SAPIENZA
UNIVERSITÀ DI ROMA

Ontology extraction and population from user-generated text on online marketplaces

Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Ingegneria Informatica

Candidate

Sara Di Bartolomeo
ID number 1494990

Thesis Advisor

Prof. Riccardo Rosati

Academic Year 2016/2017

Thesis defended on 16 January 2018
in front of a Board of Examiners composed by:

Prof. Riccardo Rosati (chairman)
Prof. Silvia Bonomi
Prof. Giuseppe De Giacomo
Prof. Luca Iocchi
Prof. Riccardo Lazzeretti
Prof. Andrea Marrella
Prof. Roberto Navigli

Ontology extraction and population from user-generated text on online marketplaces
Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: January 15, 2018

Author's email: dibartolomeo.sara@gmail.com

Dedicato a
Nonno Lamberto

Abstract

Online marketplaces have become representative of a significant part of the global market, and they have some peculiar characteristics that make them uniquely interesting. They aggregate an enormous amount of data about the products that they are selling, in the form of product descriptions and reviews. This form of commerce is indeed a trust-based system: users form their opinions about an item according to other users' opinions, expressed in the form of reviews. Reviews, thus, contain a wealth of relevant data, but this information is expressed in natural language, meaning that it's understandable for a human being but difficult to translate into meaningful data for a computer program. This thesis addresses the problems involved with the programmatic analysis of the language contained in reviews with the purpose of extracting information to build and populate an expressive knowledge model - an ontology.

Acknowledgments

I would like to thank the whole Dipartimento di Ingegneria Informatica, Automatica e Gestionale at Sapienza. From what I experienced, the department aggregates a great deal of brilliant and passionate students and professors, from which I had the opportunity to learn the proudness in dedication and discipline, the love for the subjects, the long lasting satisfaction in pursuing achievements. The years I have spent in this building forged me into an Engineer, with a sharp attention for details and a deep rooted thirst for knowledge.

In particular, I would like to thank professor Riccardo Rosati, who has been my thesis advisor, who trusted my ideas and wisely guided me through the process of shaping them into a Master's thesis.

I would also like to thank each one of my colleagues, as many of them have been role models and figures of guidance to me. I am grateful for the opportunity that I had to get to know Matteo, Alessandro, Lorenzo, Gabriele, Alessio, and many more brilliant young computer scientists. Along with them, I want to mention a couple of colleagues, Massimo and Bruno, who made me really understand that creativity and coding weren't mutually exclusive concepts, and whose advice has been fundamental in my growth.

A special place goes to my grandfather and my brother, who have been, to me, the greatest source of inspiration.

Lastly, I want to thank Giorgio, who has been, at the same time, a loving partner, a great teammate, and a best friend, and who has been at my side through all the challenges I had to face.

Contents

Chapter 1

Introduction

In recent years, the shopping habits of the general public have been changing considerably: more and more people are choosing to use online marketplaces for their purchases. This form of shopping platform has introduced several advantages: users have access to a much bigger selection of products, and barriers that would otherwise have prevented access to the market to smaller sellers have been broken down.

Nevertheless, these advantages are balanced by other downsides. As an example, judging the quality and features of an item is a more complex task, as an user will only have access to photos and textual descriptions of the item.

In this context, a trust-based system between multiple users has gained more and more relevance: reviews and reputation. The judgement of a buyer will be substantially affected by other users' reviews, as well as a level of trustworthyness assigned to sellers by other users.

We have to take into account that reviews represent a relevant strategic resource to sellers. Making an user able to gain knowledge about the quality, purpose and details of an item is a problem that concerns not only the users, but also the sellers. Indeed, a number of good reviews may be the deciding factor for a successful sale.

Since reviews are so relevant, they have been the object of a series of studies. They are, in fact, made of unstructured informations about a product: the data they contain, if succesfully extracted, may help in identifying precisely the features of a product. Though, the overwhelming amount of informations contained in reviews makes them difficult to consider in every detail for a human reader. If we consider the scale on which the informations we obtain becomes significative, though, we notice that an automatized system for extracting information from text is needed.

The idea that, after the world wide web, we would one day have a Semantic Web in which the information is much more structured has existed for quite some time, now. Tim Berners-Lee explained the Semantic Web with these words (Berners-Lee, Hendler, and Lassila 2001):

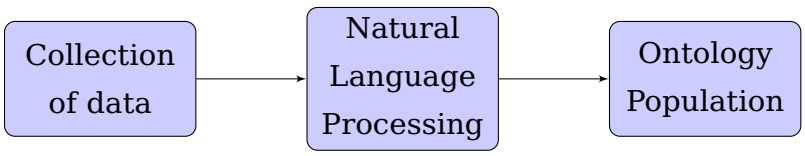
> *Most of the Web's content today is designed for humans to read, not for computer programs to manipulate meaningfully. Computers can adeptly parse Web pages for layout and routine processing—here a header, there a link to another page—but in general, computers have no reliable way to process the semantics.*
>
> ...
>
> *The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users.*
>
> ...
>
> *The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in*

cooperation.

This idea, though, requires the information to have an underlying structure. As mentioned in the text above, internet contains an incredible amount of data, but it's not machine-readable. The need for a system for automatic extraction of ontologies from text is poignant for this purpose, therefore a number of researchers have concentrated their efforts on the field of **ontology learning**, that is information extraction targeted at building an ontology.

The project presented in this thesis is aimed at extracting and structuring the information contained in user generated content regarding products on online marketplaces. The idea is to bridge different disciplines, using ideas and algorithms from machine learning, natural language processing and knowledge representation, to create an ontology.

The project is composed of three main steps, in which the output of a step is the input of the following one.



In the following chapters, we'll describe in depth these three steps:

- The first chapter, **Collecting Data**, describes the initial step of gathering the data used for the project, including the challenges and the methodologies. The reviews have been collected from Amazon via scraping, so the process required some foresight to be efficient and functional.
- The second chapter, **Processing the Information**, describes how we extracted relevant information from the wealth of text collected previously. Natural Language Processing is the field that studies the interpretation, from an algorithmic point of view, of phrases in commonly written/spoken languages. Thanks to advancements in

this field, we are able to identify, extract and classify details about the products from the reviews.

- The last chapter, **Ontology Extraction and Population**, describes how the analyzed data has been used to programmatically populate an OWL ontology.

Each chapter refers to a slightly different field of study, and starts with a general description of the field and of the techniques, concepts and algorithms used, and then describes how the same concepts have been applied during the development of the project.

1.1 Related works

The objective of automatic or semiautomatic population of ontologies has been addressed with several techniques. There are, though, some details to take into account: the domain and the structure of the input. The input can in fact come in multiple forms: unstructured natural language, semi-structured documents (e.g. XML), an existing knowledge base, a dictionary, or a database. This thesis concentrates on unstructured natural language, and the following are possible approaches at dealing with this kind of input (AL-Aswadi and Yong 2017, Biemann 2005):

- **Word co-occurrence:** (Roark and Charniak 2000, Yarowsky 1995) Simply measuring how many times two words appear in the same phrase can give us an idea of how much two words are related. In Roark et al., a number of ’seed’ words, representative of several categories, is chosen at the beginning, then the words that co-occur with the seed the highest amount of times are added to the respective categories. Also related to **association rules**.
- **Pattern-based extraction:** (Thelen and Riloff 2002) Similar to regular expressions, we look for patterns in the text. Starting from ’seed’ words - we look for predefined patterns such as "<some verb> <some object>" (e.g. "is made of <X>")
- **Clustering-based techniques:** (Schickel-Zuber and Faltings 2007,

Buitelaar and Pustejovsky 1998) a distance measure of terms has to be defined in order to find out which terms are most similar and cluster them in groups. One particular word - the hypernym of the cluster (Hearst 1992) - can be extracted from each cluster to be representative of that cluster. If we extend this to hierarchical clustering, we can obtain a hierarchy of words. Methods using Latent Semantic Indexing or Latent Dirichlet Allocation fall under this category, building clusters by trying to discover topics.

- **Part-of-speech tagging:** (Maedche and Staab 2001, Maedche and Staab 2000) Part-of-speech tagging (identifying subject, verb, object in a phrase) is used to discover entities and possible relations in the phrases. **Named entity recognition** may be used to identify the types of particular named entities.
- **Ontology pruning:** (Kietz, Maedche, and Volz 2000, Volz et al. 2003) A generic ontology (e.g. WordNet) is used. Input text is classified according to the generic ontology to acquire domain concepts. The generic ontology is pruned to remove non-domain specific entries. Decision about which entry is domain specific and which one is not is often based on the assumption that a domain-specific concept is mentioned more often in a domain-specific corpus than in generic text.

It is also interesting to explore the range of domain on which said techniques have been applied. For example, in 2007, Jeschke et al. Jeschke et al. 2007 used Natural Language Processing techniques in their project, mArachna, to extract relations from mathematical text.

It is relevant to mention Navigli et al.'s work on Ontology Learning (Navigli, Velardi, and Gangemi 2003), and in particular on OntoLearn (Reloaded) (Velardi, Faralli, and Navigli 2013), in which a measure of relevance is used to extract the most relevant domain-specific terms in a collection of domain-specific documents, uses part-of-speech tagging to extract term definitions, and uses graph pruning to refine the produced graph.

1.1.1 Extracting an ontology via the structure of a website

The idea of extracting an ontology from the structure of a series of HTML documents has been also the object of research. Websites already have a structure that can be used as a base for building an ontology.

It has been applied in (Pappachan et al. 2015). The domain of interest was composed by the apps on Google Play. Since Google Play has a similar policy of making content difficult to retrieve programmatically, the cited paper required a similar work of scraping and parsing pages. The work by Pappachan et al., though, does not consider the extraction of information from reviews present on the store.

