extract certain words. As a consequence, algorithms can be less error prone,(by discarding elements that may not be part of a desired group) and more efficient, (eliminating the need of expensive data mining algorithms to process a whole set of words). Finally, an algorithm can create intelligent associations, by deducing semantics in a sentence. The following examples show some sentences tagged with their part-of-speech:

Example 3:

Part-of-Speech from Penn Treebank

I. *I_PRP have_VBP a_DT good_JJ camera_NN ._.*

II. *My_PRP$ camera_NN is_VBZ bad_JJ ._.*

III. *The_DT autofocus_NN function_NN never_RB works_VBZ ._.*

IV. *I_PRP always_RB have_VBP problems_NNS with_IN the_DT autofocus_NN ._.*

V. *This_DT camera_NN comes_VBZ with_IN a_DT recharger_NN that_WDT works_VBZ everywhere_RB ._.*

Each tag represent a part-of-speech (*JJ*, *NN*, *VBZ*, etc). In the above annotation *JJ* stands for *adjective* while *NN* means *nouns*.

2.3 Data Mining

As discussed in section 2.2, Information Extraction deals with the problem of extracting explicitly available information, mainly from texts. Data Mining on the other hand can build correlations between data, which is still not explicitly available, therefore building a knowledge out of it. This process is also known as *Knowledge Discovery* and it is defined as the "non-trivial extraction of implicit, previously unknown and potentially useful information from data"[6]. This makes Data Mining techniques a useful necessary tool in many applications, specially in systems where huge amount of data has to be analyzed, something infeasible for an human being to perform manually. Subsections 2.3.1 presents two important data mining techniques. Finally, subsection 2.3.2 gives an overview of what is *Web Mining*, and introduces *Opinion Mining* a subarea of Web Content Mining, which addresses all the subjects explored by this work.

2.3.1 Association Rules

Association rule is one of the most widely used data mining concepts. The goal of an association rule mining algorithm is to discover associations between data items. It is classically defined as: Let I be a set of items $i_1, i_2, \ldots, i_n$. Let $T = t_1, t_2, \ldots, t_n$ be a set of transactions. Each transaction t in T has unique id and contains a subset of items in I. An association rule implies the following: $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$.

Examples of association mining applications are, market basket analysis, medical diagnosis and research, web site navigation analysis and homeland security. An association rule is illustrated in example:

Bread →Cheese (support = 10%, confidence = 90%)

This rule says that 10% of customers bought bread and cheese togheter and those who bought bread, also bought cheese 90% of the time. Support and confidence are two important measurements association rule mining. In [5], support and confidence are defined as follows:

Support: The support of a rule, X →Y, is the percentage of transactions in T that contains $X \cup Y$, and can be seen as an estimate of the probability, $Pr(X \cup Y)$. The rule support thus determines how frequent the rule is applicable in the transaction set T.

Let n be the number of transactions in T. The support of the rule X →Y is computed as follows:

$$support = \frac{(X \cup Y).count}{n} \tag{2.1}$$

Confidence: The confidence of a rule, X →Y, is the percentage of transactions in T that contain X also contain Y. It can be seen as an estimate of the conditional probability, $Pr(Y|X)$. It is computed as follows:

$$confidence = \frac{(X \cup Y).count}{X.count} \tag{2.2}$$

The following example can illustrate association rules better:

Each transaction is a set of items purchased by a customer in a supermarket:

t_1 : Cheese, Bread, Milk

t_2 : Orange, Sugar , Milk

t_3 : Cheese, Milk, Sugar

t_4 : Bread, Cheese, Butter

t_5 : Bread, Butter, Milk

Letting the user specify the minimum support(minsup = 30%) and the minimum confidence(minconf = 60%), for the following association rule:

Bread →Cheese (minsup = 2/5, minconf= 2/3)

The above rule is valid as the minsup = 40% and the minconf = 66.6%.

Cheese →Butter (minsup = 1/5, minconf = 1/3)

The above rule is invalid as the minsup = 20% is below the specified minsup(30%), and minconf = 33.3% which is below the 60%.

There are a large number of available association rule mining algorithm with differences between mining performances. The most famous association rule mining algorithm is the Apriori algorithm. More details about this algorithm can be found on **??**.

2.3.2 Web Mining: Data Mining the Web

The classical data mining systems has its origins long before the Internet. These systems were used mainly for pattern recognition or knowledge discovery inside large databases in data warehouses. Databases maintain data in a structured way, where information is stored in small meaningful pieces which are responsible for building a certain knowledge about a specific domain. Historically, data mining has been addressing specially databases, but recently the field has evolved to accommodate needs of the Internet. The name Web Data Mining or simply Web Mining has been used recently to reference the use of data mining techniques on the Internet context. Web Mining has some important differences from classical data mining, specially with respect to data collection. In data mining it is often assumed that data is already collected and stored in the database. In Web Mining special mechanisms imported from IR and IE have to be employed to collect data and to prepare it before it is suitable to be mined. Concerning the goals of the mining tasks, Web Mining is subdivided in three categories as illustrated in figure 2.7.

Web Structure Mining

Web structure mining aims to discover useful knowledge from hyperlinks to maximize knowledge about web page relations.

Web Usage Mining

Web usage mining aims to discover patterns from web usage logs. These logs record every click made by an user and they are important because they translate what users are looking for.

Web Content Mining

Web content mining is very important as it deals directly with information. The goal is to mine content from web documents in order to build knowledge from it. This knowledge can be either hidden or somehow simply difficult to be analyzed in a straightforward way. Next subsection will introduce an important field of Web Content Mining called Opinion Mining.

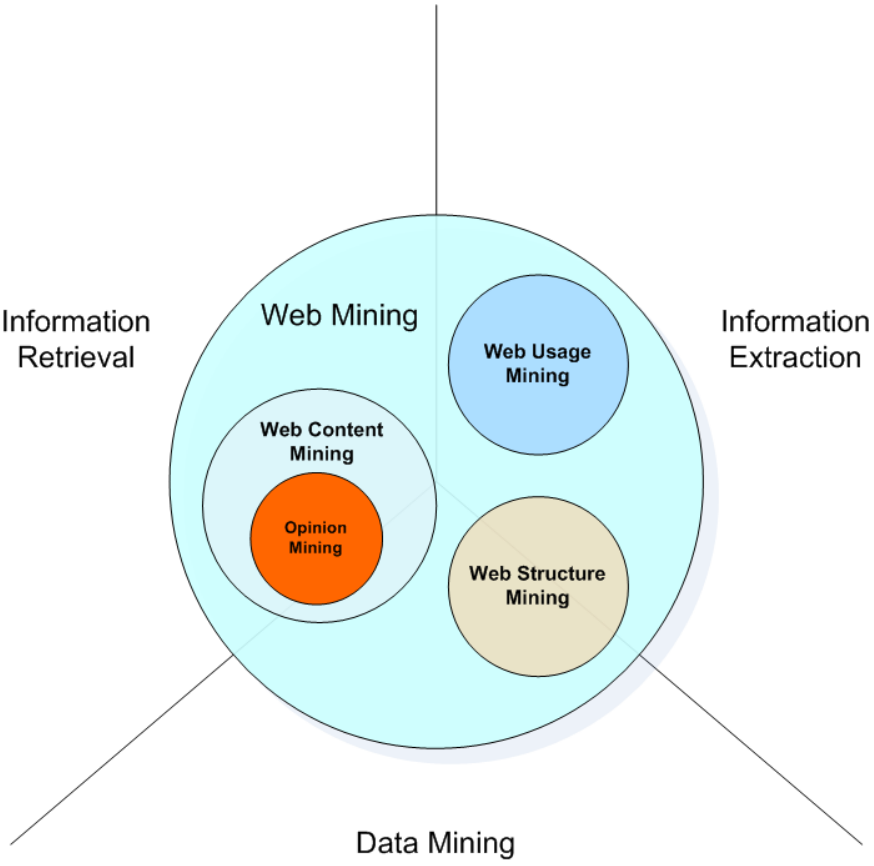

Figure 2.7: Web Mining - Web Mining is not a specific sub-division of classical data mining. Besides the similarity between the names, this field uses techniques and concepts from IR, IE and Data Mining.

2.3.3 Opinion Mining

Opinion Mining is a field of Web Content Mining that aims to find valuable information out of users opinions. Mining opinions on the web is a fairly new subject, and its importance has grown significantly mainly due to the fast growth of e-commerce, blogs and forums.

With the high profits of e-commerce increasing year after year many people had changed the habit of going to a shop for the comfortable virtual shopping. Either ways, searching for useful information on users opinions before purchasing a product, became a common practice for many people. A major problem however, is finding the desired information on them. It is not difficult to find web sites with thousands of reviews for a single product, and thus finding an useful information among them can be a very difficult task. For example, a new customer may be interested in reviews that talk about many features of a certain camera. However, old customers of a specific brand, may be interested only in what people have to say about the auto-focus function of a newly released model.

From the business perspective getting important information out of opinions can represent a good source of advertisement or product feedback. For example, a web site specialized on electronics reviews could place advertisements on their pages based on consumer opinions. For instance, if the majority of users express negative opinions about a given product, the web site could place ads from an alternative product from a competitor. Also, manufacturers can get the feedback of their products to improve their products or services.

An opinion is a personal belief or judgment of a subject, so it is important to differentiate between types of opinion holders. To achieve this differentiation, the term *product review* would refer to an expert opinion (or someone with a privileged status or higher knowledge on the subject) while *users opinions* or *users reviews*, refer to opinions given by common users(in this case,the consumers). An expert opinion is usually far superior in quality, richer in technical details, and goes through all the most relevant aspects of a product. Users and customers usually give opinions with less commitment, and generally what you see are just some pros and cons being discussed. In this work the terms users opinions, users reviews and product opinions are used interchangeably which are different from product review as explained above. This work will not deal

with *product reviews*, thus it will will focus only on mining ordinary customers opinions on the Web where there is no different hierarchy level between opinion holders.

Chapter 3

Related Work

This work and the ones presented on this chapter, emphasizes the use of Opinion Mining (briefly introduced on the last chapter) within the e-commerce context. First due to its importance for the business itself (mainly for marketing strategies) and second to help customers decide between a range of products that better fits their needs (many customers use opinions as the primary and only source of information for guiding purchases). Even though these works are focused on e-commerce, the same principles can be used in many different domains that are likely to admit opinions.

Many of the researches in Opinion Mining have been placing efforts on product features identification and finding opinion sentiment/orientation. In this chapter, the works done by [15] *"Mining and Summarizing Customer Reviews"* and [9] *"A Holistic Lexicon-Based Approach to Opinion Mining"* are going to be exposed with more details than others, with more attention to the last one. The reason why these works were chosen among others is their solution for identifying features automatically and sentiment analysis at an optimal granularity level. They also deal with some cases of sentence context, and the efficiency of their methods could be verified on their evaluative results. Also, both define problems that resemble in many ways the one explored by this work, specially for coping with opinions in an e-commerce context. Finally, an important argument favors the study of [9] with more detail. In [15], sentiment is analyzed at the sentence level, while this approach works reasonably well can hide many important details. In [9], this problem is solved through a fine-grained sentiment analysis done at the feature level.

3.1 Defining Opinion Components in a Opinion Mining Context

The definitions used in this section were proposed in [9], and they summarize important elements that compose an opinon. Some of these definitions are just a natural observation of elements present in opinions, while others refer to definition of problems addressed in [9]. For this reason some of these definitions may not apply to other works as they can differ a lot from their goals as well as from the strategies they employ to achieve them.

Object Model Definition

The main goal of an opinion is to highlight possible strengths and weaknesses about objects under discussion (OuD). Objects can represent a variety of things in the real world, such as products, organizations and persons.

An OuD is defined as a tree and uses the part-of relationship to decompose an object in different components (which in turn can be decomposed by sub-components). An object is associated with the pair *O:(T,A)*, where T is a taxonomy of components (or parts of an object) and possibly sub-components, and A is a set of attributes of O. Like in a tree hierarchy, the components can also have their own set organized. For example, a camera represents the root node and opinions can highlight aspects about a camera attribute as well as attributes of parts of the camera (components). The sentence *"This camera has a great design"* as an example. Here, *design* is an attribute of camera (the root node). On the other hand the sentence *"The battery life is too short"* talks about *battery*, which is a component of *camera* and *life* which is an attribute of *battery* (battery life). An opinion must not necessarily highlight just attributes of objects or components, they can also refer to the object itself. Figure 3.1 illustrates how opinions reference objects.

The next sections are going to use this model to reference opinions as well as target objects. In this chapter more emphasis will be given to the product example which represents a concrete instance of the object model discussed above. Hence, the word *feature* corresponds to both components and attributes, which will also simplify the model by omitting the hierarchy.

Figure 3.1: Opinions highlight strenghts and weaknesses of attributes of an OuD or its sub-components. - Opinions refer to attributes, but they can also refer directly to objects itself

Explicit and Implicit Feature

When a feature f is readibly available in a review R, f is called an **explicit feature**. There are cases where a feature f is not readibly available, in R, therefore it is considered to be an **implicit feature**.

Example 1:

I. "The battery life of this camera is too short"

II. "This camera is too large"

In the first sentence, *battery life* is an **explicit feature**, while in the second one, *size* is an **implicit feature**. *Size* is not mentioned in this sentence, but it is easy to realize that large indicates a negative feature of the size attribute.

Explicit and Implicit Opinion

An explicit opinion on feature f is the one which directly expresses positive or negative aspects of a feature *f*. An implicti opinion on feature f is an objective sentence that implies an opinion.

Example 2:

I. "The picture quality of this camera is amazing."

II. "This earphone broke in two days."

The above example taken from [9] shows that in the first sentence is clear and explicit that the opinion about picture quality is positive. In the second case however, the opinion about the earphone is not explicit, but one can assume that it is negative, based on the sentence context. Identification of implicit features were not addressed on the mentioned works.

3.2 Opinion Mining System Architecture Overview

An opinion mining system proposed by [9] and [15] has some of the following components as illustrated in figure 3.2.

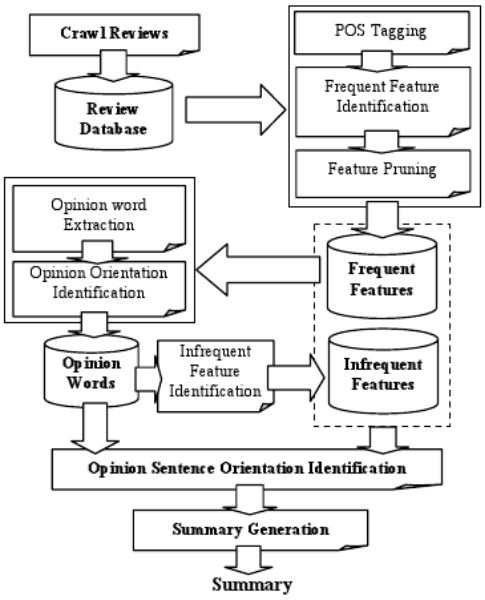

Figure 3.2: Architecure of an opinion mining system - The output of the system is a feature-based summary [15]

The system counts with a crawling module, which first downloads all the reviews and stores them in the database. After that a POS tagger tags all the reviews which will work as hooks for the mining part responsible for finding frequent features. This step is skipped by some systems which employs manual feature annotation as in [21], where ontologies were used to annotate movies features manually. Next, with the

tagged sentences and features identified, opinion words are extracted and their semantic orientation are identified with the help of WordNet. Now with opinion words identified and extracted, the system identifies infrequent features. In the last part of the process the orientation of each sentence is identified, and a summary is generated. The next sections will discuss some of these steps in detail.