The same concept is also applied in Fernández Villamor et al. 2011, in which they map HTML elements to RDF resources with the objective of building a semantic scraper, a concept that is also explored in (Liu, Pu, and Han 2000) and (Muslea, Minton, and Knoblock 2001).

1.1.2 More related works

The following works are not necessarily related to the actual extraction of the ontology from natural language, but have been relevant for our project as well.

Our first inspiration for using the content of Amazon reviews came from (Haider 2012). In 2012, he published a case study in which he used reviews from Amazon to populate an ontology. His approach used a pre-made ontology about mobile phones, and applied sentiment analysis to the Amazon reviews of three specific items (three mobile phones) to understand the opinion of the public about the qualities of the items that were being analyzed.

The ideas about classifying questions and answers in open-ended ques-

tions and closed-ended questions came from Julian McAuley and Alex Yang (McAuley and Yang 2016).

1.2 Objective of the project

The purpose of this thesis is to build on the aforementioned works to:

- Extract an Ontology from user-generated content on Amazon;
- Focus on reviews and question/answer sections;
- Build a taxonomy of product categories from the structure of the HTML documents;
- Extract the properties of the products to be included in the ontology as properties of the entities in the ontology;

Chapter 2

Collecting Data

Modern online marketplaces often offer customers the ability to give feedback to the vendors, often in the form of reviews. Recently, some platforms also started to insert question/answer forms, in order to let the users be able to formulate questions about a product, and have other users or the seller respond to their inquiries about the product.

Feedback is given by the users in the form of:

- Numerical score (i.e. 1 to 5 stars)
- Reviews
- Questions and Answers

Although the first value is very easy to be semantically represented, the latter ones are not so straightforward. They are, in fact, expressed in natural language, and their analysis will be the focus of the following chapters.

The first step of the project is, thus, the collection of data.

2.1 Amazon

The main focus of the project is extracting data from the biggest marketplace as of today, namely, Amazon. As stated in an article by Business Insider (Intelligence 2017), Amazon is accountable for 43% of online sales in the US in 2016.

As of November 2017, Amazon has 573 million products currently on sale [1], and is undoubtedly a giant in its field. In the last couple of years, they also attempted the sale of a wider number of products, like fresh food and pantry items, expanding the range and types of products that can be found on their platform.

More than the number of products that the marketplace offers, Amazon gives to the buyers the ability to review items, and post questions and answers to previously published questions. Precise data about the numbers of reviews and questions present on Amazon remains undisclosed, but a popular product can easily reach thousands of both. Having access to user comments is incredibly relevant to online buyers, because the trust they can put in other users is the best way they have to judge the quality and properties of a product.

Amazon reviews also give users the possibility to give a numerical rating expressed in stars, from 1 to 5. This element had made Amazon reviews the object of a considerable amount of studies on sentiment analysis: the textual content of a review is already associated with an opinion - although a simple one - and this makes it useful to build a base datatset to be used on sentiment analysis on more text. The same happens to movie reviews on IMDB.

Nevertheless, having this wealth of data in reviews on Amazon doesn't make it really accessible. Not only the reviews are written in natural

[1] source: `https://www.scrapehero.com`

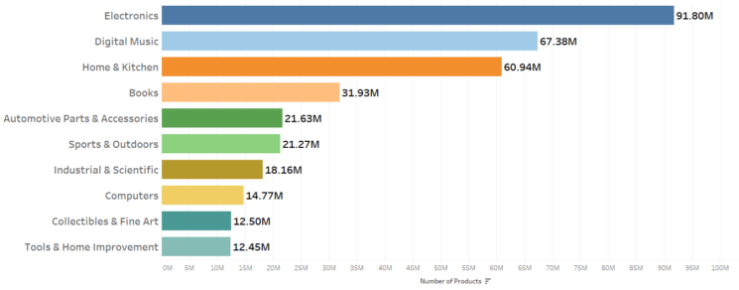


Figure 2.1. a comparison between different product categories on Amazon

language - thus not really comprised of machine-readable data - they are also organized in a difficult way to navigate. Amazon displays 10 reviews per page, therefore a person that wants to read all the reviews of a product might have to click through hundreds of pages. Not to mention that reading a thousand reviews is not feasible nor useful for a person, as the data they contain may be repetitive and may make the person miss some of the details. The users who wrote reviews, though, may have written relevant details about a product that are not present in the seller's description - even in the last review on the last page, there may be a relevant detail.

2.2 The challenges

2.2.1 Amazon API

Amazon offers an API that makes available some of the information I wanted to examine. Unfortunately, neither the request limits imposed to API users nor the information made available were useful for the purpose of this project.
Indeed, on 8 November 2010 Amazon removed the possibility to retrieve reviews from their API, returning now an URL to an IFrame containing just the first three reviews. The reason behind the removal of this feature were never explained by Amazon, and it has been discussed that the huge amount of data represented by written feedback is now considered a

valuable resource that Amazon wants to protect.
Offering the first three reviews in an IFrame is intended for sellers to showcase on their website some Amazon reviews about their products, but I needed much more than just three reviews per product. The nature of the IFrame DOM element makes it difficult to deal with for accessing its contents, and makes the process of retrieving clean review content similar to the process of scraping a web page.

2.3 Scraping

Scraping is a technique for extracting data from a website via a software program. Scraper programs simulate human behaviour in accessing the content in webpages, with the purpose of collecting and transforming unstructured data, often found inside the HTML code of webpages, in metadata useful for being memorized, manipulated and analyzed locally.

A scraper program will access the content on a page directly through the HTTP protocol. A page is first downloaded through an HTTP GET request, in the same way in which a browser would request a page for visualizing it. The web server at the requested address will respond with the HTML code used by the browser to visually render the page. Instead of rendering it, a scraper will parse the code of the page to find patterns and references for purposeful data in the HTML. An example of this is contact scraping: a scraper is useful in finding all the email addresses in a page.

Commonly, scraping is done on a huge number of pages at the same time, and is therefore deeply linked with web crawling, that is the technique of navigating through a series of links to obtain a huge number of pages.

In addition to a number of challenges directly involved in the process, some websites will try to prevent this practice through various techniques, like IP blacklisting, CAPTCHAs, A/B testing and obfuscation of the content.

2.3.1 Scraping a single page

Scraping a single web page is performed in the following steps:

1. An HTTP GET request is used to download the HTML code of the page that is being analyzed. The response, if the request was successful, is HTML plain text.
2. The response is then parsed to identify HTML components.
3. Thanks to the tree-like structure of HTML elements in a page, each component is stored in a tree shaped data structure, called parse-tree.
4. The parsetree is searched according to several criteria (an example may be regular expression matching) to extract relevant data.

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>The requested URL was not found on this server.</p>
</body>
</html>
```

For the purpose of parsing, storing and accessing the result of the request, I used a python library designed to ease this task, BeautifulSoup (*Beautiful Soup: We called him Tortoise because he taught us.* 2017). Processing the page in this way allows me to navigate HTML easily and query the contents.

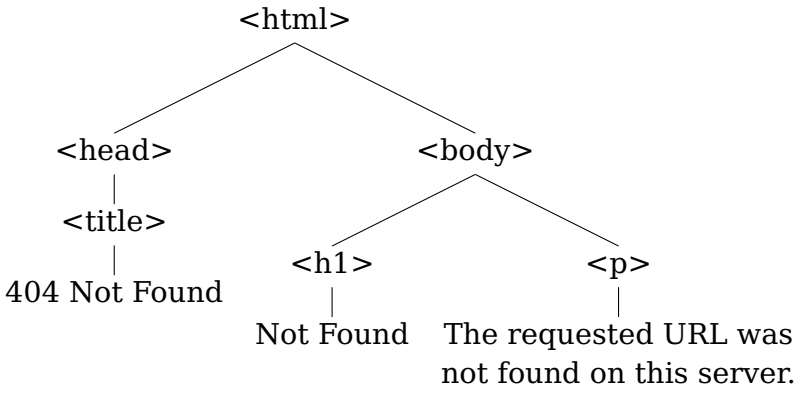


Figure 2.2. The parsetree generated from 2.3.1

In the code example above, at 2.3.1, retrieving the title of the received page would be written like this:

```
1 BeautifulSoup bs = BeautifulSoup(res, 'html.parser')
2 title = bs.find('title').text
```

This code would perform a search on the document's parsetree, then extract what is contained in the tag `<title>`.

Real world html pages are much more complex than this, usually containing thousands of tags, and are often dynamically generated, thus they require fine scraping criteria. The procedure used needs to be able to adapt to slight modifications in the page structure and contents.

As you can see in 2.3, the structure of a standard product page on Amazon is convoluted, and identifying the parts that contain relevant information via code may not be straightforward.

2.3.2 Dynamic pages

The modern web is comprised of a huge amount of content that is purposefully tailored for the user's needs and desires. In particular, when an HTTP request is made to a server, the server may respond with content specifically tailored for the particular user who made the request.

Think of facebook: each user sees the same website at the same address in a unique way, depending on the user's account, the cookies he has

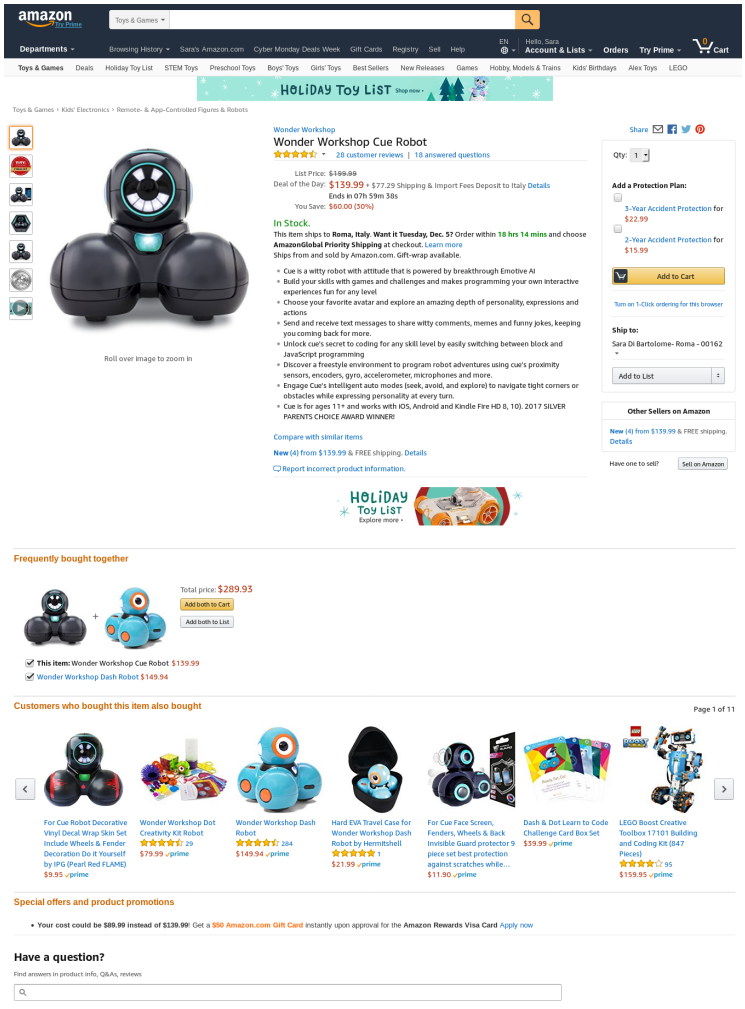


Figure 2.3. a product page on amazon

stored in its website, the times and types of content that the user's friends posted.

This may happen in the case of:

- visual elements: a small screen with scarce resolution won't have the ability to display pictures as large and detailed as those needed on a Retina display. For this reason, websites often choose to serve smaller resolution pictures to devices with less displaying capabilities. The visual differences may also affect the way the website's UI is intended, to reflect the way of interacting with the requesting devices: mobile browsers used via touch input often require a different interface design than the one used for their desktop version.
- content: the content of a page can also be the object of tailoring based on the user. Some websites may choose to display certain contents based on the nationality of the user, to directly send to the users content in their native languages. In other cases, the websites send to the users content according to the account that the user has on that particular website, based on the cookies their browser stores.

Therefore, a web server does not store a single web page for each possible user or product, but rather the page is built according to the user's needs every time. This can be implemented using different tools, like PHP and Javascript.

In the case of PHP, we talk about a server-side scripting languages, meaning that PHP is executed and the page is completely built before it is sent to the client. This case does not create a problem for a scraper program, as the scraper will only receive the full, already built HTML code.

In the case of javascript, instead, modifications to the structure and functionality of the pages are executed client-side. A browser would receive HTML and javascript code, then execute javascript along the rendering process. A python scraper doesn't have the ability to execute javascript without simulating the execution of a browser, therefore can't

access the content after modifications included via javascript. As these modifications can include additional content in the form of text or images added to the page, this can become a relevant problem.

In many cases, though, websites tend to have a fallback version in case the user's browser is not enable or doesn't have the capability of running javascript. The fallback version is often implemented by having an HTML layout that is still functional even though javascript can't be run, or by having HTML code that is displayed only when scripts can't be executed thanks to the HTML tag `noscript`.

2.4 Scraping on a larger scale

Although scraping a small set of pages may be easy, scraping a large quantity of content introduces several challenges that I had to overcome. A part of these difficulties involved overcoming bottlenecks due to the large use of HTTP requests and the management of the results, whilst another part involved the problem of not making Amazon upset.

Some of the problems associated with large-scale scraping are related to I/O bottlenecks and memory management. In the following sections, we are going to address each one of the problems and discuss how we managed to overcome the obstacles.

2.4.1 Multithreading

If we suppose that the average time for an HTTP request to return with a response is 2-3 seconds (counting the ones that work almost immediately and the ones that come back after 10 seconds), the time for processing a large number of pages adds up very fast. HTTP requests are blocking, meaning that a thread who performs an HTTP request will not continue its execution until a response is received, and if they are executed on the main thread, the whole program is blocked for a few seconds. The problem with HTTP requests is that they are highly unreliable, and the probability of one request coming back in a long time or not coming back

at all is very high. For this reason, requests are never done on the main thread, and, since a scraper needs a huge number of requests, we also need a huge number of threads.

2.4.2 Tampering requests

More than being careful about managing I/O nework flow, we also have to be careful about not being identified as a script by Amazon. Indeed, Amazon applies some measures to prevent scraping, and will try to identify the nature of the requests to detect if they can be considered legitimate requests (meaning that they have been made by a human) or not.

If the source of requests is suspected to be a bot, depending on the website's adopted policy, some measures can be taken against the practice of scraping. In the case of Amazon, the source's IP will be blacklisted, making us unable to use the same IP to keep making requests. Since my project required a huge amount of pages to be collected, using a script was necessary. The following sections will deal with concealing as much as possible the nature of the scraper program against being detected as a bot by Amazon. The tecniques used in this project were a result of circumventing common bot detecting methods.

Proxies

Many websites will try to keep track of requests coming from specific IPs. So, if a certain IP performs a huge number of requests in a short timeframe, it will be obvious that behind that IP there isn't a human, but a script. The navigation pattern is also something to keep an eye on, as it is another indicator that may identify a bot: if a server detects too many similar requests from the same user, or requests of a weird nature (i.e. too many searches), then the user may be considered suspect.

As the reader will already know, a proxy server is an intermediary between a client and a destination server. If a client requests a page through a

proxy, the server will receive requests that look like they are coming from the address of the proxy, and not from the address of the real source, concealing what our real IP is. So, if our IP has been blacklisted, we can use a proxy server to keep making requests from a non-blacklisted IP.

At this point, though, we have to consider two details:

- If we always use the same proxy address to make requests to the same server, if the server suspects us, it will blacklist the address.
- We want to be able to make many requests fast, and not make our scraper sleep for some seconds between requests to fake being a human.

To solve these problems, we are going to use hundreds of proxies. The scraper randomly chooses and address from the list each time it makes a request, making them look like each one is coming from a different location.

Using a proxy is made simple enough by python's `requests` module, and it only requires the IP address of the proxy server to be included in the request.

```
proxy_list = ['191.103.252.169', '87.98.157.128',
              # ... many more addresses ...
              '52.224.181.154', '5.196.189.50']

res = requests.get(url,
    proxies = { 'http' : random.sample( proxy_list, 1 )},
)
```

Another detail to take into account is that proxy servers are not totally reliable in terms of availability: they are just machines in different locations, and we are given no guarantee that the machines are always available on the network or will be available for the whole execution of the program, nor are we guaranteed that the same addresses will be valid in the subsequent days. For this reason, we need a set of proxies that is always up to date.

Since the proxy addresses were collected from a webpage serving the addresses [should I insert the webpages?], the solution was to use the same scraping method on the proxy serving page and request programmatically an updated page containing a fresh list of proxies after a certain amount of time.

2.4.3 Crafting user agents

The IP address is not the only detail that can identify a user: the headers of the requests contain data that can uniquely identify the source of the requests. One example is the user agent: a string, contained in the header, that informs the server of the browser type and version and operative system from which the requests are being made. **Browser fingerprinting** is the idea of being able to uniquely identify a user based on these characteristics: these details may seem scarce, but the intersection of the set of users with the same characteristics can identify a unique users.

As an example, as of December 2017, the details on my machine identify as a unique user because:

- 1.78% of the users are running Firefox 57
- 14.78% of the users are running Linux
- 62.99% of the users have 'en' set as their primary language
- 20.19% of the users have UTC+1 set as their timezone

[source: `https://amiunique.org`]

These details are crafted by the browser and sent to the server, so obviously if my request is not sent through a browser, but through a script, the details will be different. The most relevant of the details is the user agent, that informs the server of the operative system and browser version. The default header sent to the server when making a request through `python requests` is:

```
1 {
```

```
        'Connection': 'keep-alive',
        'Accept-Encoding': 'gzip, deflate',
        'Accept': '*/*',
        'User-Agent': 'python-requests/2.18.4'
}
```

Therefore, if the user agent is not changed, we are explicitly communicating to the server that we are making our requests through a script. We can, thus, programmatically change the header details and user agent. User agents are simple scripts, a valid user agent string is:

```
# chrome on linux 64 bit
'User-Agent' : 'Mozilla/5.0 (X11; U; Linux x86_64; en-US)
   AppleWebKit/540.0 (KHTML,like Gecko) Chrome/9.1.0.0 Safari
   /540.0'
```