3.2.1 Part-of-Speech Tagging

In [15] and [9] a Part-of-Speech tagger (POS tagger) was used to define boundaries (split opinions into sentences) and to produce for each word a given part-of-speech as shown in section 2.2.3. The reason why opinions are split into sentences is basically to achieve a finer granularity as many discussed aspects may reside in different sentences that compose the whole text. Later on it will be discussed the optimal granularity level to analyze opinions.

The tagged sentences produced by the NLProcessor in this step, will play a very important role for the rest of the system. In feature identification, a data mining system will depend on the noun or noun phrases (two up to three neighbor nouns in a sentence) generated on this step to produce a number of *frequent features*. Also, the classification of sentiment will depend on the words classified both as adjectives and adverbs in this step to produce a set of possible *opinion words*.

Opinion Word and Opinion Sentence

An *opinion word* is a term used by [15] and [9] to reference a word that usually qualifies an object or an attribute of this object. They are usually adjectives and adverbs, but they can also be nouns and verbs. An *opinion sentence* is a sentence which holds at least one reference to the object (that could be the object itself or any attribute of the object) and also includes one or more opinion words. The sentences *"I bought this camera last year. Since then I have been very happy with its image quality."*. Here, the first sentence will be discarded and will be not further analyzed as no opinion word is found. The second sentence satisfies the definition of an opinion sentence as *happy* is a opinion word and *image quality* is a camera feature.

3.3 Feature Identification

Feature identification is the process used to deduce possible product features out of the tagged texts generated by the last step. Both [15] and [9] use some heuristics to narrow words that are more likely to be a feature inside a sentence. Normally, the part-of-speech responsible for giving names to entities of the real world are *nouns*, in this case a noun gives name to the product and its features (i.e zoom, battery life, image quality, etc.). In these works, they define two categories of features, *frequent features* and *infrequent features*.

In [21] *"Ontology Based Opinion Mining for Movie Reviews"*, an ontology based approach was used to extract features from opinions. In they work, they experimented it with movies reviews, where they identified sentences containing the ontology terminologies.

Here it is important to differentiate between the two approaches with their pros and cons. In [15] and [9], feature identification is performed automatically. The great advantage of this method is to perform the whole process automatically, with minimal human intervention. The biggest downside however is that the output (the frequent features) depend a lot on the number of opinions being analyzed. Also, there is no guarantee that a frequent feature found by the system is actually a real feature. In [21] and other works where features were manually annotated, the advantage is that the system will always identify real features, being frequent or not. This will only depend on the correctness of the previously made annotation. The great downside however, is that a great number of annotations have to be made. They may not be only specific to categories (such as digital cameras, video games, celular phones), but they could be even more specific such as models of a specific brand (Nikon P90, Nikod D5000, etc). That would make the annotation of features a very hard work. Also, people may comment on the lack of features of a given product, or they may use different words to refer to the same feature which a system with manual annotated features will fail to recognize.

Given the short comparison between the different approaches above, the method explored in [15] and [9] will be further discussed, as it needs a minimal human intervention to perform its task reasonably well which could be later improved by other methods.

3.3.1 Frequent Features Identification

In, [15] and [9] the proposed system extract only nouns or noun phrases (explicit possible features) from the text. In this step, the extracted nouns are called *candidate features*. Then an association mining algorithm finds all the frequent itemsets, which are the set of frequent features (those ones that many users write about). The idea behind this technique is that features that appear on many opinions have more chance to be relevant, and therefore, more likely to be actually a real product feature. The Apriori algorithm [2] was used to generate the set of frequent itemsets. However, for this task there was no need of finding association rules among items, therefore only the part of the algorithm that finds frequent itemsets was interesting for these works. In [9] and [15], a minimum support of 1% was used.

3.3.2 Infrequent Features Identification

A very simple heuristic was used in [15] to discover possible infrequent features (the ones referenced by a small number of people), as the association mining is unable to identify them.

Example 3:

I. "The pictures are absolutely amazing."

II. "The software that comes with it is amazing."

In the above example, the two sentences have one opinion word in common: *amazing*. Because an opinion word could be used to describe more than one object, these opinion words are used to look for features which couldn't be found in the step described on 3.3.1. The infrequent features are extracted as illustrated in figure 3.3.

```
for each sentence in the review database
  if (it contains no frequent feature but one or more opinion
      words)
    { find the nearest noun/noun phrase around the opinion
      word. The noun/noun phrase is stored in the feature
      set as an infrequent feature. }
```

Figure 3.3: Infrequent feature extraction

3.4 Opinion Sentiment Analysis

Sentiment classification or sentiment analysis is an area of study that aims to classify the sentiment encoded by texts as shown in the following example:

Example 4:

I. The girl is angry → negative

II. The sun is absolutely beautiful today → positive

The word *sentiment* is a synonym for *polarity* and both are widely used to describe the orientation of texts, sentences and words as in example 4. This work will deal with sentiment classification of texts represented by users opinions, therefore the name *opinion sentiment*.

The analysis of sentiment can be performed on several levels of granularity (words, sentences, texts and documents). For many applications, classifying the sentiment of documents as a whole is sufficient, for others a finer level of granularity might be necessary.

The work done by [18] and [22] classifies each user opinion as a whole. In [15] and [3] sentiment classification is done at the sentence level. In [9], each feature within a opinion has a sentiment associated. The reason why this last approach is preferable than the others, is easy to realize through a simple observation. To illustrate it, think about a specialized web site for cameras, where customers can write their opinions about a certain product, as illustrated in figure 3.4. An opinion holder writing his opinion, could call attention for both negative and positive aspects of a certain product, all within the same text (opinion). Also, the approach of splitting the whole opinion into sentences and finding the sentiment of each of them may still not suffice. For example the sentence: *"I like my camera and the 24x zoom, but i think the battery life is too short"*. Here it is easy to realize that the sentence is "more positive than negative", however that would still hide one negative aspect of the camera under discussion. This may represent a very important piece of information, which can be obfuscated by classifying the whole sentence as positive. For this reason the method explored by [9] achieves an optimal granularity level as it treats each attribute of an OuD with the necessary details. The next subsection 3.4.1 presents a method explored by [9] and [15] to find the orientation of opinion words.

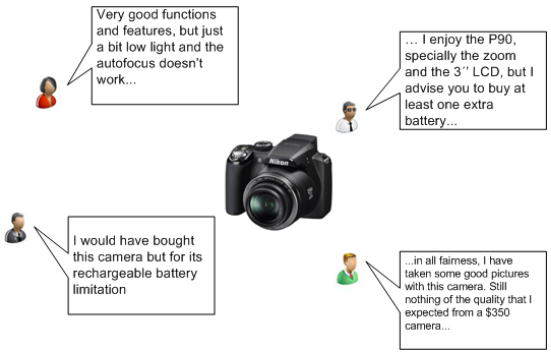

Figure 3.4: Users opinions for a digital camera - The nature of customers opinions show that the same sentence can share both positive and negative aspects about products features.

3.4.1 Identification of Word Sentiment

Opinion words encode an emotional state, which can be desirable or undesirable. Opinion words that encode desirable states (beautiful, nice, happy, awesome) have a positive orientation, while the ones that encode undesirable state (bad, terrible, disappointing) have a negative orientation. As already discussed in 3.2.1, opinion words can belong to many syntactic groups, however they are usually adjectives and adverbs.

In, [15] and [9] a simple and effective solution was proposed by the authors to find the orientation of opinion words. The authors used a seed list with some adjectives and their respective orientation annotated. The idea is to use Wordnet (an online lexical reference system that organize words in synonym sets, called *synsets*), to search for words found in opinions, and let the list grow with the new found words.

In Wordnet, adjectives are organized as bipolar clusters, as illustrated in figure 3.5. The cluster fast/slow consists of two half clusters. *Fast* and its antonym *slow* are called *head synsets*. Each head synset has *satellite synsets* associated with it, which are meanings for the corresponding head synset. Also, the dashed arrow in figure 3.5 represents the association of fast with its antonym slow.

For each new found word (which is yet not in the seed list), the system searches in WordNet for possible synonyms. If any found synonym has a match in the seed list and because synonyms are different words with the same meaning, the system includes this word in the list giving it the same orientation as the synonym in the seed list. If no synonym is found, then the system searches for an antonym. If an antonym exists

and it has a match in the seed list, the new found word is included. However because antonyms have an opposite meaning the same rule is applied to the orientation which will also has the opposite orientation as the match found in the seed list. During this process the seed list will grow. Words with no match may fall in two different cases: (1) They have no match in the seed list, and therefore they should be later manually annotated. (2) The word is context dependent, therefore the part of the system which handles context dependent words will decide its orientation.

This process can be better noticed in figure 3.7. In their work, the seed list started with 30 adjectives, with positive adjectives(great, fantastic, cool) and negative adjectives (bad, dull).

3.4.2 Determining Sentiment of Opinions at the Sentence Level

As discussed, an opinion can be analyzed at different levels of granularity. In this work, the approaches used to analyze the opinion as a whole are going to be discarded as in [18] and [22]. The reasons for this decision was already discussed in **??**. Therefore, the two remaining approaches with respect to their granularity, are going to be exposed with more details. Figure 3.6 shows a pseuco-code which aims to find the sentiment of opinions at the sentence level. The next section, will analyze the sentiment of an opinion at the feature level, as proposed by [9], which is of greater importance for this work.

Negation Rules

A negation word such as *no*, *not* and *never* and also some words that follow patterns such as *"stop" + "vb-ing"*, *"quit" + "vb-ing"* and *"cease" + "vb-ing"* change the orientation of opinion words in the following way:

I. Negation Negative → Positive
II. Negation Positive → Negative
III. Negation Neutral → Negative

Some examples for each negation rule defined above:

I. "no problem"
II. "not good"
III. "does not work"

TOO Rules

The word *"too"* is usually used to give a negative connotation if found before adjectives. The idea is to apply this rule to words which have orientation dependent on context.

I. "The battery life lasts long"
II. "The initialization time takes too long."

Therefore whenever a *too* word is found before an opinion word, the orientation of the opinion should be negative.

3.4.3 Determining Opinion Sentiment at the Feature Level

In [9], after identifying all the opinion words for a given feature, the system calculates the opinion orientation for each feature using the following equation:

$$Score(f) = \sum_{w_i : w_i \in S \wedge w_i \in V} \frac{w_i.SO}{dis(w_i, f)}$$

S is a sentence with a set of features

w_i is an opinion word

V is the set of all opinion words(including idioms)

w_i.SO is the semantic orientation of opinion word w_i

$dis(w_i, f)$ is the distance between feature f and opinion word w_i in the sentence s

For a sentence s that contains a set of features and for each feature f the above orientation score is computed. A positive word is assigned the orientation +1, and a negative one is assigned -1. In the above equation w_i is an opinion word, V is the set of all opinion words (including idioms) and s is the sentence that contains the feature *f*, $dis(w_i, f)$ is the distance between feature f and opinion word w_i in the sentence *s*. The reason for the multiplicative inverse in the formula is to give low weights to opinion words that are far away from the feature *f*. The pseudocode of figure 3.7 was used to find the opinion orientation at the feature level.

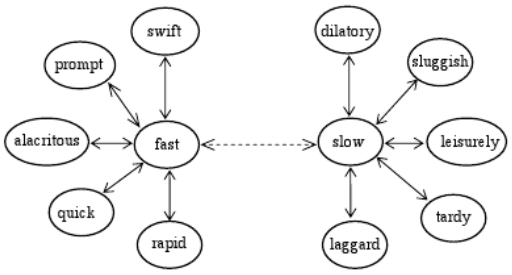

Figure 3.5: Bipolar adjective structure.The solid arrows mean "similar", and dashed one "antonym"[15]

```
1.    Procedure SentenceOrietation()
2.    begin
3.       for each opinion sentence s_i
4.       begin
5.          orientation = 0;
6.          for each opinion word op in s_i
7.             orientation += wordOrientation(op, s_i);
8.             /*Positive = 1, Negative = -1, Neutral = 0*/
9.          if (orientation > 0)  s_i's orientation = Positive;
10.         else if (orientation < 0) s_i's orientation = Negative;
11.               else {
12.                 for each feature f in s_i
13.                    orientation +=
14.                    wordOrientation(f's effective opinion, s_i);
15.               if (orientation > 0)
16.                  s_i's orientation =  Positive;
17.               else if (orientation < 0)
18.                        s_i's orientation = Negative;
19.                     else s_i's orientation = s_i-1's orientation;
20.            }
21.      endfor;
22.   end

1.    Procedure wordOrientation(word, sentence)
2.    begin
3.       orientation = orientation of word in seed_list;
4.       If (there is NEGATION_WORD appears closely
            around word in sentence)
5.          orientation = Opposite(orientation);
6.    end
```

Figure 3.6: Predicting the orientation of opinion sentences [15]

```
1.  Algorithm OpinionOrietation()
2.    for each sentence s_i that contains a set of features do
3.      features = features contained in s_i;
4.      for each feature f_j in features do
5.        orientation = 0;
6.        if feature f_j is in the "but" clause then
7.          orientation = apply the "but" clause rule
8.        else remove "but" clause from s_i if it exists;
9.          for each unmarked opinion word ow in s_i do
10.             // ow can be a TOO word or Negation word as well
11.           orientation += wordOrientation(ow, f_j, s_i);
12.         endfor
13.       endif
14.       if orientation > 0 then
15.         f_j's orientation in s_i = 1
16.       else  if orientation < 0 then
17.               f_j's orientation in s_i = -1
18.           else
19.               f_j's orientation in s_i = 0
20.           endif
21.       endif
22.       if f_j is an adjective then
23.         (f_j).orientation += f_j's orientation in s_i;
24.       else  let o_ij is the nearest adjective word to f_j, in s_i;
25.             (f_j, o_ij).orientation += f_j's orientation in s_i;
26.       endif
27.     endfor
28.   endfor;
      // Context dependent opinion words handling
29.   for each f_j with orientation = 0 in sentence s_i do
30.     if f_j is an adjective then
31.       f_j's orientation in s_i = (f_j).orientation
32.     else  // synonym and antonym rule should be applied too
33.       let o_ij is the nearest opinion word to f_j, in s_i;
34.       if (f_j, o_ij) exists then
35.         f_j's orientation in s_i = (f_j, o_ij).orientation
36.       endif
37.     endif
38.     if f_j's orientation in s_i = 0 then
39.       f_j's orientation in s_i = apply inter-sentence
                conjunction rule
40.     endif
41.   endfor
```

Figure 3.7: Predicting the orientation of opinions on product features [9]

Chapter 4

Requirements

This work proposes an opinion mining system which provides users a quality feature-based summary, out of product opinions from ordinary customers on the web. Therefore the system must achieve two important goals: (1) Gather opinions from different sources on the web. (2) Prepare a feature-based summary from the opinions retrieved in (1), which consists of: product name, the number of opinions analyzed and each identified feature (either using the manual or the automatic approach). Then, for each found identified feature, the system must classify the sentiment associated with each of them (either negative or positive), along with text fragments where these classified features were mentioned.

4.1 Intended Audience

Basically, three types of stakeholders (people or organizations which are interested in the system behavior) are identified for the proposed OMS. From the system perspective it is important to differentiate between active stakeholders (also known as the *users* of the system) from the ones which are just interested on the output generated by the system, but that do not directly interact with it (for simplicity called simply *stakeholders*).