```
# facebook app on ipad
'User-Agent' : 'Mozilla/5.0 (iPad; CPU OS 6_0_1 like Mac OS X
   ) AppleWebKit/536.26 (KHTML, like Gecko) Mobile/10A523 [
   FBAN/FBIOS;FBAV/6.0.1;FBBV/180945;FBDV/iPad2,1;FBMD/iPad;
   FBSN/iPhone OS;FBSV/6.0.1;FBSS/1; FBCR/;FBID/tablet;FBLC/
   en_US;FBOP/1]'
```

By mixing valid strings to create new OS/browser combinations, we can then not only fake our position (with proxies), but we are now even faking what type of machine we are generating the requests on. By regenerating a new randomized user agent string every time, and using a random proxy every time, the scraper program makes the server believe that the requests are coming each time from a completely different source and user.

2.5 Flow diagram of the scraper

The flow diagram at 2.4 descibes an overview of in what sequence the scraper program applies the techinques described in the previous chap-

ters.

At the beginning, a list of ASINs (unique product IDs on Amazon) is collected through an API request (this is the only step in which the API is used), together with a list of proxies. For each ASIN, a thread is started. The purpose of each thread at this point is to retrieve the entire product page, as it not only contains purposeful data about the product - i.e. in the seller's description - but it also contains the reviews and question/answers links that need to be used in the following steps. Each thread uniquely crafts a new, random user agent and uses a random proxy. When a thread receives a response, stores the reviews and question/answers links in queues, extracts the relevant content in the collected page and saves it to json.

Then, a new set of threads is started, this time working on the urls in the queue. Both reviews and question/answers are divided in multiple pages, each page containing 10 reviews or 10 questions, therefore to collect all the reviews of a product that has, for example, 100 reviews and 200 questions, we need to make in total 30 requests, plus one for the product page. After collecting the pages, the threads extract the relevant information via HTML parsing, structure them in python dictionaries according to what type of data they collected and save them in json files.

2.6 amazon-scraper

I implemented all the previously discussed techniques in a CLI application that can be found at `https://github.com/picorana/amazon-scraper`, in order to make it useful to other people or to be used as a reference. `amazon-scraper` is indeed built to be used on Amazon, but the same techniques are valid for many other domains.

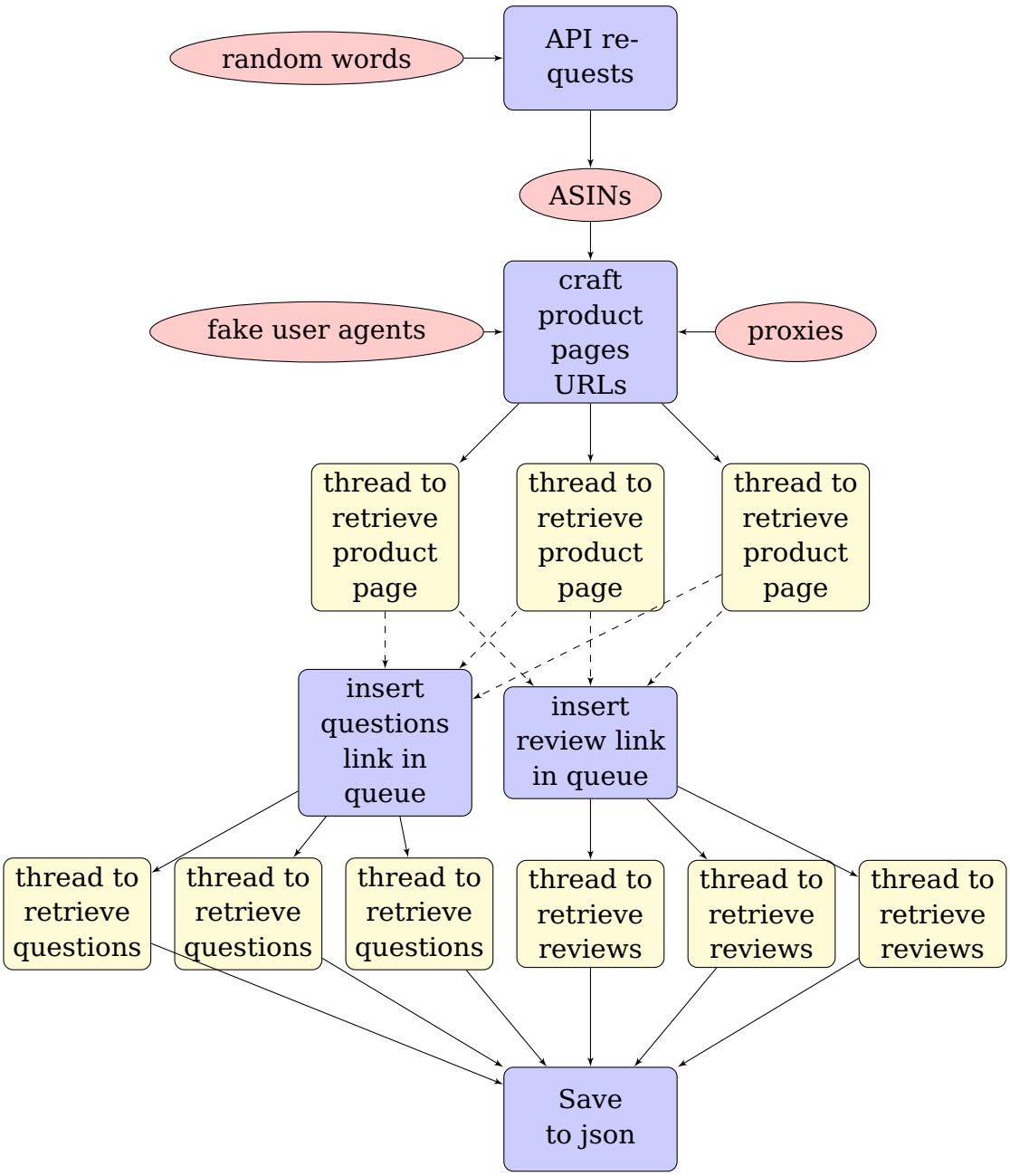


Figure 2.4. Flow diagram of the scraper

```
python app.py
usage: app.py [-h] [--file FILE] [--save-pages] [--verbose] [--no-reviews]
              [--no-questions] [--destination DESTINATION] [--ignore-dups]
              [--quiet]
              [asin [asin ...]]

amazon-scraper downloads questions and reviews from amazon products

positional arguments:
  asin                  Amazon asin(s) to be scraped

optional arguments:
  -h, --help            show this help message and exit
  --file FILE, -f FILE  Specify path to list of asins
  --save-pages, -p      Saves the main pages scraped
  --verbose, -v         Logging verbosity level
  --no-reviews          Do not scrape reviews
  --no-questions        Do not scrape questions
  --destination DESTINATION, -d DESTINATION
                        Set a destination folder
  --ignore-dups         Do not consider previous operations
  --quiet, -q           Be quiet while scraping
```

Figure 2.5. `amazon-scraper` usage

2.7 Shape of the collected data

After the data has been collected, it is then stored in jSON and HTML files to be later used by the next section of the program, which will proceed with the natural language analysis phase.

Reviews are stored with metadata extracted from the page. They contain relevant information both in the title and in the actual text of the review, and they are also associated with a rating, that is relevant to us in later steps.

```
{
        "rating": "5.0 out of 5 stars",
        "author": "Tony",
        "text": "I agree with the other positive reviews of this
           book. First, it provides a lot of useful information.
           All of my questions were answered in this book, and
           then some. Second, it's just a high quality, good
           looking book. If you read this book and you are not
           sure you want to be a beekeeper, you should probably
           decide not to be a beekeeper.",
        "title": "Love this book.",
        "date": "on April 16, 2014",
        "author_id": "ref=cm_cr_arp_d_pdp?ie=UTF8"
}
{
        "rating": "5.0 out of 5 stars",
        "author": "Amazon Customer",
        "text": "killer beekeeper's handbook.  I tote it around
           all over.  Has helped me gain much knowledge for my new
            hive!  Recommended!!",
        "title": "Recommended!!",
        "date": "on October 6, 2016",
        "author_id": "ref=cm_cr_arp_d_pdp?ie=UTF8"
}
```

Questions and answers have much less relevant metadata. so each dictionary contains just a question-answer couple.

```
{
        "question": "How long is the cable?",
        "answer": "id say about 18 inches"
}
{
        "question": "Is this the cable and plug all together or
           seperate? Picture shows cable and box, then one
           together which doesn't come apart?",
        "answer": "It is separate and using this charger it work
           great no complaints happy with product."
}
```

This last document is made by data extracted from the product page and/or collected via what the API offers. The sources can be tables or text fields included in the product pages, so this helps identify what kind of information about a product is being expressed without having to analyze the language. Although some parts of this document are expressed in natural language - like the editiorial review, that is just a short description of the product written by the seller - the majority of it is already classified in data types, therefore easier to deal with.

This document may change according to what kind of product it is about. The following extract describes a book, therefore having fields such as "authors" and the number of pages makes sense, while it may not be as meaningful if we are describing, for example, a phone.

```
{
    "release_date": "1994-04-27 00:00:00",
    "product_type_name": "ABIS_BOOK",
    "isbn": "006097625X",
    "features": [
        "Great product!"
    ],
    "parent": null,
    "ean": "9780060976255",
    "color": null,
    "brand": "William Morrow Paperbacks",
```

```
    "sales_rank": "6092",
    "is_adult": null,
    "edition": "Reprint",
    "studio": "William Morrow Paperbacks",
    "authors": [
        "Scott McCloud"
    ],
    "genre": null,
    "publication_date": "1994-04-27 00:00:00",
    "editorial_review": "<p><strong>The bestselling international
       classic on storytelling and visual communication</strong></
       p><p><strong>\"You must read this book.\" </strong>\u2014<
       strong>\u00a0Neil Gaiman</strong></p><p>Praised throughout
       the cartoon industry by such luminaries as Art Spiegelman,
       Matt Groening, and Will Eisner,\u00a0Scott McCloud's <em>
       Understanding Comics</em>\u00a0is\u00a0a seminal
       examination of comics art: its rich\u00a0history,
       surprising\u00a0technical components, and major cultural
       significance.\u00a0Explore the secret world between the
       panels, through the lines, and within the hidden symbols of
        a powerful but misunderstood art form.</p>",
    "reviewTexts": [],
    "pages": "224",
    "manufactures": "William Morrow Paperbacks",
    "running_time": null,
    "sku": null,
    "asin": "006097625X",
    "price_and_currency": 14.48,
    "author": "Scott McCloud",
    "eisbn": null,
    "product_group": "Book",
    "region": "US",
    "publisher": "William Morrow Paperbacks",
    "upc": "884387471390",
    "label": "William Morrow Paperbacks",
    "languages": [
        "english"
```

```
    ],
    "mpn": "09787503",
    "part_number": "09787503",
    "actors": [],
    "parent_asin": "None",
    "title": "Understanding Comics: The Invisible Art",
    "model": null,
    "creators": [],
    "editorial_reviews": [
        "<p><strong>The bestselling international classic on
           storytelling and visual communication</strong></p><p><
           strong>\"You must read this book.\" </strong>\u2014<
           strong>\u00a0Neil Gaiman</strong></p><p>Praised
           throughout the cartoon industry by such luminaries as
           Art Spiegelman, Matt Groening, and Will Eisner,\u00a0
           Scott McCloud's <em>Understanding Comics</em>\u00a0is\u
           00a0a seminal examination of comics art: its rich\u00a0
           history, surprising\u00a0technical components, and
           major cultural significance.\u00a0Explore the secret
           world between the panels, through the lines, and within
            the hidden symbols of a powerful but misunderstood art
            form.</p>",
        "A comic book about comic books. McCloud, in an incredibly
            accessible style, explains the details of how comics
           work: how they're composed, read and understood. More
           than just a book about comics, this gets to the heart
           of how we deal with visual languages in general. \"The
           potential of comics is limitless and exciting!\" writes
            McCloud. This should be required reading for every
           school teacher. Pulitzer Prize-winner Art Spiegelman
           says, \"The most intelligent comics I've seen in a long
            time.\""
    ],
    "offer_url": "http://www.amazon.com/dp/006097625X/?tag=
       picorana-20",
    "detail_page_url": "https://www.amazon.com/Understanding-
       Comics-Invisible-Scott-McCloud/dp/006097625X?SubscriptionId
```