Users of the system are illustrated in figure 4.1. A marketing analyst represents the e-commerce sector while a customer spokesman represents the interests of manufacturers within the system. Web site administrator represent owners of independent

Table 4.1: Stakeholders of the system and their goals.

Stakeholder	Example	Goals
Specialized web sites	CNET, Epinions.com	Help customers by guiding product purchases
E-commerce	Amazon.com, OTTO.de	Analyze opinions to assist marketing strategies
Manufacturer	Nikon, Canon	Analyze opinions of customers to help future product improvements

speciliazied web sites, blogs, or forums. Customers are ordinary internet users, which have public access to the services of the system.

4.2 Use Case Analysis

The use case diagram of figure 4.2 shows how the system work to help each actor to achieve their goals. Next subsection will present a detailed description of each individual use case. From these scenarios a better understanding of the system helps to find functional and non-functional requirements, which will be presented on sections **??** and 4.4.

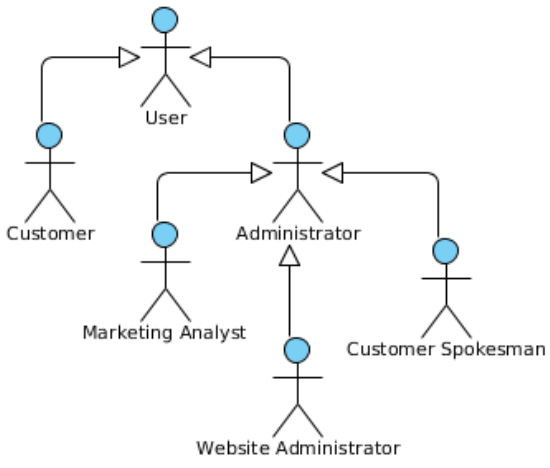

Figure 4.1: Users of the system with respect to their roles.

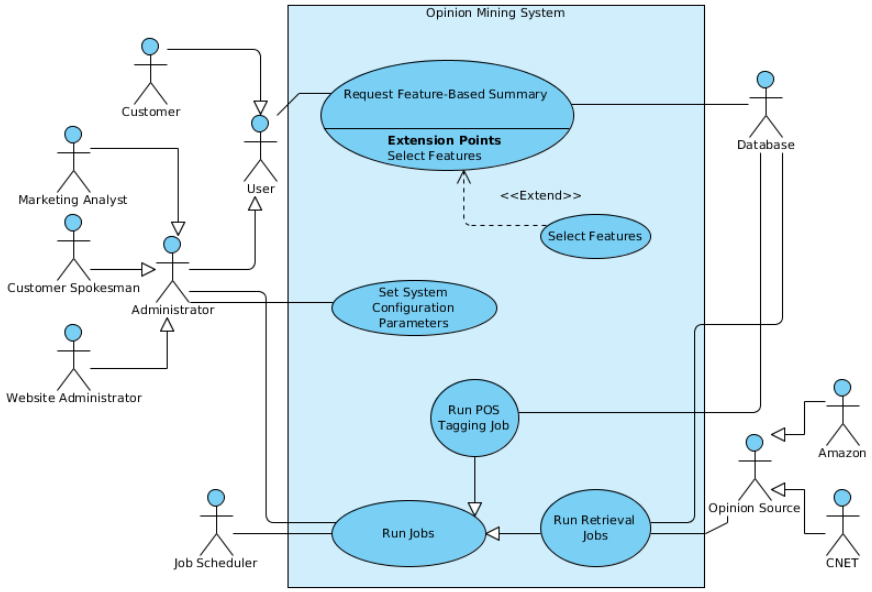

Figure 4.2: OMS use case diagram.

4.2.1 Use Case Description

Request Opinion Summary

Use Case Element	Description
Use Case Description	The user requests a feature-based summary
Primary Actor	Customer / Marketing Analyst / Customer Spokesman / Web admin
Extension Points	Select Features
Precondition	None
Trigger	User input
Basic Flow	1. User enters product name. 2. The system queries for available opinions on local database. 3. The system generates the feature-based summary. 4. The system replies to the user.
Alternate Flow	1. User enter product name and list of features. 2. The system does not find opinions on local database. (a) Store user request. (b) Inform user that service could not be attended.

Run Jobs

Use Case Element	Description
Use Case Description	Scheduler run queued jobs
Primary Actor	Job Scheduler
Extension Points	None
Precondition	There are available jobs in the queue
Trigger	Scheduler
Basic Flow	1. The scheduler triggers the process responsible for running jobs. 2. The system performs the job. 3. The process keeps listening for new incoming jobs.
Alternate Flow	2. The job fails (a) Reschedule job.

Set System Configuration Parameters

Use Case Element	Description
Use Case Description	The administrator set specific parameters for the system
Primary Actor	Administrator
Extension Points	
Precondition	User has been succesfully authenticated
Trigger	Administrator inputs data
Basic Flow	1. Administrator set configuration parameters 2. The system store them persistently
Alternate Flow	

4.3 Functional Requirements

4.3.1 General Operation

The following operations should be performed either as a response of human interaction (user requesting service) or a third-party application requesting it through a special web service interface.

1. The OMS must use automatic feature identification mode if the user provides just the product name.

2. The OMS must use manual feature identification mode if the user provides a list with one or more features.

3. **Automatic Feature Identification Mode**

   3.1 The OMS must identify product features in opinions automatically.

   3.2 The OMS must provide a feature-based summary to the user if the number of available opinions in the database is superior then a minimum number (specified by the administrator).

   3.3 The OMS must inform the user that the request could not be attended, if the number of available opinions to evaluate is insufficient.

4. **Manual Feature Identification Mode**

   4.1 The OMS must ignore the policy of minimum number of opinions to analyze (set by the administrator) and produce a summary with the number of available opinions, regardless of how many of them are available.

   4.2 The OMS must inform the user that the request could not be attended if there are no available opinions.

5. **Job Scheduling**

6. In 3.3 and 4.2 The system must identify if the number of available opinions are insufficient for a given request or if there are available opinions, however they are not yet "pos-tagged" and hence not yet ready to be processed.

7. The OMS shall automatically schedule jobs (either retrieval or pos-tagging jobs) depending on each situation that led to an unattended request.

4.3.2 Setting System Configuration Parameters

The OMS must allow the administrator to change the following configuration parameters:

1. **Set configuration parameters for retrieval tasks**

   1.1 Set value for maximum allowed web scraping operations.

   1.2 Set value for maximum allowed web service operations.

2. **Set configuration parameters for opinion mining algorithm**

   2.1 Set minimum number of opinions for the automatic feature identification mode.

   2.2 Set the minimum threshold number used to prune infrequent features.

3. **Set configuration parameters for job scheduling**

   3.1 The set number of jobs that can run concurrently.

   3.2 Schedule jobs specifying job priority and time of the day.

   3.3 Set the maximum number of opinions processed per pos tagging job.

   3.4 Maximum number of attempts if a job fails.

4.4 Non-Functional Requirements

The functional requirements represent the main goals that the system must achieve. The non-functional requirements represent requirements that should work to assist the application to accomplish its goal.

Interface

The system must provide a web-based as well as a web service interface, which allows both human and third-party applications to interact with the system.

Performance

The system must be able to handle an user request in an acceptable amount of time. Whenever the system can not afford to meet the user's request, it shall schedule the task to occur in parallel (asynchronously), without blocking the flow between user and system.

Scalability

The system must provide ways to retrieve as many as necessary opinions from the sources.

Fault-Tolerance

The system must be able to cope with bad formatted html, network failure and parsing errors. The system shall schedule jobs automatically in face of failures.

Compatibility

The system shall be compatible with the Fedseeko system to seamless integrate the existing platform as a module/plugin.

Interoperability

Other applications can request the services of the system automatically (without direct human interaction with the GUI). The system must be able to interact with other applications using standard technologies.

Chapter 5

Design

This chapter will present the design of the proposed opinion mining system specified on the last chapter. The design of an OMS requires extra attention, specially because these systems may deal with complex algorithms of information extraction, natural language processing combined with heavy text processment (which may consume lots of computational resources) while still suffering from structural problems. For instance, communication problems of any nature between the system and the Internet, would severely affect the sub-system in charge of feeding the local database with opinions. Therefore, non-functional requeriments play an important role to assist the system to accomplish its task.

This chapter is divided as following: section 5.1 will show an overview of the system's architecture. Sections 5.2 to 5.5 will explain the role of each internal component of the proposed OMS in details.

5.1 General Architecture

The general architecture is composed by four main components. These components are called: System Management Module (SMM), Opinion Retrieval Module (ORM), POS Tagging Module (PTM) and the Opinion Mining Module (OMM).

As shown in figure 5.1 the System Management Module is responsible for coordinating users requests and to assign jobs to the Opinion Retrieval and the Pos Tagging Module. A job is defined as a long running task (such as opinion retrieval tasks and

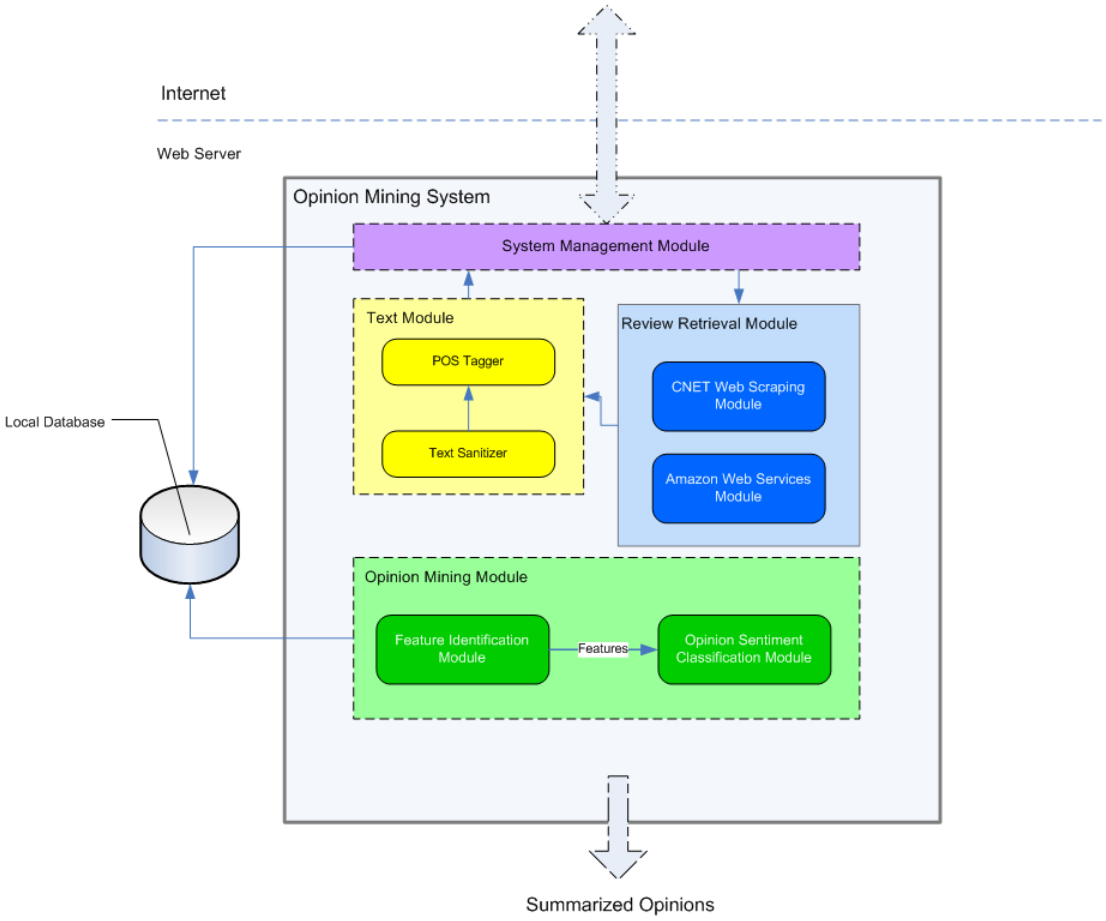

Figure 5.1: The Opinion Mining System Architecture - The Opinion Mining Module is decoupled from the other subsystems by means of a producer-consumer architecture with the database as the common interface.

pos tagging operations), which are required to prepare data before it is suitable to be used by the Opinion Mining Module.

The Opinion Retrieval Module is in charge of gathering opinions on the web and storing them in a local database. Therefore, all the specialized algorithms on how to perform such operations is held by this module.

The POS Tagging Module is responsible for tagging opinions with their part-of-speech. As with the ORM, the operations performed by the PTM are also scheduled by the System Management Module. After, ORM and PTM accomplishing their operations successfully, the opinions are finally ready to feed the Opinion Mining Module.

Finally, the OMM identifies product features out of a set of opinions, classifies the sentiment related to each of them and generates a final output to the user, which consists of each discovered feature and the sentiment related to them (positive and negative). Neutral opinions are omitted.

As shown in figure 5.1 both ORM and OMM do not communicate directly. The output generated by the Retrieval Module (the retrieved opinions) are stored in a database. The ORM, PTM and OMM use the producer and consumer paradigm to decouple them in space and time, thus the database acts as a junction point between these modules. The decision for this design is mainly to achieve process deserialization (where tasks can be performed either in parallel or during different times of the day). The advantage of this architecture is to let the modules perform their jobs asynchronously, which also decouples the long running tasks from the HTTP request/response cycle. Therefore the SMM can serve the user and provide answers in an acceptable amount of time, while still scheduling long running jobs for parallel processment on the background or on later opportunities. Also, to cope with limited computational resources, the SMM allows multiple machines to run these jobs in parallel.

5.2 System Management Module

The System Management Module is the coordinator of all OMS operations. This layer sits beneath the GUI and it is responsible for handling users requests and for assigning jobs to workers. A job is any long running task that needs to run asynchronously. These tasks include the web scraping and the web service operation, as well as the part-of-speech tagging. The SMM transforms any unmet user request in a scheduled

job. Each scheduled job stays in a persistent queue and a separate process is responsible for assigning a worker to a job in the queue. A worker is a system thread or another process responsible for executing a job. Workers are not limited to run locally, and they can also run on multiple distributed machines. The number of workers can be set by the system administrator, and an optimal number to choose should be based on available resources and system workload.