```
        =AKIAIRCSTDTCOYE3OA6Q&tag=picorana-20&linkCode=xm2&camp=202
        5&creative=165953&creativeASIN=006097625X"
}
```

Chapter 3

Processing the information

Once enough data has been gathered, we can proceed with the language analysis part. The intent of this analysis is to extract meaningful information from text. User generated text on product pages can in fact contain relevant details about a product, that may or may not be specified in the seller's description of the same product.

The reasons for a missing detail in the seller-provided description may be several:

- The seller did not consider the detail relevant, but discussion about the detail present in the product's comments proves that clients consider the detail important
- The seller forgot to include the detail
- The seller did not realize that his product had a particular characteristic
- The seller lied about a characteristic of his product

The characteristics listed above define a knowledge about the product that may be considered useful for a client to judge a product, but may not be easily understood. Moreover, these characteristics are relevant in our intent of building an ontology of products.

3.1 Natural language processing

Natural Language Processing is the process of programmatically treating information written in natural language, that is in this case American English.

Natural language contains data, but that data is not easily usable by a computer program. A phrase like

> This tablet costs 279$

clearly contains a relevant information: the price of an item we may be interested in, clearly underlined by the fact that the string contains a number. Nevertheless, our human brain is wired to understand that the number represents a price, but the same is not a straightforward process for a computer program. The process is made more difficult by the underlying ambiguity of the human language.

The process of analyzing a string is performed in four different stages:

- Splitting the string in sentences and words, namely *tokens*
- Assigning each chunk (or word) a part-of-speech tag
- Arranging the tokens in a syntactical structure (a *parsetree*)
- Assigning to the structure a semantical meaning

To perform these tasks (for example, part-of-speech tagging), we rely on machine learning techinques to learn from a large *corpus* [1] how certain words in certain positions are intended.

A machine learning approach is much more versatile because it is able to manage unfamiliar structures - that is, structures of the phrase that were not present in the original dataset, but can be derived from similar entries.

[1] A corpus is a large collection of documents with human-made annotations

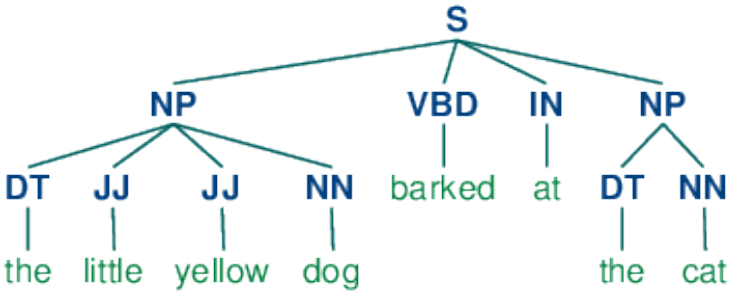


Figure 3.1. A tree of chunks of a sentence

3.1.1 Named entity recognition

An important task in this project was the extraction of relevant entities from the sentences written by the users.

For this purpose, a python library, `spacy` (*spaCy - Industrial-strength Natural Language Processing in Python* 2017), has been used.

An example of Named Entity Recognition performed on a simple phrase:

> Linus Benedict Torvalds [Person] (born December 28, 1969 [Date]) is a Finnish [NORP] software engineer who is the creator, and for a long time, principal developer of the Linux [ORG] kernel.

It can easily be noticed that this sentence contains a lot of information: named entity recognition can recognize where the relevant data is, and what kind of data type it is.

3.1.2 Language use differences in reviews and questions

After having collected enough data, it became apparent that there were relevant differences in language usage in Q/A and reviews. Questions, indeed, tend to be much shorter, use a simpler language and are much more straightforward than reviews. Although very short reviews can be found when dealing with a very diverse set of user generated content, they usually talk about multiple features of a product, while questions

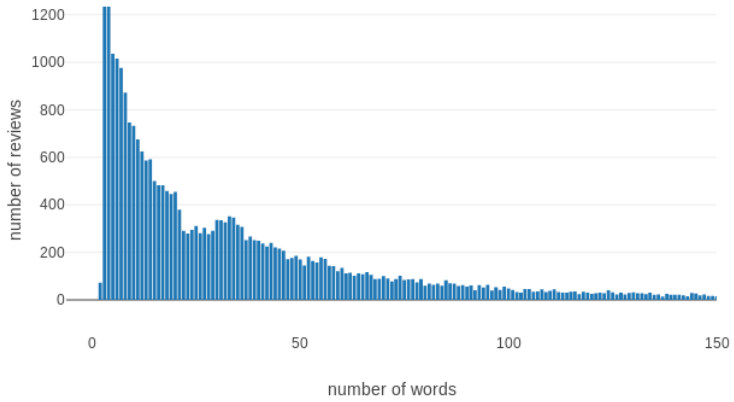


Figure 3.2. Frequency distribution of the number of words used in reviews

and answers usually address just one information. The structure of a sentence is also evidently different.

For this reason, I deemed appropriate to separate the methods in which Q/A and reviews are treated in the analysis process.

In 3.3 and 3.2, the frequency distribution of the number of words used in both reviews and questions/answers is shown. It can clearly be seen that it is much more common for questions to employ much less words, while reviews are more verbose.

3.2 Questions and Answers

3.2.1 Open-ended questions and closed-ended questions

Questions about a product on a online marketplace can be open-ended or closed-ended. The affiliation of each question to one of these categories entails major differences in how information is expressed.

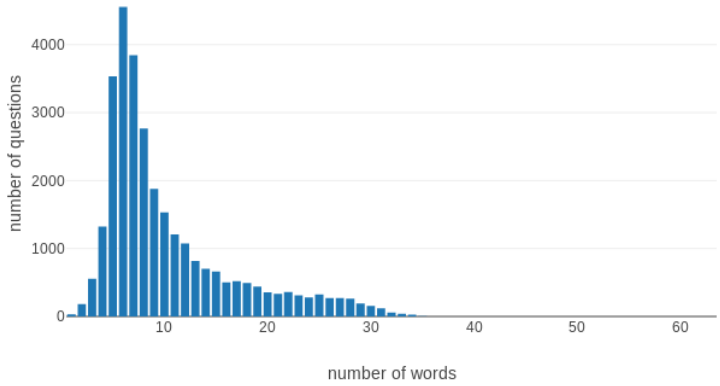


Figure 3.3. Frequency distribution of the number of words used in questions/answers

A **closed-ended** question is a question that accepts 'yes' or 'no' as answer. An example of a closed-ended question is:

> **Question:** Does this phone have a camera?
> **Answer:** Yes.

An **open-ended** question is, instead, a question that expects a more complex answer, such as a list, an explanation, a description. An example of an open-ended question is:

> **Question:** What features does this phone have?
> **Answer:** A camera, two sim slots and a headphone jack.

As you can see, information is expressed in a very straightforward way in closed-ended questions. If we, through natural language processing, manage to understand the object of the question, the answer is just a boolean value representing if the analyzed product corresponds to the requested quality or not.

Classifying the questions

The approach used in this section is based on the research of (McAuley and Yang 2016), who in turn based their approach on a paper from two researchers at Google, (He and Dai 2011).

According to He and Dai, recognizing a closed ended question is similar to matching a regular expression. If we, in fact, consider the following categories:

Be verbs : {am, is, are, been, being, was, were, . . . }
Modal verbs: {can, could, shall, should, will, would, . . . }
Auxiliary verbs: {do, did, does, have, has, . . . }

A closed-ended question is formed in this way:

$$[\ S_{be} \mid S_{modal} \mid S_{aux}\ ]\ .*\ +? \tag{3.1}$$

Still, this approach would catch as closed-ended questions constructs like "Is he married or not?" or "Does anybody know the price of this product?", that are clearly not closed-ended. Therefore, we exclude from the set of closed-ended questions the two following formulae:

$$[\ S_{be}\ ][\ a-z\ ]*[\ or\ ][\ a-z\ ]\ ? \tag{3.2}$$

and

$$[\ a-z\ ]*[\ anyone \mid anybody\ ][\ a-z\ ]*[\ tell \mid know\ ][\ a-z\ ]\ ? \tag{3.3}$$

He and Dai claim that this approach detects 91% of the questions correctly.

Classifying the answers

In the case of closed-ended questions, I analyzed the answers. The answers may be as simple as a "Yes" or "No", and in this situation the outcome is straightforward, but it is not always the case. Positive answers may appear in forms similar to:

> **Question:** Does it work in Argentina?
> **Answer:** It does work very well in Argentina.
>
> **Question:** Does it work in Argentina?
> **Answer:** Absolutely.
>
> **Question:** Does it work in Argentina?
> **Answer:** I think it does.

These are positive forms, but do not include the term "Yes".

Another detail to take into account is the possibility for a user to answer with an unclear or undefined answer. For this purpose, when classifying the answers, we consider a third "Unknown" label. Answers along the lines of:

> **Answer:** I don't know. **Answer:** I'm not sure.

or other unclear answers are considered Unknown answers.

3.2.2 Detecting the object of a question

With the term "named entities", we refer to parts of the phrase that indicate specific types of individuals. The names of organizations, names of people and dates are good examples of details that could be identified as named entities. The table below reports types and examples of the most common classifications of named entities: while organizations, persons, location, date, time, money and percent may be self-explanatory, "GPE" indicates "Geo-political entities" (e.g. country names), while "facility" refers to monuments and specific places.

ORGANIZATION	Google
PERSON	President Obama
LOCATION	Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 dollars
PERCENT	18.75%
FACILITY	Stonehenge
GPE	South East Asia

Named Entity Recognition (NER) is the practice of discovering and identifying all the named entities in a given text. First, the boundaries of the named entity need to be recognized (e.g. how many and which words compose a given NE). Then, the type of NE is identified, meaning which one of the aforementioned types does the NE belong to.

In our case, Named Entity Recognition is used for **closed-ended questions**, to identify the object of the question and the type it belongs to (e.g. "Does it ship to **Rome?**").

Named Entity Recognition can also be useful to find the answer to a question in a document, by recognizing and isolating the chunk that contains the answer. Suppose we have the question "Who was the first President of the US?", and we know that the answer is contained in the passage:

> The Washington Monument is the most prominent structure in Washington, D.C. and one of the city's early attractions. It was built in honor of George Washington, who led the country to independence and then became its first President.

We know that we should expect, as a response, a named entity classified as a PERSON. Although the phrases contains the word "Washington" twice, only one of them will actually be identified as a person.

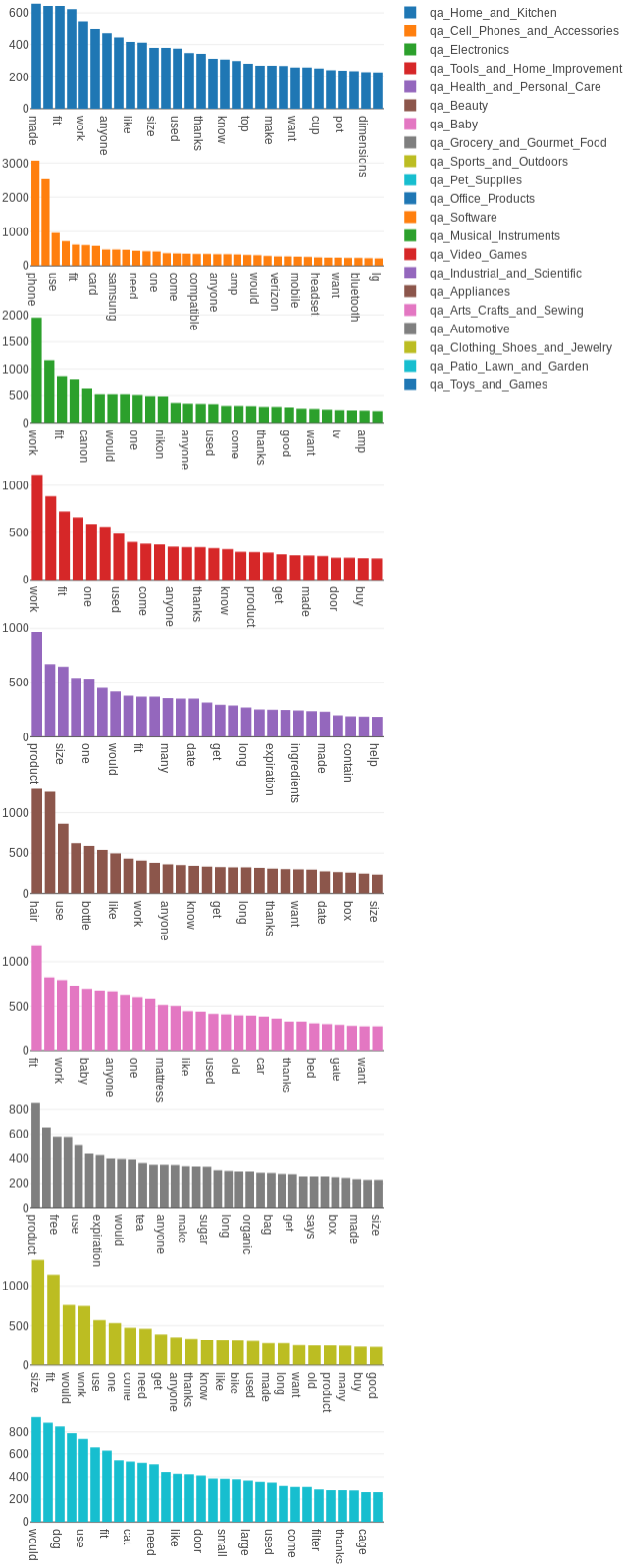


Figure 3.4. Most common words in some macrocategories on Amazon

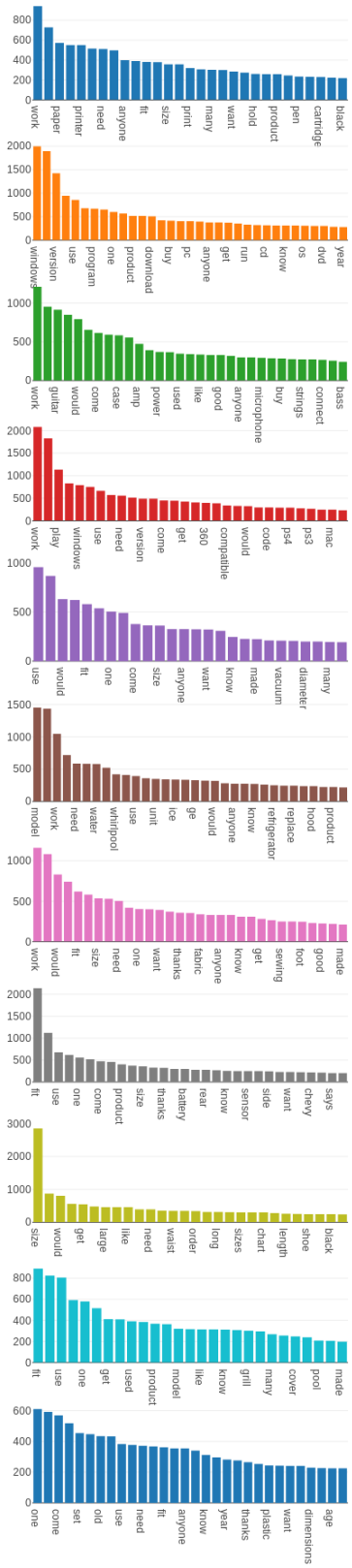


Figure 3.5. Most common words in some macrocategories on Amazon

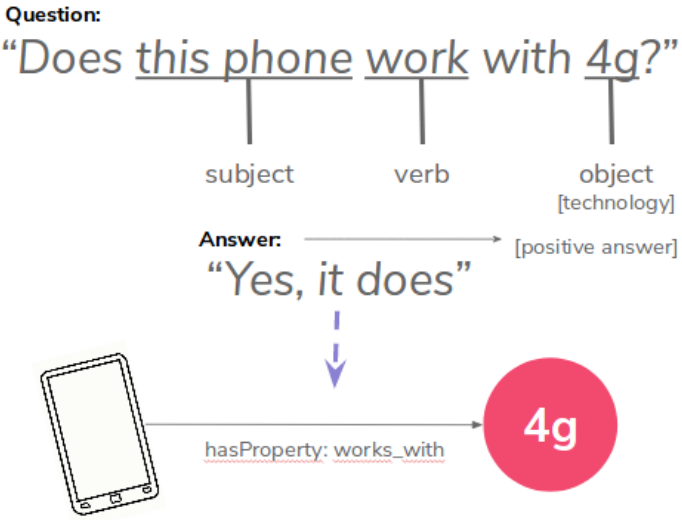


Figure 3.6. An example of a question translated to a property of an entity into an ontology

3.3 From NLP to Ontology population

An information extraction system is a system that looks through large bodies of text for specific types of entities and relations. The text is first segmented, tokenized, and then part-of-speech tagging is performed. The resulting data is used for named entity recognition: we look for specific types of entities. Based on the position of the words in the text, we try to determine if there exists a particular relationship between the entities.

Since I wanted to focus the object of the analysis on phones, I added three new types to the standard set of types used for Named Entity Recognition:

TECHNOLOGY	IR sensor, bluetooth, NFC
OS	Android, Marshmallow
CARRIER	Vodafone, Verizon

The main idea is that, after we learn which one is the subject of a phrase, we can say that <subject> has a relationship of type <verb> with the <object>, with adjectives if there are some.

In figure 3.6, it can be seen that a closed-ended question has been tagged with part-of-speech tags. Once we recognize that the object of the question (4G) is actually a technology, we can say that the phone has a relationship of type "works with" the technology 4G.

In the following section, some results extracted from the questions are shown:

```
[{'asin': 'B00QHYI488',
        'carriers_supported': {
                'at&t': [
                        (u'Does it work with at&t',
                u'Yes, it works'),
                (u'does this phone work with at&t',
                u'Yup.'),
                ...
            'metro pcs': [
                (u'Does this phone work with metro pcs?',
                u'Yes a fully unlocked GSM phone will work on
                   Metro PCS.'),
                (u'will it work with metro pcs?',
                u'Yes'),
                ...
            'sprint': [
                (u'Does this work with sprint carrier?',
                u"No, it doesn't work."),
                (u'Will this work with a sim card from sprint?',
                u"No, it doesn't work with sprint.")],
        'locations': {
        u'argentina': [
                (u'Does this phone work in Argentina? Thanks!',
            u'yes')],
            ...
        u'chile': [(u'it works in Chile?', u'Yup')],
        u'colombia': [
            (u'Will this phone work in colombia, south america?!',
            u'Yes, is perfect'),
            ...
```

```
30        u'egypt': [
31                (u'Is these phone work in Egypt or Saudi Arabia?',
32            u'Yes, it works on gsm network around the world.'),
33        u'eritrea': [
34                (u'Will this phone work in eritrea or ethiopia ?',
35            u'Yes. Works with all gsm')],
36            ...
37        u'europe': [(u'Hi is this phone work in Europe with
              European card',
38            u'Yes. Quad band (gsm networks)')],
39            ...
40        'made_in': {
41                u'taiwan': [
42                        (u'Is this made in Taiwan? Does i have any
                              logos on it (i.e. AT&T)?',
43            u"It doesn't have any logo it being import from Hong
                  Kong.")]},
44        'technologies_supported': {
45                'hotspot': [
46                        (u'Does it have a mobile hotspot?',
47            u"Yes it do. It's under the AT&T cluster of apps."),
48            (u'does this phone have hotspot capabilities',
49            u'Yes it does. Please check out gsmarena website for
                  specific details. Check reviews on youtube.'),
50            ...
51        '4g': [
52            (u'can this phone work on 4g band?',
53            u'Yes ! Of cure'),
54            (u'Is this phone compatible with 4g?',
55            u'compatible yes.....'),
56            ...
57        ...
```

Chapter 4

Ontology extraction and population

The idea of the whole project was to make data collected from product pages on the marketplace - any kind of data present on the page, from the product details that are directly provided by the vendor, like size and color of an item, to data present in user generated text, namely reviews and questions/answers.
The data collected from the analysis is composed by different data types, classes of items and relationships between them.
As defined in (Gruber 1993),

> A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called an ontology.

An ontology is perfectly apt to describe, in a human-readable and useful for [being interpreted by a computer] data and relationships between data.

4.1 Ontologies

An ontology serves to organize, provide and ease the access to knowledge about a specific domain, in order to make such knowledge readable and

useful to both computers and humans. It improves the process of information retrieval and reasoning, and makes the information interoperable between different applications (Zhou and Chaovalit 2008).

Still, according to Meersman (Meersman 2005), most of the ontologies are models that structure a fairly low amount of information about a very specific domain. He also states that automatically populated ontologies based on lexicons and thesauri are going to represent an important step in the field, as he says:

> “ It is unmistakable that with the advent of e-commerce, and the resulting natural language context of its related activities, that ontologies, lexicons and the thesauri and research in their use for system design and interpretation will receive a major market driven push”

Ontologies can be:

- language-specific ontologies (e.g. Italian, English)
- domain-specific ontologies (e.g. Laptop manufacturing, physics, corporate law)
- application-specific ontologies (e.g. Inventory control, airline reservations, conference organization)

As for Haider (Haider 2012),

> Mathematically, an ontology is a domain, while the relational schema is the range of the semantic interpretation mapping. Both can be seen as a representation of a commonly perceived reality and intimately related.

4.1.1 Ontology Learning

In the context of this thesis, the core concept is the idea of ontology learning, that is the automatical extraction of content from a collection of documents or relevant data with the purpose of creating an ontology. The automatical extraction of an ontology is currently a poignant topic,

because we have access to an enormous amount of data, but manually populating an ontology would be a time consuming task. As already discussed in the previous chapter, the method of ontology extraction applied in this project is based on natural language.

4.2 Choosing the formal language to represent the ontology

A series of languages has been developed to formally be able to formally describe and interpret an ontology.

The data used for this project has some peculiar characteristics that must be taken into account when discussing the best method to represent it:

- **Open World Assumption:** Closed World Assumption is the assumption that a statement which is not known to be true or false is to be considered false. In our case, using CWA would mean that if we don't know if an item has a characteristic, then we have to consider that item as NOT having that property. Due to the inability, in this context, to demonstrate that the information extracted is complete, we have to adopt the opposite of CWA, namely Open World Assumtpion, which allows to consider unknown values as unknown properties.
- **A need for extensive expressivity:** Languages have differences in what kind and how knowledge can be expressed through them. Given the diverse types of data and properties, extensive expressivity was a poignant characteristic to look for.
- **An evolving ontology:** Items on Amazon are constantly changing. More items are added, more reviews are written, and details may be updated. For this reason, we would ideally need to keep the contents of the ontology updated.
- **Possible inconsistencies:** Reviews written by diverse users may contradict themselves or the description of the item. Dealing with

the inconsistencies may have interesting outcomes, for example the discovery of a lie in the description of a product. Still, we need to manage the case in which the analyzed data contradicts itself.

We choose OWL to describe the ontology.

OWL (Dean et al. 2004) can be used with different syntaxes, the most common one being RDF/XML, that is based on RDF (Resource Description Framework) an XML standard. The interoperability of this standard, and the support it got, makes it useful for being used on a multitude of places - written via python script, used with a reasoner, or read via javascript.

4.3 Translating the data into an OWL ontology

To reach our final goal, we needed to define bridges from the data we extracted and the representation language we choose.

- Each item on Amazon is an OWL `class`, which describes a set of instances - that are the physical objects with the same name and same characteristics.
- Each category of items on Amazon is a `class`.
- Each property that an item can have is either a `Data Property` or an `Object Property`. More on this to be discussed in later sections.
- Properties extracted from the reviews may either be considered standard data types (strings, integers, floats, dates) or having their own class if there is more data regarding them specifically. Examples are the classes Brand and Author.

In the next sections, each one of the points is analyzed in more detail.

4.4 Populating the ontology via python

In order to populate the ontology programmatically, we used the **owlready** library for python 3.

4.4.1 RDF/XML representation of a product

The following is the simple, example representation, in OWL with RDF/XML, of a product on Amazon with a single property: the color of the item.

```
<owl:Class rdf:about="#
   Bonafide_HardwareTM__Smart_Phone_Repair_Tool_Kit_17
   _Piece_Set_Screw_Driver_Torx_Pentalobe_Cell_Tools">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
     owl#Thing"/>
  <rdfs:subClassOf rdf:resource="#Replacement_Parts"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#has_color"/>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/
         XMLSchema#string">red</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The product is a smart phone repair toolkit: products are represented as classes in the produced ontology. The class of the product is also a subclass of a broader category of products, in this case, "Replacement Parts". The color of the item is represented as a property of the class. Since each color did not need, in this domain, to be described as a class, this property is represented as a Data Property, of type string, and not as a relation with another object in the ontology.

The one above is a simplified example of a product into the ontology, included to show how an item would be represented into the ontology. As discussed in the previous chapters, each item has an extensive set of properties associated with it, but due to the verbosity of OWL syntax, including the full description of an item would require too much space for the pages of this thesis.

4.4.2 Taxonomy of classes

First of all, we extracted a taxonomy based on the item categories already present on Amazon. Each item, indeed, belongs to a category that describes a broader set of items. Each category may have child categories, and a parent category, designed mainly to ease navigating through products, but not always consistent - some categories may overlap in the items they contain.

These classes form a tree-like hierarchical structure.

For example, the item `Trivial Pursuit Game: Classic Edition` is classified as belonging to the class `Board Games`. The class `Board Games` is a subclass of the class `Games`, which is in turn subclass of `Toys & Games`.

`Super Jumbo Playing Cards`, instead, are categorized as `Standard Playing Card Decks`, subclass, again, of `Games`

The same structure of categories has been extracted from the page and represented in OWL as classes and `subClassOf` relationships. In the example below, it is shown how the category "Wearable Technologies" is a sub-category of "Electronics":

```
1 <owl:Class rdf:about="#Wearable_Technology">
2   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
       owl#Thing"/>
3   <rdfs:subClassOf rdf:resource="#Electronics"/>
```

```
4 </owl:Class>
```

Amazon does not offer a clear visualization or explanation of the categories. The tree of categories has been rebuilt by scraping the content of the page and extracting the data about how the product is classified. Every item taken into account in this thesis may belong to depth 2, 3 or 4 of the tree.