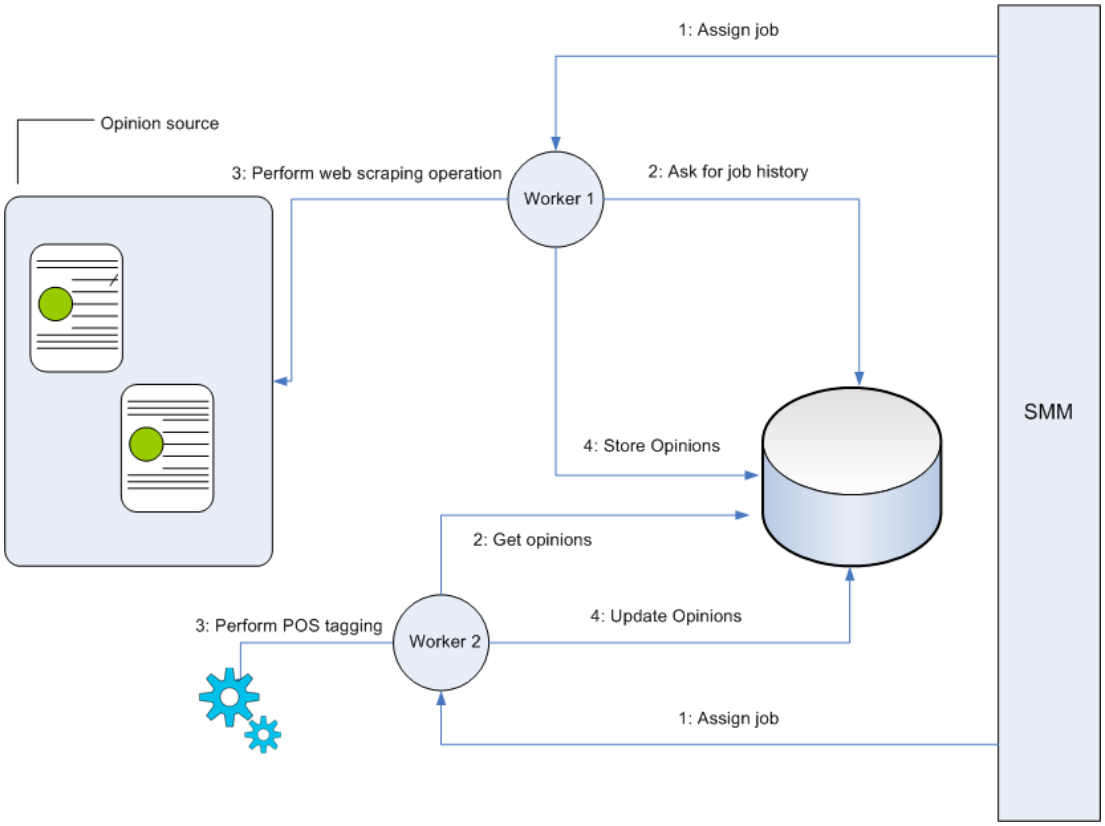

Figure 5.2: The SMM assigns jobs for workers - Jobs are long running tasks such as web scraping and pos-tagging operations

Another important aspect of the SMM is its fault-tolerance with its ability to reschedule failed jobs. As an example the jobs performed by the ORM can suffer from network partition or parsing errors. Thus, the SMM attempts to reschedule jobs immediately after failures or later on a future time (specially useful if a web server is going under a long maintenance).

Finally when assigning a job the system can schedule them based on time of the day and priority). The advantage of it is that jobs may depend on available network bandwidth and computational resources (such as CPU and memory), thus letting these jobs running during specific times of the day (for example at night) where usually resources are more available, could be a good strategy to cope with limited resources.

5.3 Opinion Retrieval Module

The basic unit that feeds an OMS is the opinion itself. Not only is necessary for the system to have a large available number of opinions, which are necessary for generating opinion diversity, but they are also necessary to improve the quality of the output generated by the OMM algorithms . As later will be shown, the number of opinions being analyzed by the system will also directly affect the output generated by the feature identification component, when working on the automatic mode.

To achieve correctness and simplification this work assumes that all the sources of opinions are previously known at design time and therefore discovering new sources of opinions is beyond the scope of this work. To justify this design decision, it is valid to think about the main method used to discover sources on the Internet: keyword searches through search engines. Four main reason make this method not explored on this work. (1) The system would have to perform a search by means of search engines. The first problem refers to an intrinsic issue of currently search engines, as they do not guarantee that the desired information was actually found (they just give the user a clue but there are no guarantees). (2) The system would need a good solution that could decide upon several results and verify each of them to understand which one is relevant, this is by itself already a good research work. (3) Assuming, that the process described on 2 would work with current technologies, this task would be extremely time consuming, and would introduce one more large overhead within the system, which already suffers with time limitations. (4) Current technologies for extracting information from web sites (mainly Web scraping) are domain dependent as discussed on chapter 2 and therefore finding a generalization could be somewhat difficult.

Considering the aforementioned reasons, all the sources of opinion are known at design time. The reason why Amazon.com and CNET.com where chosen among others are not only the large repertoire of customer opinions, but specially in Amazon's case

is the facility to request them through web services. CNET lacks to provide a web service like Amazon.com, however due to their good reputation (which can be easily verified through search engines)[1] their opinions would be indispensable to produce a local repertoire with high quality opinions. It is not the goal of the proposed prototype to explore other sources of opinions, since the two sources already provide enough information for evaluation. The important thing is to clarify the two techniques used by the system in order to extract/request the desired information. In a real application scenario, extending the functionality to other opinion sources can be accomplished using the same design principles employed here, based on the examples used in this work.

Next subsection will show the details of the retrieval operations for each of the mentioned source and how the system is able to scale by maintaining a history of performed actions.

5.3.1 Retrieving Opinion from Sources

Amazon provides the facility to request reviews of a product from their database through web services. Each web service call returns a maximum of five opinions. Through this service is also possible to visualize the total amount of available opinions for a given product. It is important to consider cases where hundreds or even thousands of opinion are available. For 1000 opinions for instance, a total amount of $1000/5 = 200$ web services calls would be produced. This number represents a great overhead especially considering time and network limitations.

With CNET.com the overhead is far bigger when compared to the Amazon case. First, CNET provides no web service facility where the information can be retrieved using web service calls. Without a proper interface to request the information, the web scraping technique described on 2.2.2 would best the system needs. The performance issues of web scraping were already discussed in 2.2.2.

The overhead of this technique is due to the need of requesting whole pages when just a small information, among several other useless information (advertisements, photos, videos, etc) are necessary. Another problem already discussed, was the fact that many web servers may interpret a high number of HTTP GET requests in a short amount of time as an attempt of a DoS attack.

[1] This decision was mainly taken by issuing product names + the word "review" in search engines such as Google and Yahoo, and manually analyzing and verifying the result list

Product	Source	Retrieved Opinions	Available Opinions	Base URL
Nikon P90	CNET	2	16	http://reviews.cnet.com/digital-cameras/nikon-coolpix-p90/4864-6501_7-33520041

Table 5.1: Job description for product "Nikon P90" on CNET

The next subsection 5.2 will expose how the system copes with the need of requesting large number of opinions from the sources. Thus, the system is able to scale with limited resources while still not abusing from rules imposed by the sources.

5.3.2 Opinion Retrieval Jobs

In 5.2 it was discussed how the SMM schedules long running jobs to be performed asynchronously. All the tasks executed by the ORM either a Amazon web service or a CNET web scraping operation are wrapped as a job, which are then scheduled by the SMM.

An important feature of the ORM is the ability to store description of successful accomplished jobs. Each job description is uniquely identified by a product/source pair(e.g Nikon P90/CNET, Nikon D5000/CNET, Nikon P90/Amazon). These jobs descriptions holds important information, such as the total number of available opinions, the total number of opinions retrieved by all operations and some other additional information such as a base URL address (applied only in the web scraping case), used to optimize future operations. Table 5.1 shows how relevant data for a web scraping job description.

Available Opinions

The total number of available opinions helps to manage future operations so as to not allow other operations to be performed if the number of opinions stored in the local database is equal to the total number of opinions available.

Retrieved Opinions

Keeping track of the number of retrieved opinions of similar jobs, gives the system the possibility to split big retrieval jobs into smaller ones, enabling it to scale and to avoid

Review	URL
1	http://reviews.cnet.com/digital-cameras/nikon-coolpix-p90/4864-6501_7-33520041-1.html
2	http://reviews.cnet.com/digital-cameras/nikon-coolpix-p90/4864-6501_7-33520041-2.html
3	http://reviews.cnet.com/digital-cameras/nikon-coolpix-p90/4864-6501_7-33520041-3.html

Table 5.2: CNET's URLs to Nikon P90 camera reviews

too much requests all at once. Also, in the web scraping case prevents the source to interpret it as a DoS attack.

Base URL

This information is used for web scraping operations and it is extremely important to optimize the web scraping process. By storing a base URL the system minimizes the overhead cause by repetitive steps. Figure 5.3 illustrates how this technique minimizes dramatically the interactions between server and the web scraping agent. By recognizing patterns in the URL, it is possible to extract a base URL which is used by future operations to construct new URLs corresponding to specific reviews. The table 5.2 shows cached URLs for the camera "Nikon P90". From the above table is possible to realize that the pattern "*http://reviews.cnet.com/digital-cameras/nikon-coolpix-p90/4864-6501_7-33520041*" is the base URL for the product Nikon P90. Figure 5.3 shows how this information helps to optimize the process.

5.4 POS Tagging Module

This module is responsible for tagging opinions in the local database with their part-of-speech. The major concern here is performance, as these tasks can consume too much computational resources and time. For this reason the SMM is also responsible for coordinating these tasks. Also, for performance reasons, the operations performed by the PTM are decoupled from the mining part, by storing the results in the database.

The tasks performed by the PTM are also wrapped as job which are managed by the SMM. A pos-tagging job queries the database for opinions which were not yet tagged. Here, the administrator of the system can also set a limit of opinions processed

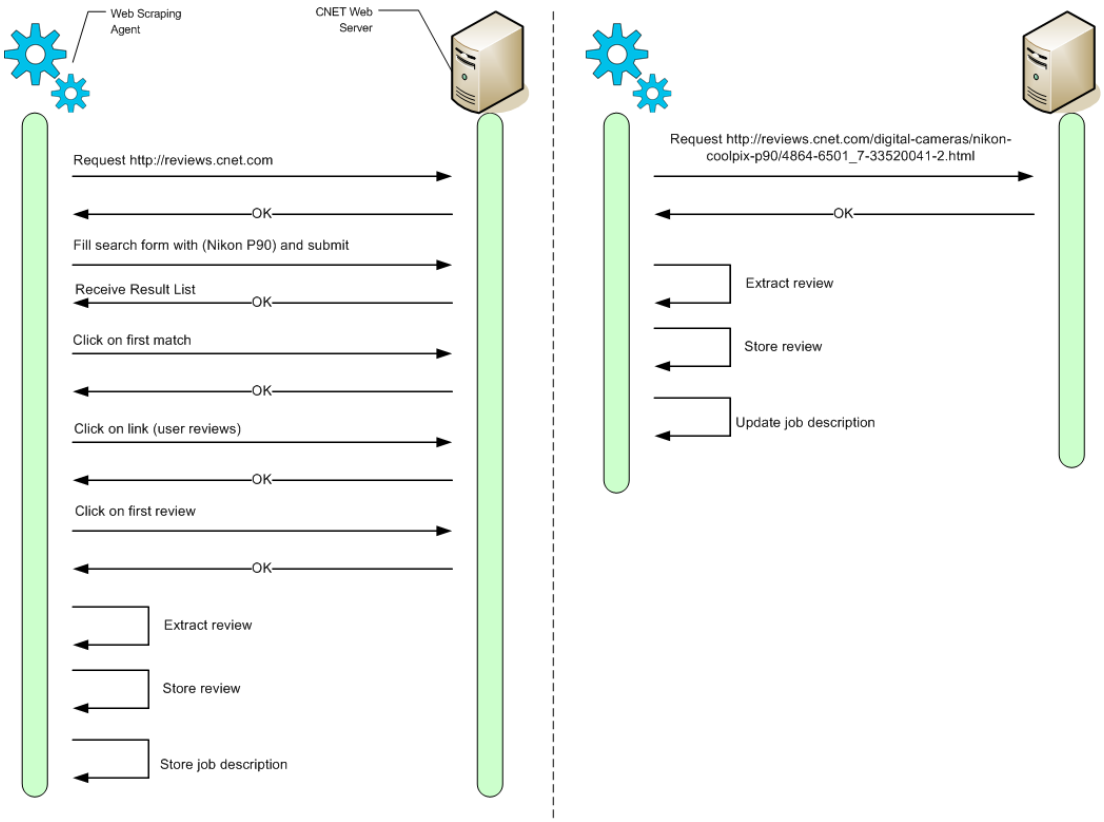

Figure 5.3: Web Scraping agent extracting reviews of Nikon P90 on CNET web site. - The left side of the picture shows a retrieval job performed for the first time. The right side of the picture shows a second request for a retrieval job for the same product. The system shows how the interactions between web scraping agent and web server are minimized by storing the base url address and the number of retrieved opinions.

per time. This is useful to prevent the system to consume too much computational resources. The operations performed by the PTM are detailed in figure 5.2.

5.5 Opinion Mining Module

The Opinion Mining Module (OMM) is the one responsible for adding the real value to the system output. The biggest data transformation happens inside this module as a result of three tasks: feature identification, sentiment classification and summary generation. The first one identifies features for each analyzed opinion, the second analyzes and classifies the sentiment of each feature within a sentence, and finally summary generation aggregates opinions in a meaningful way for better user analysis.

5.5.1 Opinion Model

As discussed in 3.4 there are important reasons to analyze opinions at the feature level. This is due to the inner nature of opinions, which usually highlights positive and negative aspects about an OuD all within the same text. However, increasing the level of granularity also increases the complexity of the problem. The proposed OMS uses the ideas explored in [9] to calculate the orientation of opinions for each detected feature in a sentence. Also, as in [9], the system uses the equation presented in 3.4 to aggregate the sentiment of each opinion word in a sentence to a given feature.

After querying the tokenized opinions from the database with their respective part-of-speech, the OMM splits the text into sentences. The system treats the sentence as the smallest unit where is possible to convey an idea or express an opinion. Sentences may contain one or more words and usually analyzing the relation of these words is sufficient to understand if an opinion is negative, positive or neutral. In some cases however it is necessary to analyze one or more sentences in order to gain knowledge about the orientation of an opinion. To simplify the analysis model, the system treats a sentence as an isolated unit of information. In other words a sentence is an independent unit without any context related to other sentences. Therefore, context is analyzed only within the scope of the sentence itself.

After splitting a text into sentences, each sentence is divided into words. The OMM divides words into one of the following categories: *a*) opinion words *b*) feature words *c*) negation words *d*) orientation inverter words Other words that do not fall in one of

these categories, have no semantic role within the sentence. Opinion words comprises adjectives and some verbs and adverbs. Feature words the set of all nouns/noun phrases within a sentence. Negation words are the ones represented by words like (not, no, n't) and orientation inverter words (e.g *however*, *except for*, *but*, etc) which are used to sinalize a change of opinion. Figure 5.4 shows a composition model for opinions.

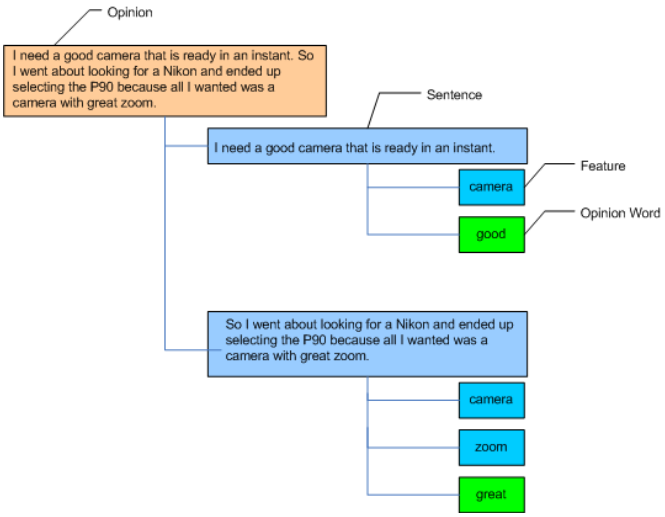

Figure 5.4: A composition model for opinions - A sub-set of words are identified in a sentence: opinion words, feature words, negation words, but-clause words

5.5.2 Automatic Feature Identification

Automatic feature identification is the process of automatically identifying product features from opinions, without apriori knowledge about the product under discussion. This process is also called "Frequent Feature Identification" and it was introduced in [9], due to the way features are identified in a sentence. The idea behind this technique is that real product features are likely to be commented on by many customers, thus the use of the same or similar terms converge to a limited number of words. In [9] and [15] the authors used an association mining algorithm to find the most frequent feature words. This work used a much simpler mechanism and yet very effective. A simple algorithm can count the frequency with which words appear in different opinions, eliminating those less frequent. The end result is a subset of words called "frequent features" that have great chance of actually being a real feature. Occasionally

some words appear as frequent features when they are not. To achieve a good result the system must have an optimal number of analyzed opinions and should have the threshold set accordingly.

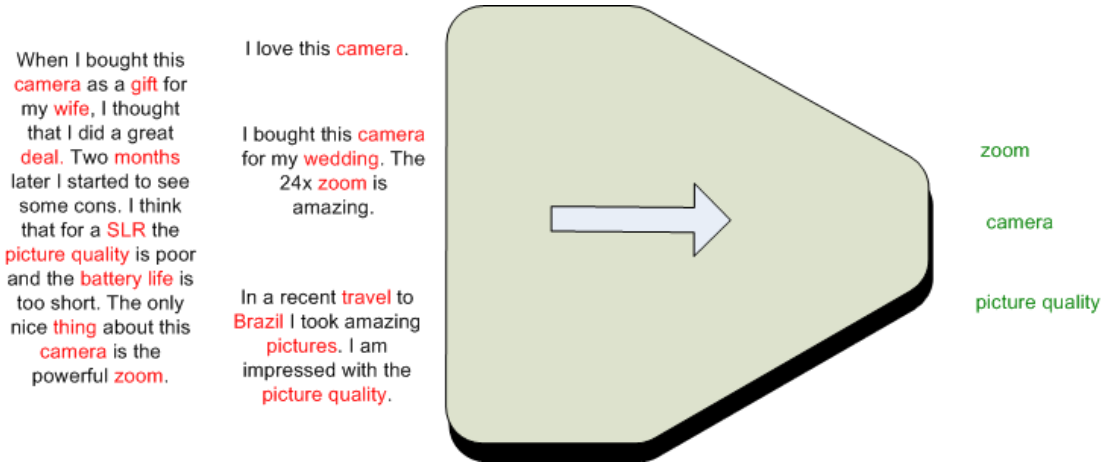

Figure 5.5: Representation of the Wordnet word orientation search algorithm
-

The real advantage about this technique over a manual approach, is that usually it is difficult to recognize all the real features of a product. Also, sometimes users use other words to refer to the same thing. As an example, an user can either say *"My camera is good."* or *"I love the Nikon D5000"*. From the context is easy to realize that both are commenting about the same object. Therefore a manual annotated list with a feature called *"camera"* would fail to recognize Nikon D5000 (the model of the camera) and such opinions would be discarded by the OMM. Still, the system lets users request feature-based summary based only on specified features (manual identification approach).

5.5.3 Word Sentiment Classification

An opinion word by definition encodes an emotional state. Different opinion words can encode different sentiment intensity levels as studied in 3.4. For the sake of simplicity only discrete values are assumed for each opinion word. It can be either positive, negative or neutral (+1, -1 and 0 respectively). To find the orientation of words a bootstrap process is used to link opinion words to their orientation. A resident file

consisting of annotated opinion words with their respective orientations, is loaded in memory during the system initialization. The system has also the ability to make the list grow in the process by searching non-annotated terms in WordNet.

```
1  method WordOrientation(word, sentence)
2  word_orientation = 0
3  for word w in opinion_word_list do
4   if word == w
5       if apply_negation_rules?(word, sentence)
6         word_orientation = (orientation of w) * -1
7     else
8       word_orientation = (orientation of w)
9     end_if
10   end_if
11 end_for
12 if word_orientation == 0
13    word_orientation = checkOrientationInWordnet(word)
14    if word_orientation != 0
15      if apply_negation_rules?(word, sentence)
16        word_orientation = word_orientation * -1
17      end_if
18      saveWordToOpinionWordList(word)
19    else
20      saveWordtoResidueList(word)
21    end_if
22 end_if
23 return word_orientation
24 end_method
```

Listing 5.1: Searching for word sentiment/orientation

The pseudo-code in **??** shows how this process works. From lines 4 to 11 the system searches if the word passed as a parameter to the method is already in the opinion word list. If a match is found the system checks if negation rules and too rules should be applied. This rules will typically change the orientation of the word based on the sentence context. If negation words are found nearby opinion words, then negation rules are applied. A negation word typically inverts the orientation of opinion words. These rules were described in 3.4.1.

When the program reaches line 12 a word had no match in the opinion word list. Then another routine searches in WordNet for synonyms or antonyms that might be present in the opinion word list. This process is described in details by the flowchart of figure 5.6. Finally on line 18 if the system finds an orientation the word is stored in the opinion word list together with its orientation. If no match is found, the word is added to a list consisting of opinion words with unknown orientation. This list can

be later analyzed by the administrator in order to manually add words to the opinion word list (when this is the case).

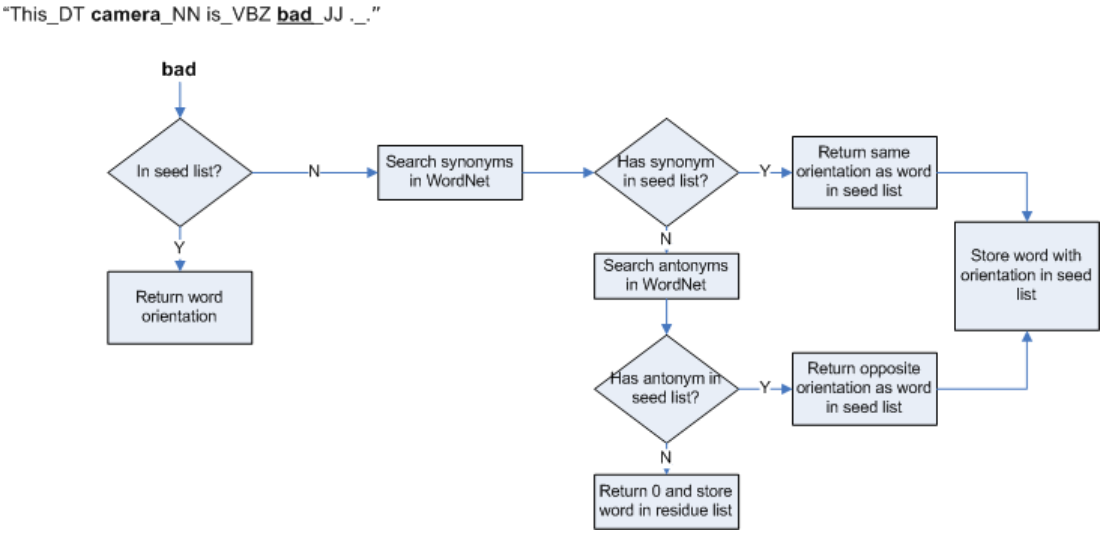

Figure 5.6: Flowchart representing the WordNet word orientation search algorithm -

Chapter 6

Implementation

The concepts discussed in chapters 4 and 5 are realized now as a multiplatform web based opinion mining system called POECS (Platform for Opinion Extraction, Classification and Summarization). This chapter is organized as follows: Section 6.1 will present the technologies used in POECS development, such as programming languages, frameworks and libraries. Section 6.2 to 6.6 will expose the implementation details of each internal module introduces on the last chapter, through object oriented analysis. These sections will include UML classes diagrams as well as concrete realizations of them in Ruby code.

6.1 Technologies Overview

Different technologies were used in the development of POECS. The following considerations concerning the decision about which technologies to use were taken into account:

1. The system must provide strong non-functional requirements guarantees (e.g performance). The system needs guarantees that the non-functional requirements (e.g perfomance) will be adhered to, so that the whole system can work properly. Without such guarantees the system may become infeasible. As an example, the SMM discussed on the last chapter, has to deal with performance and fault-tolerance requirements, and building them from scratch can take a considerable amount of time in developing the prototype. It is wise to consider technologies oriented to agile development, which provides as many as possible "ready-to-use"

components. Thus, more time can be devoted to the development of functional requirements, delegating the responsability of building the whole infrastructure to COTS (Components off-the-shelf). Also many frameworks using the agile development metodology supports the use of tools that promote code debugging and verification, ease of installation of new modules and automation of common tasks. Altogheter, these features contribute largely to leave the programmer only concerned about domain specific problems.

2. An existing platform called Fedseeko used the same core technologies used by POECS, and as discussed in chapter 4, POECS was also intended to be used as a plugin to seamlessly integrate this existing system.

6.1.1 Core Technologies

JRuby (Java implementation of the Ruby programming language) is togheter with Rails (web application framework for Ruby) the core building block used in POECS. Ruby on Rails or simply Rails is a web application framework for Ruby that promotes the Agile web development methodology. Rails uses the MVC archictecture, which separates data, logic and presentation. This architecture improves dramatically code readibility and maintanence. Rails also hide many messy details from the programmer, as for example using an Object Relational Mapping technology known as ActiveRecord, where a database table is wrapped as a object class. Another important feature is that Rails enables automation of tiresome tasks, such as creating databases, migration, etc.

POECS use the technology known as JRuby on Rails which is the combination of the above mentioned technologies. The main advantage of using this technology is the possibility to use Java code with Ruby code togheter. In practice this means that an application can benefit from libraries and frameworks available to any of the programming languages.

6.1.2 Software Components

POECS uses some important COTS which are responsible for building a great deal of the system. As described in 6.1.1, POECS uses Java as well as Ruby libraries. In the Ruby jargon, such libraries or software components are also called *Gems*.

Mechanize

Mechanize is a Gem that aims to automate interaction with web sites. It supports many interactions as accessing urls, submitting web forms, following links, page redirection, store and send cookies. This is a powerful library and togheter with Nokogiri provides all the functionalities needed by the Web Scraping tasks.

Nokogiri

Nokogiri is a powerful HTML & XML parser, with the ability to search documents via XPATH and CSS selectors. It is also used by Mechanize and it is useful to search for information inside documents.

Delayed_Job

Delayed_Job is a Gem which wraps tasks as jobs. A job is a long running task which is persistently stored in a database for later processment. Delayed_Job is also in charge of running them in a separate process, therefore decoupling the application flow from the process responsible for executing these jobs.

Ruby-aaws

Ruby-aaws is a library used to interact with the Amazon web site via the Product Advertising API, by means of web service calls.

Rita.Wordnet

Rita.Wordnet is a Java library used to retrieve information from the WordNet database. Rita.Wordnet is extremelly useful compared to other Wordnet libraries as it simplifies the installation process since WordNet is already embedded on it.

Stanford POS Tagger

Stanford POS Tagger is a Java library responsible for reading texts and assigning each word its corresponding parts-of-speech.

6.2 User Interface

There are two possible ways to request the services of the system. The user can directly interact with it, through a Web based graphical interface or a third party application can request the services through web service calls. The next subsections present each of them.

6.2.1 Web-based User Interface

The Web-base user interface provides a very easy and straight-forward way to interact with POECS. The user equests the service by providing a product name as illustrated on the upper right corner of figure 6.1.

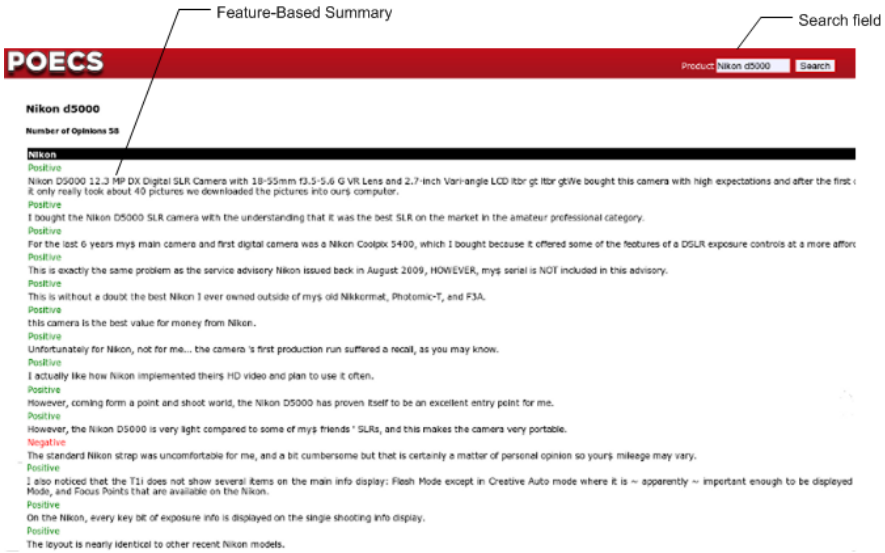

Figure 6.1: Web-based user interface - A user requests the service by entering a product name on the search field and submitting it.

6.2.2 Web Services User Interface

Third-party applications request services through web service calls. Figure 6.2 shows an example of a XML message representing a feature-based summary generated by the system.

```
<product>
  <name>nikon d5000</name>
  <number_of_opinions>58</number_of_opinions>
  <item>
    <feature>Nikon</feature>
    <sentence>
      <text>
        "I bought the Nikon D5000 SLR camera with the
        understanding that it was the best SLR on the
        market in the amateur professional category."
      </text>
      <score>1</score>
    </sentence>
    <sentence>
      <text>
        " For the last 6 years my$ main camera and first
        digital camera was a Nikon Coolpix 5400, which I
        bought because it offered some of the features of
        a DSLR exposure controls at a more affordable
        price."
      </text>
      <score>1</score>
    </sentence>
```

Figure 6.2: A XML message generated by the Web Service Interface - The feature-based summary represented in the XML format

6.3 System Management Module

The System Management Module mediates requests between the user and the system. This is performed by delegating the request to the appropriate internal subsystem responsible for processing it. This mechanism is regulated by a set of policies that can be configured by the system administrator. Each policy operates in a particular area of the system and they are required for the following reasons: (1) Moderate users requests, by limiting the number of allowed web service and web scraping tasks. This procedure is necessary because normally the system will handle the user request dispensing the need of the administrator to check and validate them. (2) To set internal variables of the system which are necessary for the automatic feature identification algorithm. Typically, the output generated by this algorithm will be affected by the total number of opinions analyzed and the minimum support. The administrator can configure these parameters as necessary through the *sysmanager.conf* file.

6.3.1 Request Management

Listing 6.1 , shows how the SMM handles an user request. On line 1 the method *request_summary* assumes that features are *null* by default. In other words, the method will work in the automatic feature identification mode, unless a list of features are

Table 6.1: POECS configuration parameters

ID	Description
1	Maximum number of web scraping operations
2	Default number of web scraping operations
3	Maximum number of allowed web service calls
4	Default number of allowed web service calls
5	Minimun number of opinions to evaluate
6	Minimum support for the frequent feature algorithm

passed as a parameter to the method. Lines 4 to 12 shows a if-statement where the feature-based summary is generated either as using the automatic or the manual mode. Finally, on line 18, if the system is unable to produce a summary, the user request is scheduled by the system. The details about how the system schedules the request will be presented in subsection 6.3.2.

```
1  def request_summary(product_name, features=nil)
2   # @min_opinions / sysmanager.conf file
3   if features.nil? && @opinions.size >= @min_opinions
4    @opinions = Opinion.find(:all, :conditions => "candidate_features IS NOT NULL
         AND tagged_text IS NOT NULL AND query = '# {product_name}'")
5    feature_id = FeatureIdentificator.new(@opinions)
6    features = feature_id.find_frequent_features(@minsup_ff_alg)
7    @op = OpinionSentimentClassifier.new(product_name, features, @opinions)
8    @op.calculate_feature_score
9    @op.summary
10  # The number of opinions in manual mode is irrelevant to the final result
11  elsif features != nil && @opinions.size != 0
12   @opinions = Opinion.find(:all, :conditions => "tagged_text IS NOT NULL AND
         query = '# {product_name}'")
13   @op = OpinionSentimentClassifier.new(product_name, features, @opinions)
14   @op.calculate_feature_score
15   @op.summary
16  else
17   self.schedule_request(product_name)
18  end
19 end
```

Listing 6.1: The SMM handling users requests

6.3.2 Job Scheduling

The SMM uses the Delayed_Job gem to handle the details of scheduling operations. A job is a class that responds to the *perform* method, which will be covered later on section 6.5. Therefore all performed operations can be enclosed in this method, which will be processed later during job execution.

Lines 3 to 5 shows the system scheduling three different jobs. After the execution of these lines, as depicted in listing 6.2 jobs are stored persistently in a MySQL DBMS. Jobs are stored as serialized Ruby objects usign the YAML format, as illustrated in figure 6.3.

id	priority	attempts	handler	last_error	run_at	locked_at	failed_at	locked_by	created_at	updated_at
4	1	0	--- !ruby/object:PosTaggingJob {}	NULL	2010-05-23 18:25:47	2010-05-23 18:25:52	NULL	host:felipe-laptop pid:28942	2010-05-23 18:25:42	2010-05-23 18:25:42
5	1	0	--- !ruby/object:CandidateFeaturesJob {}	NULL	2010-05-23 18:27:12	NULL	NULL	NULL	2010-05-23 18:25:42	2010-05-23 18:25:42
6	0	0	--- !ruby/struct:OpinionRetriever logger: !ruby/o...	NULL	2010-05-26 01:38:45	NULL	NULL	NULL	2010-05-26 01:38:45	2010-05-26 01:38:45

Figure 6.3: Jobs stored persistently in the database - The handler field shows a job serialized as a Ruby object

```
1 def schedule_request(product_name)
2   op = OpinionRetriever.new(product_name, @default_n_ws_calls, "amazon")
3   Delayed::Job.enqueue(op)
4   Delayed::Job.enqueue(PosTaggingJob.new, 1, 30.seconds.from_now)
5   Delayed::Job.enqueue(CandidateFeaturesJob.new, 1, 90.seconds.from_now)
6 end
```

Listing 6.2: Scheduling jobs through the Delayed_Job Gem

6.3.3 Running Jobs

The execution of jobs are also handled by the Delayed_Job gem. This functionality is given by a bundled rake task (a software build tool for Ruby) which is reponsible for creating a new process which listens for the job queue. When running the jobs, the process deserializes the object and perform its operations. Failures are also handled by this process through job rescheduling. The configuration of rescheduling parameters can be performed through a specific file.

```
1 jruby -X+O -S rake jobs:work
```

Listing 6.3: Running jobs through rake tasks. The parameters -X+O enable ObjectSpacing in JRuby

6.4 Opinion Retrieval Module

The general architecture of the Opinion Retrieval Module is presented on figure 6.4. Next subsections will present each component in details.

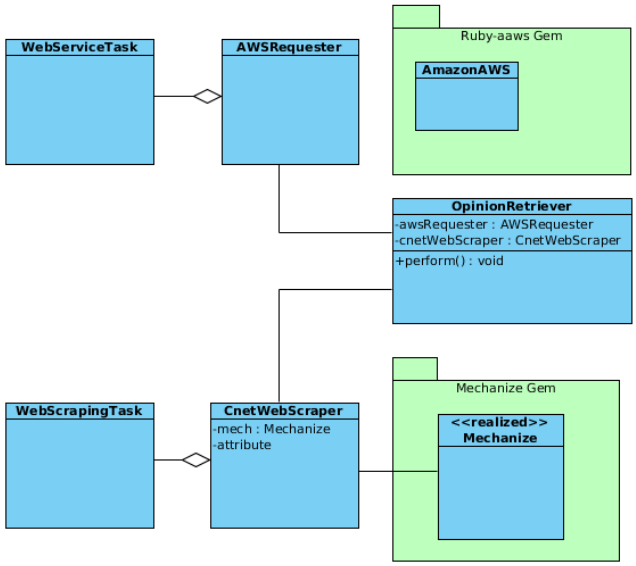

Figure 6.4: The ORM UML class diagram -

6.4.1 Amazon Web Services

The Amazon Web Service Module uses Ruby-aaws to communicate with the Amazon website. The system keeps track of already perfomed retrieval tasks by storing in the database information related to the performed task. This provides a good solution for the system to be able to scale, as with this approach the system can split long jobs into smaller pieces while still maintaiing a history of performed tasks.

id	query	asin	retrieved_pages	next_page	total_pages
1	Nikon P90	F3C4G6	4	5	30

```
1 def lookup_reviews(asin, page_num)
2   # Lookup an item with the unique amazon identification number
3   il = ItemLookup.new('ASIN', {'ItemId' => asin , 'ReviewPage' => page_num})
4   # Creates a response group object.
5   rg = ResponseGroup.new('Reviews')
6   req  = Request.new
```

```
7    # Searches for items.
8    resp = req.search(il, rg)
9    return resp.item_lookup_response[0].items[0].item[0]
10 end
```

Listing 6.4: Requesting reviews from Amazon.com

The code presented in listing 6.4 shows how the system requests reviews from Amazon.com. The *asin* parameter is an unique identifier key used to search for products in Amazon. The object *ItemLookup, ResponseGroup* and *Request* are bundled with the Ruby-aaws Gem and are part of the Amazon::AWS namespace.

6.4.2 CNET Web Scraping

Like Amazon Web Service Module, the CNET Web Scraping Module also searches in the database if existing web scraping tasks are available. The code in listing 6.5 shows how this procedure works. If there were no previous web scraping tasks for the same product, a new task description is generated, which is used by the current or by any later similar job. The idea behind this technique is to avoid repetitive interactions with CNET's web site by generating a common URL which leads directly to the target review.

```
1  def initialize_task(product_name)
2    @task = Scraping.find(:first, :conditions => {:query => product_name})
3    if @task.nil?
4      self.generate_task_description
5    end
6  end
7
8  def submit_search(query)
9    # Select CNET Website
10   page = @mech.get(URL)
11   search_form = page.form(FORM_NAME)
12   search_form[SEARCH_FIELD_NAME] = query
13   # Returns the result list after page submission
14   return @mech.submit(search_form, search_form.buttons.first)
15 end
16
17 def generate_task_description(query)
18   # Creates a new empty Web Scraping task object
19   @task = Scraping.new(:query => query)
20   # Fills the web search form of Cnet and submit it.
21   pre_page = self.submit_search(query)
22   #Page with the results, it takes the first result from the list
23   result = pre_page.search(XPATH_TO_FIRST_LINK_RESULT_PAGE)
```

Product	Source	Retrieved Opinions	Available Opinions	Base URL
Nikon P90	CNET	0	16	http://reviews.cnet.com/digital-cameras/nikon-coolpix-p90/4864-6501_7-33520041

Table 6.2: Job description for product "Nikon P90" on CNET

```
24   #Click on the first match in the result page
25   product_page = Mechanize::Page::Link.new(result[0],@mech,pre_page).click
26   #Gets the total number of available opinions for the given query
27   total = product_page.search(XPATH_TOTAL_REVIEWS)[0].inner_text.match(/out of
         ([0-9]+)/)
28   @task.total_opinions = total[1].to_s.to_i
29   #Click on link "User reviews"
30   result_page = @mech.click product_page.link_with(:text => "User reviews")
31   #Gets the base uri and sets to the first review
32   res = result_page.search(XPATH_FIRST_REVIEW)[0]
33   base_url = Mechanize::Page::Link.new(res,@mech,result_page).click
34   @task.base_url = base_url.uri.to_s
35   @task.save
36 end
```

Listing 6.5: Generating a web scraping task description for CNET

Assuming, a query with the product name "*Nikon P90*", after the execution of line 35 (*@task.save*) a new job description is generated, as shown in table 6.2

If a new retrieval request in CNET web site for product "Nikon P90" is requested, there is a great optimization since the system avoids many interactions with the server.

As in code on line 5 the system searches a product, then clicks on the first occurrence of the result list. Then it extract the total number of available opinions. Then the agent clicks on the link "user reviews" and finally reaches the page containing the target content(review). This same process, as a human perspective is illustrated in figure 6.5.

Figure 6.5 shows a CNET review where only the content of *"Summary"* is extracted. Listing 6.6 shows how the code performs the extraction. The function *next_opinion* browses through different reviews, using the base URL from the task description.

```
1 def next_opinion(op_num)
2   return @task.base_url.sub!(/-\d*.html/,"-"+op_num.to_s+".html")
3 end
4
5 def extract_opinions
6   self.prepare_job
7   i=0
```

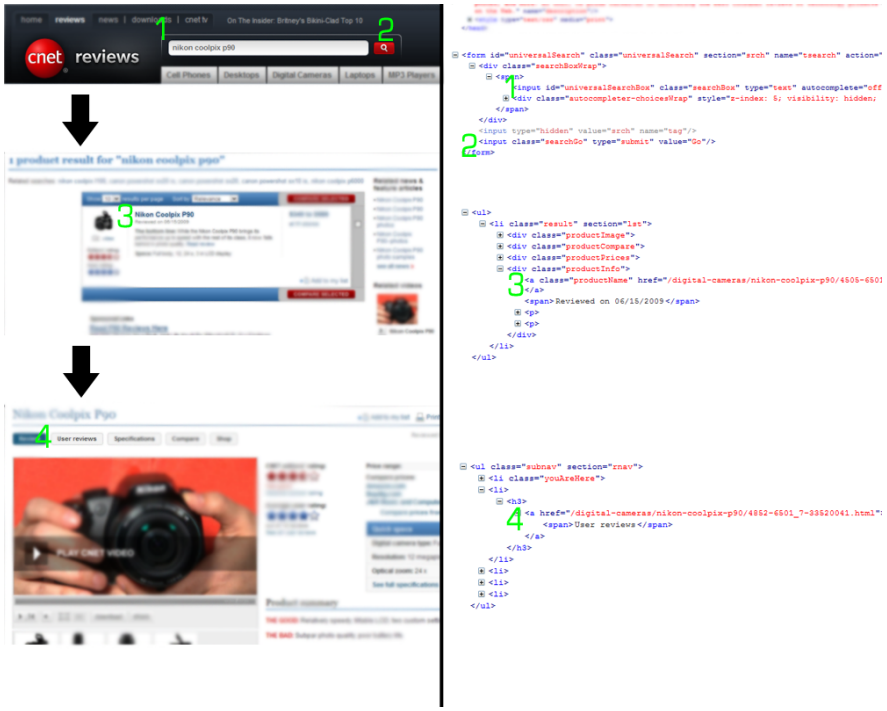

Figure 6.5: Web-based user interface - An user requests the service by entering a product name on the search field and submitting it.

```
8   @num_reviews.times do
9     i = i + 1
10    opinion = Opinion.new
11    #opinion.source_id = source_id
12    opinion.query = @query
13    url = self.next_opinion(@task.scraped_opinions.to_i + i)
14    review_page = @mech.get(url)
15    # Converts a Nokogiri Node to html (to preserve the tags), and
16    # erase \n which if found will make the match fail
17    data = review_page.parser.xpath("//ul[@id='summaryList']//li")[0].to_html.
          gsub("\n", ". ")
18    if data.include?("<b>Summary:</b>")
19      if data.include?("<span class=\"notifyMsg\">")
20        opinion.text = data.to_s.match(/<b>Summary:<\/b>(.*)<span class=/)[1].
              to_s.gsub(%r{</?[^>]+?>}, '')
21      else
22        opinion.text = data.to_s.match(/<b>Summary:<\/b>(.*)<p id=/)[1].to_s.
              gsub(%r{</?[^>]+?>}, '')
23      end
24      @opinions << opinion
25    else
26      skipped_opinion = @task.scraped_opinions.to_i + i
27      @task.skipped_opinions = @task.skipped_opinions.to_s + ":" +
            skipped_opinion.to_s
28    end
29  end
30  @task.scraped_opinions = @task.scraped_opinions.to_i + i
31  return @opinions, @task
32 end
```

Listing 6.6: Delayed_Gem pos tagging job

6.5 POS Tag Module

After retrieving opinions and storing them in the database, the PTM tags the opinions with their part-of-speech. As discussed on the last chapter, the PTM tasks are also wrapped as jobs, which are managed by the SMM. As it happens with the ORM, a PTM job must respond to the *perform* method as shown in listing 6.8.

The *POSTagger class* as shown in 6.7, uses the Java API called Stanford POS Tagger to perform the pos tagging operations. Line 1, "*include Java*" tells JRuby to initialize its Java support. Lines 2 to 3 renames the reference to Java strings to "*JString*" to avoid name collision (the name *String* is already reserverd to the existing String class in Ruby).

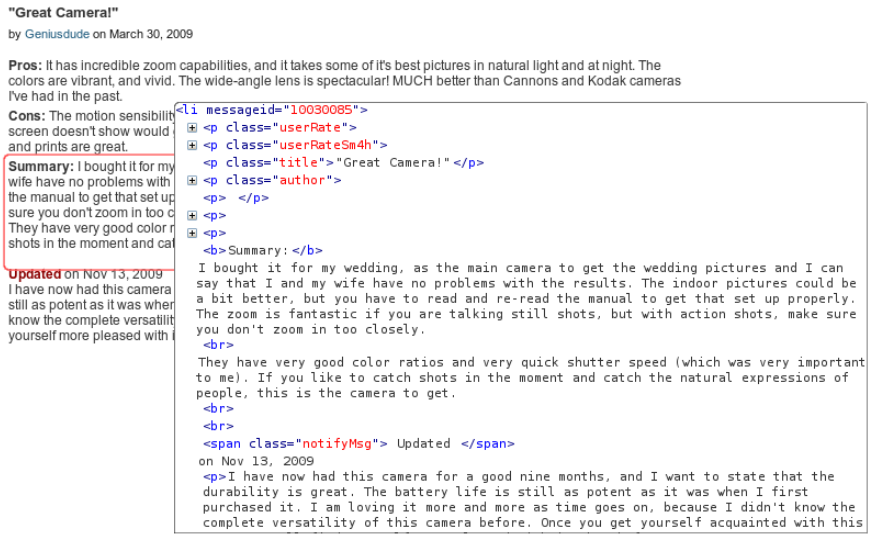

Figure 6.6: A CNET review presentation and the HTML code - The semi-structured document provides a way to narrow the target content. Finer granularity can be achieved with regular expressions

On line 4, the library is linked to the *PosTagger* class. Line 15 shows the instantiation of the Java object *MaxentTagger*. Important to notice is that the object constructor expects a model as a parameter. The model is a corpus with a large number of parsed sentences. The Stanford POS Tagger already provides some of these models in their distribution. Different models can be used depending on the application. In POECS, the English trained model *bidirectional-distsim-wsj-0-18.tagger* was used. The API documentation reports that this is the model with higher accuracy, however with the worst performance.

The Singleton module comes bundled with Ruby, and there is no need for the user to code this design pattern. Through the Singleton module, the object is instantiated only one time and it is guaranteeded that only one version of the object exists. The class can be instantiated through the command *PosTagger.instance* command as shown in 6.8.

A call to the method *tag_text* produces the tagged text. The output can be either in XML format or using a chosen character as separator (the format with an underscore as separator is assumed by default).

```
1 include Java
```

```
2 include_class 'java.lang.String' do |package, name|
3 "JString"
4 end
5 require "stanford-postagger-withModel.jar"
6 include_class "edu.stanford.nlp.tagger.maxent.MaxentTagger"
7 require 'singleton'
8
9 class PosTagger
10
11   attr_reader :tagged_text
12   include Singleton
13
14   def initialize
15     @tagger = MaxentTagger.new("jar:bidirectional-distsim-wsj-0-18.tagger")
16   end
17
18   def tag_text(text)
19     @tagged_text = JString.new(MaxentTagger.tagString(text)).to_s
20     return @tagged_text
21   end
22
23 end
```

Listing 6.7: The POS Tagger class

The code in listing 6.8 shows how the tagging operation is wrapped as a job.

```
1 class PosTaggingJob
2
3   def perform
4     @tagger = PosTagger.instance
5     opinions = Opinion.find(:all, :conditions => "tagged_text IS NULL", :limit
          => 15)
6     opinions.each do |opinion|
7       opinion.text = text
8       opinion.tagged_text =  @tagger.tag_text
9       opinion.save
10    end
11  end
12
13 end
```

Listing 6.8: Delayed$_G$*empostaggingjob*

6.6 Opinion Mining Module

The Opinion Mining Module is represented by the UML Class diagram of figure 6.7.

The class *Opinion* in figure 6.7 shows the realization of the composition model presented on 5.5.1. Each opinion is composed by sentences which in turn are composed

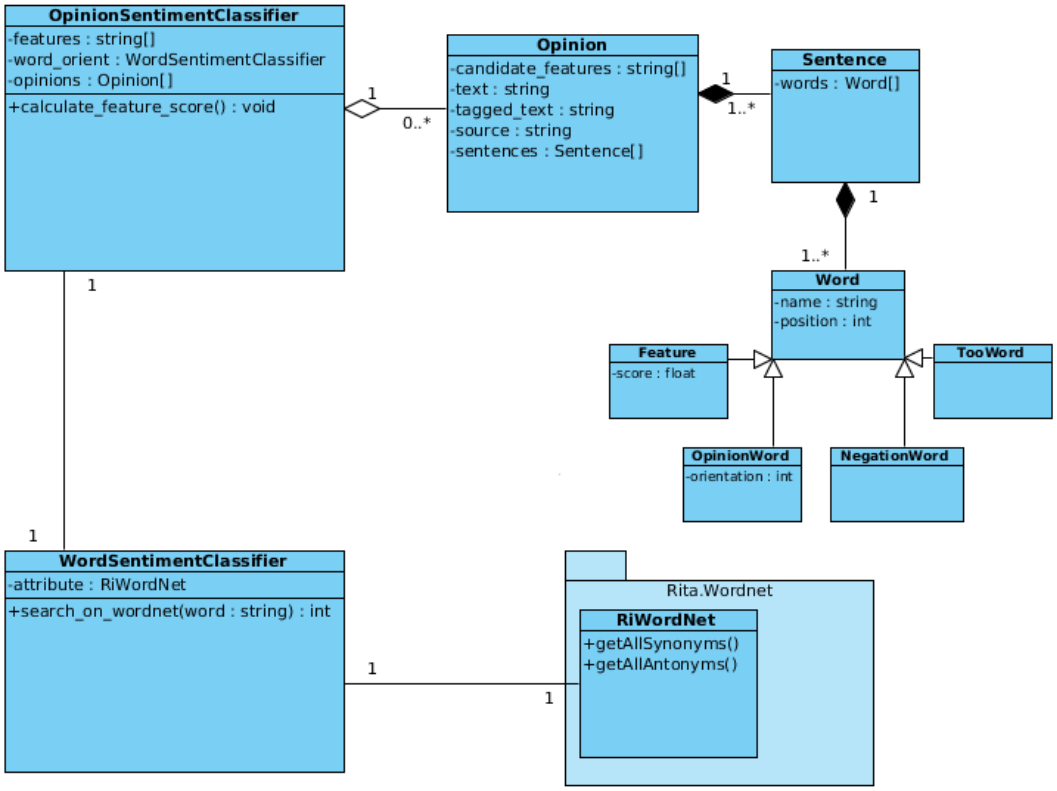

Figure 6.7: Opinion Mining Module UML Class diagram - The class diagram shows the most relevant methods and attributes of each class

by words. Different object types are used to represent different words. For instance, a feature word has different attributes from an opinion word. A feature has a score attribute while an opinion word has an orientation attribute, however they have all in common the attributes *position* and which are inhereted from the class *Word*. Other classes representing negation words, "too" words and orientation inverter words has the same attributes (the ones inhereted from the Word) but the difference between class type helps to identify them. Other words which do not fall in one of these classification are simply words, and their only function in a sentence is to represent points of reference for word distance calculation. The role of the two classes OpinionSentimentClassifier and WordSentimentClassifier are going to be discussed on the next subsections.

6.6.1 Feature-based Summary Generation

The generate_summary is the main function which ties all the other functions performed by other objects, as illustrated in listing **??**. This function search in all the sentences from a collection of opinions, if a feature, from a list of features (*@features*), is available (line 5). Then if a feature is found, the function performs the following: (1) Finds the sentiment of all opinion words in the sentence. (2) Calculate the score for each feature. (3) Generates the feature-based summary. To understand better how the whole process works, each of the mentioned steps are explained in details.

```
1  def generate_summary
2    @features.each do |frequent_feature|
3      @opinions.sentences.each do |sentence|
4        sentence.words.each do |feature|
5          if feature.name.remove_tags == frequent_feature && feature.instance_of?(
               Feature)
6            sentence.words.each do |word|
7              if word.instance_of?(OpinionWord)
8                word.orientation = @word_orienter.get_opinion_word_orientation(
                     word, sentence)
9                if word.orientation == 0
10                 word.orientation = self.get_orientation_from_sentence_context(
                       sentence, word)
11               end
12               feature.score = feature.score + word.orientation/(feature.position
                     - word.position).to_f.abs
13             end
14           end
15           @summary.add_item_to_summary(feature.name.remove_tags, feature.score,
                 sentence.text.remove_tags)
16         end
```

```
17        end
18      end
19    end
20      @word_orienter.save_to_seed_list
21  end
```

Listing 6.9: This function

Finding the Sentiment of Opinion Words

POECS uses a CSV (Comma Separated Value) file to link *opinion words* to their *orientation* as shown in figure 6.8. During the class WordSentimentClassifier instantiation, the content of the seed_list.csv file is loaded in a hash structure (*@opword_list* line) containing the following key/value pair: word→orientation.

```
good,1,bad,-1,well,1,risky,-1,speculative,-1,unsound,-1
hazardous,-1,dangerous,-1,dodgy,-1,quick,1,strong,1,
fantastic,1,dirty,-1,clean,1,marvelous,1,pleased,1
nice,1,powerful,1,super,1,easy,1,terrible,-1,best,1
serious,1,convient,1,happy,1,lightweight,1,wonderful,1,solid,1
real,1,outstanding,1,friendly,1,wrong,-1,excellent,1,compact,1
neat,1,durable,-1,perfect,1,questionable,-1,versatile,1,trustworthy,1
portable,1,amazing,1,impressive,1,obsolete,-1,stunning,1
compatible,1,interesting,1,frustrated,-1,convenient,1,terrific,1
```

Figure 6.8: A CSV file containing words and their orientations - POECS uses a opinion word list of approximately 40 words

Whenever an opinion word is not found on the seed list (opinion word list), the *search_on_wordnet* function is called. This algorithm, searches if the unmatched opinion word has any synonym on the seed list. In this case, if there is a synonym, the opinion word receives the same orientation as its synonym. However, if the opinion word has no synonyms on the seed list, the function tries to find an antonym. If there is a match, the opinion word receives an opposite orientation from its antonym on the seed list. The described process is illustrated by the functions in listings 6.10, 6.11, 6.12.

```
1 def get_opinion_word_orientation(word, sentence)
2   word_orientation = 0
3   @opword_list.each do |op_word, orientation|
4     if op_word == word.name.remove_tags.downcase
5       if self.apply_negation_rules?(word, sentence)
6         word_orientation = orientation.to_i * -1
7         return word_orientation
```

```
8         else
9           word_orientation = orientation.to_i
10          return word_orientation
11        end
12      end
13    end
14    word_orientation = self.search_on_wordnet(word.name.remove_tags.downcase)
15    if word_orientation == 0
16      if self.apply_too_rules?(word, sentence)
17        word_orientation = -1
18        return word_orientation
19      end
20      else
21        if self.apply_negation_rules?(word, sentence)
22          word_orientation = word_orientation * -1
23          return word_orientation
24        end
25    end
26    return word_orientation
27  end
```

Listing 6.10: Requesting reviews from Amazon.com

```
1  def search_on_wordnet(word)
2    word = JString.new(word)
3    word_orientation = 0
4    word_orientation = search_for_synonym(word)
5    if word_orientation == 0
6      word_orientation = search_for_antonym(word)
7    end
8    # Includes new word in the opinion word list or
9    # save to the residue list if no orientation is found
10   if word_orientation != 0
11     @opword_list[word.to_s] = word_orientation
12   else
13     self.save_to_residue_list(word.to_s)
14   end
15   return word_orientation
16 end
```

Listing 6.11: Searching for words in WordNet

```
1  # Search for synonyms in wordnet
2  def search_for_synonym(word)
3   # Searches up to 10 synonyms of "word". The parameter "a" means adjective.
4   synonyms = @wordnet.getAllSynonyms(word, "a", 10)
5     @opword_list.each do |op_word, orientation|
6     if synonyms
7       synonyms.each do |synonym|
8         if synonym == op_word
9           word_orientation = orientation.to_i
```

```
10            return word_orientation
11          end
12        end
13      end
14    end
15  return 0
16 end
17
18 # Search for antonyms in wordnet
19 def search_for_antonym(word)
20  antonyms = @wordnet.getAllAntonyms(word, "a")
21  @opword_list.each do |op_word, orientation|
22   if antonyms
23     antonyms.each do |antonym|
24      if antonym == op_word
25        word_orientation = orientation.to_i * -1
26        return word_orientation
27      end
28    end
29   end
30  end
31  return 0
32 end
```

Listing 6.12: Searching for synonyms and antonyms in WordNet

After discovering the orientation, the function searches if there is any nearby word which could modify the orientation of the opinion word. One such word are the "negation words" which change the orientation of a word if found nearby an opinion word. Unfortunately, it is impossible to define precisely *"nearby"*. The reason why it is difficult to define a boundary, is the lack of rule when building a sentence as shown in 6.9. In POECS, a word distance of 4 was chosen. This number was chosen based on observations during the development phase as most of the time it was realized that negation words, stays near the opinion word. The same principle is used for the "too" words, which usually when togheter with adjectives denotes negative opinion. However, in the "too" words case, the word has to be adjacent to the opinion word(word distance of 1).

```
1 # Apply negation rules if negation words are found nearby
2 # an opinion word
3 def apply_negation_rules?(word, sentence)
4   sentence.words.each do |sword|
5     if sword.instance_of?(NegationWord)
6       if (word.position - sword.position) > 0 && (word.position - sword.
            position) <= negation_word_range
7         return true
```

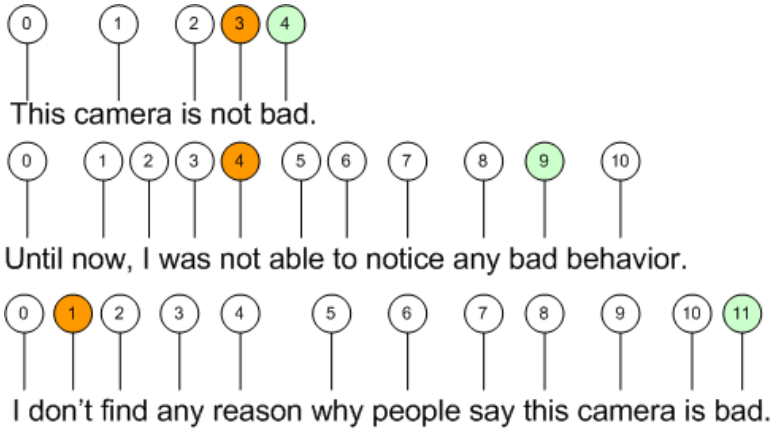

Figure 6.9: Sentences with negation words - This examples show the difficulty to define boundaries for word distances

```
8         end
9       end
10    end
11    return false
12  end
13
14  # "Too" words when togheter with adjectives usually denotes
15  # negative sentiment "This screen is too small"
16  def apply_too_rules?(word, sentence)
17    if sentence.words[word.position-1].instance_of?(TooWord)
18      return true
19    else
20      return false
21    end
22  end
```

Listing 6.13: Negation and too rules applied to word orientation in a sentence

Chapter 7

Evaluation

The evaluation of POECS tested the effectiveness of the sentiment classification and the feature identification algorithm, which are the two core algorithms used to produce the feature-based summary. Also, the system performance was tested using different mining configurations. This chapter is divided as follows: Section 7.1 presents the environment used to test the system. Section 7.2 and 7.3 shows the effectiveness of the sentiment classification and feature identification algorithm respectively. Section 7.4, exposes the system performance using different mining configurations.

7.1 Test Environment

This section describes the test environment (the system configuration) used to evaluate POECS.

Hardware

- AMD Turion(tm) 64 Mobile Technology ML-32
- 1GB RAM

Software

- Ubuntu 9.04 32 bits

Table 7.1: Sample data for evaluation tests

Product	Number of Opinions
Ipod Touch 8GB	120
Nikon D5000	86
Nikon P90	52
Xbox 360	41

- JRuby 1.5.0.RC3 running with ObjectSpace enabled [1]
- Mongrel 1.1.5 and Rails 2.3.8

Sample Data

All the tests in this chapter were performed using the opinions depicted on table **??**. These opinions were both retrieved from Amazon.com and Cnet.com in increasing order (from the first to the last opinion). An opinion sentence is a sentence with at least a feature and one more opinion words.

7.2 Effectiveness of Opinion Sentiment Classification

To evaluate the effectiveness of the sentiment classification algorithm, the orientation associated with each feature in a sentence was analyzed manually in order to achieve a high degree of confidence. The result of this analyses is presented on the graph of figure 7.1 and table 7.2. A correctly classified opinion is either a negative or positive opinion for a given feature, which was correctly identified by the system. All the features were identified automatically, and only the sentences with real features (frequent features) were analyzed. Sentences with candidate features were discarded. The effectiveness of the automatic feature identification algorithm will be discussed later on section 7.3.

It is important to consider the complexity of judging (even for a human) whether an opinion is positive or negative. There are many cases where this analyzes is pretty

[1] ObjectSpace is a Ruby module used for memory management, which is disabled by default in JRuby. One of POECS libraries use this feature which can be enabled by passing the parameters -X+O. It has been reported by the JRuby developing team that ObjectSpace can create a substantial overhead and thus affecting the performance of the system.

Table 7.2: Effectiveness of sentiment classification

Product	Correctly Classified Opinions	Opinion Sentences
Ipod Touch 8GB	79%	673
Nikon D5000	82%	452
Nikon P90	84%	273
Xbox 360	73%	181

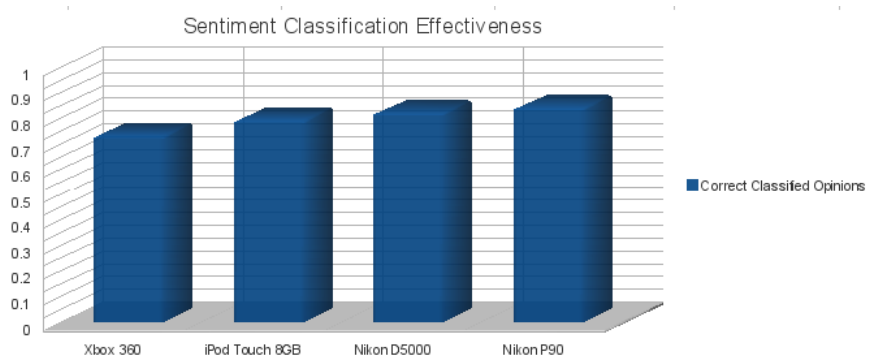

Figure 7.1: Effectiveness of opinion sentiment classification

straightforward, especially when explicit opinion words are used. However, in some cases, in order to understand whether an opinion is negative or positive, a domain specific knowledge is needed. Example 1 illustrates some of these cases:

Example 1:

I. "The iPod battery lasts for 5 hours with music and Wi-Fi turned on."

II. "The device heats very fast."

In sentence I, it is difficult to know whether a battery lasting for *5 hours* under the described conditions is positive or negative. In sentence II, heating fast depends a lot of which device is being analyzed. For instance getting hot fast could represent a negative behaviour of an electronic device such as video games or notebooks. However, if the OuD is a portable heater or an electric oven, heating fast should be a positive behavior.

There are also situations where wrong parts-of-speech are assigned to words and since the POS tagging operation is the foundation of the whole mining process, a wrong tagging classification would directly affect the mining process, as shown in example 2:

Example 2:

I. "This battery is really GOOD."

II. "[...] hard drive of 120 GB which comes with the device."

In sentence I, the word *GOOD* is written in upcase, a common practice among internet users, which aims to emphasize sentiments. The POS tagger however interpretes this word as belonging to the group of proper nouns. Therefore, the mining algorithm will fail to recognize it as an opinion word. In this work, such cases were not covered since they are not common and hence addressing them would produce a great delay for the mining tasks.

A snippet of a customer opinion for Xbox 360 is shown in sentence II. It was detected that the POS tagger would fail to recognize *hard drive* as a noun group (feature). Unlike, the POS tagger interprets the word as follows: *hard*→*adjective* and *drive*→*noun*. Thus, *hard* was taken by the mining algorithm as an opinion word.

Depending on the context, the word *hard* could be either positive or negative which affected results during evaluation. As many customers, commented about the hard drive, this also contributed to place Xbox 360 as the one less effective in sentiment classification as in 7.2.

Other difficult situations to handle include sentences where there is no opinion word available where a straightforward analysis is possible. For example, the sentence taken from the iPod Touch 8GB reviews: *"Apple PLEASE wake up and create an ITunes that is as easy to use as the IPod itself"*. Such cases show how complex can be for an algorithm to decide an opinion orientation, as there is no clear opinion word expressing a sentiment. A human analysis in this case can easily detect the opinion negatively expressed through the words *"wake up"*.

7.2.1 Considerations

The effectiveness mean of 79.5% (for the four analyzed products), shows that most of the customers tend to use simple and straightforward sentences to express their sentiments. The remaining 20.5%, where the system fails to recognize the correct orientation could be minimized with a combination of manual annotations shared among opinions of different domain (different products or products category) as well as domain specific annotations. In [9], for example a list of 1000 idioms was used to help orientation identification. POECS does not have such list available, and therefore many sentences

7.3 Effectiveness of Automatic Feature Identification

The automatic feature identification of POECS, which uses the frequency of words to determine the likelihood of a word to be a real feature, had its effectiveness evaluated as shown in figure 7.2.

The graph of figure 7.2, shows that higher thresholds are more effective when identifying only real features, however less features are also identified, as the infrequent ones are ignored. This shows clearly that the minimum threshold parameter provides a trend between accuracy and the number of detected features.

Pluralization cases such as "cameras" and "camera" were not considered as the same feature. Also, *"photo quality"*, *"image quality"* and *"picture quality"* were considered as three different features. Once more, this issue can be addressed by providing domain

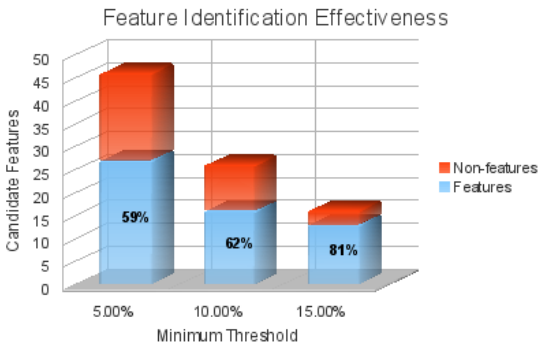

Figure 7.2: Effectiveness of the automatic feature identification algorithm

specific annotations for specific products (cameras, cell phones, notebooks, video-games, etc).

7.3.1 Considerations

The feature identification algorithm can perform pretty well when set correctly. Also, dealing with pluralization cases is not straightforward. For example, while "photo" and "photos" in many cases, could be referring to the same object, it was realized that often the word "camera" and "cameras" do not refer to the same object. While "camera" would normally refer to the OuD, the word "cameras" would refer to concurrent models. For this reason, domain specific knowledge is needed in other to correctly map pluralization cases to one object.

The tagging errors mentioned on the last section, also affects directly the feature identification algorithm. During the evaluation of opinions for the product "iPod Touch 8GB", writing the product with different cases like *"IPOD"* or *"ipod"* would make the system interpret the word as belonging to different parts-of-speech. Hence, the object would not be identified as being a feature.

7.4 System Efficiency

This section shows the trend between effectiveness and performance. To provide accuracy and correctness, performance can be deeply affected, specially because the system has to address also the exceptions cases which are not covered by the general algorithms.

The graphs of figure 7.3 shows that when working with large amount of text which needs complex finer granularity analysis at the word level, any new additional functionality has to deal with performance as this will affect running time and memory consumption.

Two evaluation tests to address performance were made. A total of 120 opinions for Nikon D5000 stored in the local database, were evaluated for different mining scenarios. The mining tasks used the automatic feature identification algorithm which largely affects the performance of the system. The reason is that with smaller thresholds more features are identified and hence more sentences are analyzed. It was performed a measurement of time and memory consumed to complete an user request for different configurations as shown in figures 7.3 and 7.4. In figure 7.3, a threshold of 1%, represented a delay of 350 seconds against 64.25 seconds when using a threshold of 5%.

All the tests in this section were performed with the help of the Benchmark and Ruby-Prof evaluation tools for Ruby.

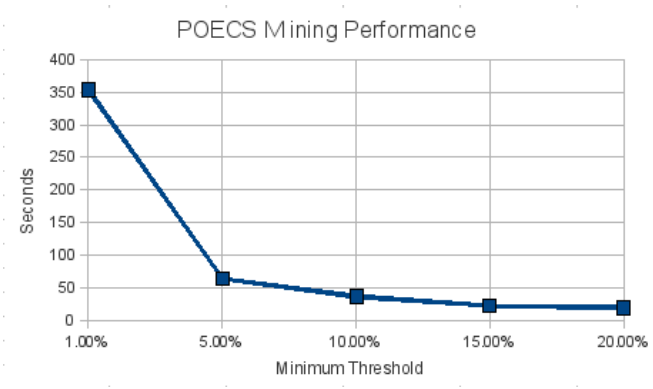

Figure 7.3: Time needed for code execution under different mining scenarios - The X axis "Minimum Threshold" is a configurable parameter of the feature identification algorithm. It represents a minimum frequency which a candidate feature must appear through different opinions in order to be considered a real feature. The Y axis represent the time in seconds needed to generate the feature-based summary

7.4.1 Considerations

When a large number of texts have to be analyzed with the precision needed in opinion mining, performance also needs special attention. This could be noticed through figures 7.3 and 7.4, where under certain configurations, a great amount of time and memory

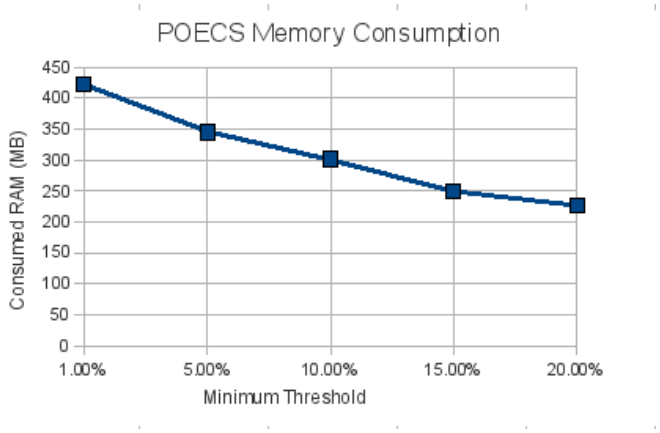

Figure 7.4: Memory consumed by the mining algorithm for different mining scenarios - The value of the mininum threshold parameter affects the number of opinions which are analyzed by the system

were necessary. Considering that this prototype do not address all exceptions cases which as discussed, could be solved with manual annotations, it is clear that any new algorithm that would compose the system in order to enhance its mining ability should take into account the trend of performance. Dealing with many exceptions should represent a big overhead for the system. Sometimes, given the frequency in which certain exceptions occur, it is preferable to favor performance.

Chapter 8

Conclusion

Opinions are a unique type of information which are different from *facts*. The methods for content classification based on ranking (like those used by search engines) are not effective or simply do not accurately depict reality, as one opinion is different from multiple opinions.

During the evaluation of POECS it was possible to see that it is feasible and reliable to build system capable of classifying and organizing opinions through the so called feature-based summary, which resumes the most relevant information for users. However, it is undeniable that a great number of opinions are difficult to classify due to the complexity of the human language.

Evaluation also showed that the system can be more effective when domain specific, using the help of manual annotations to treat common exceptions. A system can therefore combine multiple approaches with the intelligence of automatic algorithms and manual annotations in order to provide a high degree of accuracy.

8.1 Future Work

Many of the works, including this one, have mainly contributed to discover patterns in the language which can be reused in different cases without binding it to a large number of annotated terms. However, the complexity of the language has shown that finding rules for the human languange is not trivial. Opinion mining is a relatively new area of study and very challenging, since it deals primarily with a rather complex system: the human language. Many studies including this one is striving to discover patterns of use

of language that can be reused, widespread and processed by computers. While many annotations are still required, they do not offer the necessary flexibility and often turn out to be very specific domain. There are complex cases as the one presented during evaluation (which the system fails to correctly classify) which remains as challenges for new algorithms which are capable of solving disambiguation problems. As discussed, when one is commenting about a product, a word can have different meanings or can refer to different objects. Solving this problem is a good challenge for new algorithms in Opinion Mining. Also, understanding the semantics of text in a more intelligent way is necessary. A good direction would be methods which have global knowledge of opinions dependent on complex contexts, which can use this information later to help solving context problems in any local analysis (at the sentence or word level).

References

[1] Horacio Saggion Kalina Bontcheva Christian Leibold Adam Funk, Yaoyong Li. Opinion analysis for business intelligence applications. *OBI'08*, 2009.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. 1994.

[3] Oren Etzioni Ana-Maria Popescu. Extracting product features and opinions from reviews. *EMNLP-05*, 2005. 34

[4] Vadim Tkachenko Jeremy Zawodny D. Arjen Lentz Derek J. Balling Baron Schwartz, Peter Zaitsev. *High Performance MySQL: Optimization, Backups, Replication, and More*. O'Reilly Media, 2008.

[5] Liu Bing. *Web Data Mining*. Springer, 2007.

[6] Max Bramer. *Principels of Data Mining*. Springer, 2007. 21

[7] Brian Clifton. *Advanced Web Metrics with Google Analytics*. Sybex, 2 edition, 2010.

[8] Yukihiro Matsumoto David Flanagan. *The Ruby Programming Language*. O'Reilly, 2008.

[9] Philip S. Yu Ding Xiaowen, Liu Bing. A holistic lexicon-based approach to opinion mining. *WSDM'08*, 2008. vii, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37, 39, 58, 59, 87

[10] Brad Ediger. *Advanced Rails*. O G Popa, 2008.

[11] David A. Freedman. *Statistical Models: Theory and Practice*. Cambridge University Press, 2 edition, 2009.

[12] Tom Funk. *Web 2.0 and Beyond: Understanding the New Online Business Models, Trends, and Technologies*. Praeger, 2008.

[13] Abraham Kandel George Meghabghab. *Search Engines, Link Analysis and User's Web Behaviour*. Springer, 2008. 14

[14] Lynne Schrum Gwen Solomon. *Web 2.0: New Tools, New Schools*. International Society for Technology in Education; First Edition edition, 2007.

[15] Liu Bing Hu Minqing. Mining and summarizing customer reviews. *KDD'04*, 2004. vii, 27, 30, 31, 32, 33, 34, 35, 38, 59

[16] Eibe Frank Ian H. Witten. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann; 2 edition, 2005.

[17] Paul Kimmel. *UML Demystified*. McGraw-Hill Osborne Media; 1 edition, 2005.

[18] David M. Pennock Kushal Dave, Steve Lawrence. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. *WWW'03*, 2003. 34, 36

[19] Arthur M. Langer. *Analysis and Design of Information Systems*. Springer; 3rd edition, 2007.

[20] David Heinemeier Hansson Leonard Richardson, Sam Ruby. *Restful Web Services*. O'Reilly Media; 1 edition, 2007.

[21] Chunping Li Lili Zhao. Ontology based opinion mining for movie reviews. *KSEM 2009*, 2009.

[22] Simon Corston-Oliver Eric Ringger Michael Gamon, Anthony Aue. Pulse: Mining customer opinions from free text. *IDA*, 2005. 34, 36

[23] Michael Milton. *Head First Data Analysis: A Learner's Guide to Big Numbers, Statistics, and Good Decisions*. O'Reilly Media, 1 edition, 2009.

[24] Russ Olsen. *Design Patterns in Ruby*. Addison-Wesley, 2008.

[25] Vipin Kumar Pang-Ning Tan, Michael Steinbach. *Introduction to Data Mining*. Addison Wesley, 2005.

[26] Michael Papazoglou. *Web Services: Principles and Technology*. Prentice Hall; 1 edition, 2007.

[27] Tom Pender. *UML Bible*. Wiley, 2003.

[28] A P Rajshekhar. *Building Dynamic Web 2.0 Websites with Ruby on Rails*. Packt Publishing, 2008.

[29] Gary Miner Robert Nisbet, John Elder IV. *Handbook of Statistical Analysis and Data Mining Applications*. Academic Press, 2009.

[30] Toby Segaran. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly Media; 1 edition, 2007.

[31] Peter D. Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. *Proceedings of the 40th Annual Meeting of ACL*, 2002.

[32] Torsten Suel Vladislav Shkapenyuk. Design and implementation of a high-performance distributed web crawler, 2001.

[33] Pawan Vora. *Web Application Design Patterns (Interactive Technologies)*. Morgan Kaufmann, 2009.

[34] Mark Watson. *Scripting Intelligence*. Apress, 2009.

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other German or foreign examination board. The thesis work was conducted from 11.2009 to 02.07.2010 under the supervision of Dipl.-Medieninf. Maximilian Walther at Informatics Faculty, TU Dresden.

Dresden, 02.07.2010