In figure 4.1, a part of the discussed taxonomy extracted from the documents is shown, about the product category "Electronics".

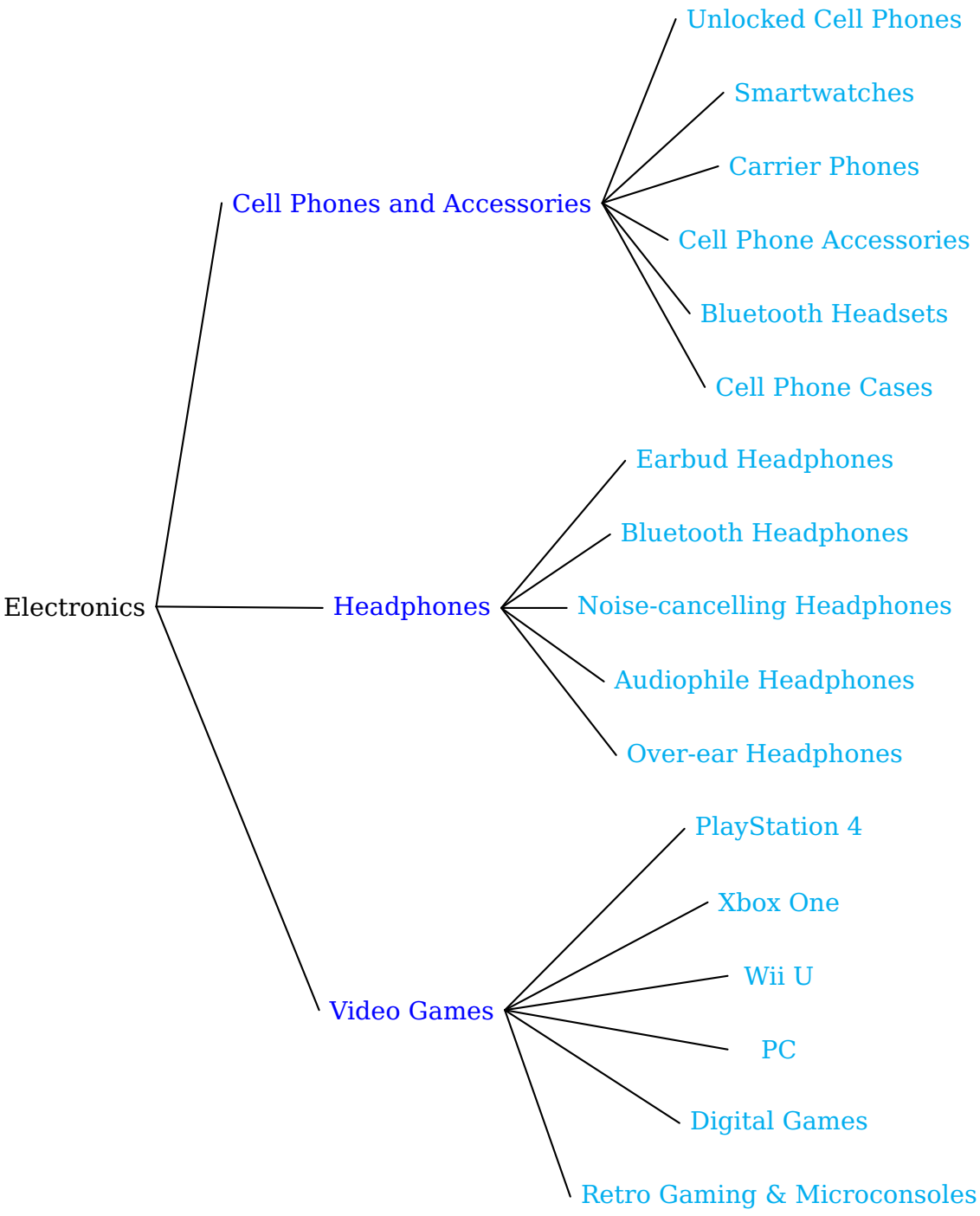


Figure 4.1. A small extract from the taxonomy of classes about the category "Electronics"

Chapter 5

Conclusions

5.1 Summary

The aim of this thesis was to propose a method for extracting and structuring information extracted from product reviews and more metadata on an online marketplace, a type of market in which reviews are particularly relevant.

After a research phase, in which we studied the possible approaches for extracting an ontology from natural language test, we experimented on what methodology and procedure would be the best for obtaining relevant information from the reviews.

5.2 Possible uses

We believe that the results of this project may be relevant not only for research purposes, but also for an eventual development for a useful tool for both clients of online marketplaces and sellers.

Suppose, for example, that you are a seller, and want to learn more about the public's opinion of your product based on reviews. One possible

approach would be using sentiment analysis on the text, or considering the star rating that accompanies each review. Still, that wouldn't inform you on what details of the product are producing bad or good reviews. For example, that your company produces phones, but one of your phones is having bad reviews. What details of the phone are causing the bad reviews? Is it the duration of the battery? Is the screen not bright enough? Or is the design of the phone considered ugly?

This kind of information is made easily accessible, allowing a seller to better identify the public's perception of the products, and without needing to read through the wealth of reviews that every product receives.

Now imagine that you are a customer, and you want to look for a specific product. You want a phone that has a durable battery, two sim slots, and you want it to have NFC. Amazon's search bar looks only through the items' names, and may therefore report not precise results. By having access to this ontology, instead, the search could be written as:

```
Phone and has_sim_slots value 2 and has_battery value "3300
        mAh" and supports_technology value "NFC"
```

5.3 Results

At the end of the data collection step, we had access to 62651 reviews, 37990 questions and answers, and 7927 product pages.

The ontology produced in the end contains 5243 products and 97870 axioms, extracted from the natural language of the products. The ontology can now be queried with DL queries, as Protegé supports them:

```
color value black and brand value Samsung
```

Would output every black Samsung item.

5.4 Future work

During the development of this thesis, there have been a number of details that we wanted to focus more attention on, but due to time constraints we have not been able to properly work on them.

- **Selection bias:** Selection bias is a distortion in a measure due to a sample selection that does not reflect the target population. In our case, selection bias may have influenced the results in the use of a different set of words or a different structure of the phrase. The question is: is the set of users that buys and reviews phones different from the set of users that buys skincare products? And, if that is the case, do they use language in a different way?
  One example of this is the case in which `amazon.com` is more likely to ship certain categories of products more than others, thus obtaining more foreign reviewers that use a simpler english, perhaps with a greater number of typos and grammatical errors. We don't have access to the nationality of the users, but a common question asked on phones, for example, is "does this phone work in <foreign country>?", that suggests that a significant number of users is interested in using the product in another country, and may therefore be using english as a second language.
  If it's true that different product categories have reviews with a different use of language, we may consider using a completely different model for each product category.

- **Manage grammatical errors:** Descriptions, reviews and Q/A are written by humans, and thus prone to typos and grammatical errors. These errors may influence the results of the natural language processing step: they could be reduced with an additional preprocessing step. One idea may be to produce a dictionary of correct words, and try to map each unrecognized word to the dictionary using Levenshtein distance [1], that measures similarity between strings by counting how many single-character edits are required to transform the first string into the second.

[1] `https://en.wikipedia.org/wiki/Levenshtein_distance`

Though, using Levenshtein distance would mean that we would be unable to distinguish words that are very similar but have different meanings. Consider, for example, the word (purposefully wrong) "hokse", that has the same Levenshtein distance to "house" and "horse": how would we understand which word was it meant to be? To some extent, we could consider a context in which the words are used, thus extract a probability for the original word to be one of the possible choices. Still, we wouldn't be able to distinguish "3g" from "4g". In this case, the problem of understanding if an unknown word is a typo or a new word should also be considered.

- **Detecting errors in item classification/item details:** As in the previous point, product descriptions are filled in by humans, and errors are possible. Sometimes, errors are evident: some items are miscategorized, some are lacking details, some have wrong details. The following are some examples of errors that we have come across:
  - Items without a name.
  - Items with impossible production years (e.g. produced in "217", probably typos)
  - Items classified in the wrong categories (e.g. a tabletop game classified as "kitchenware").

  Perhaps diminishing the miscategorization problem can be attempted via clustering on the products, finding out which products are outliers in their respective categories and possibly assigning them the category to which they are more similar to.
- **Development of a web interface to explore the ontology:** To make the ontology easily accessible by everyone we wanted to implement a web interface to visualize the entities and the relationships.
- **Bridge the produced ontology with other non domain-specific ontologies:** The content of the ontology could be expanded by finding a procedure to merge the produced ontology with other ones. For example, many concepts present in this ontology (like, the man-

ufacturing locations of the product) have exact correspondences in wordnet.

Bibliography

[1] Fatima N. AL-Aswadi and Chan Huah Yong. “A Study of Various Ontology Learning Systems from Text and a Look into Future”. In: *work* 2.5 (2017), pp. 9–10.

[2] *Beautiful Soup: We called him Tortoise because he taught us.* URL: `https://www.crummy.com/software/BeautifulSoup/` (visited on 11/24/2017).

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. “The semantic web”. In: *Scientific american* 284.5 (2001), pp. 28–37.

[4] Chris Biemann. “Ontology learning from text: A survey of methods.” In: *LDV forum*. Vol. 20. 2005, pp. 75–93.

[5] Paul Buitelaar and James Pustejovsky. *CoreLex: systematic polysemy and underspecification*. Brandeis University, 1998.

[6] Mike Dean et al. “OWL web ontology language reference”. In: *W3C Recommendation February* 10 (2004).

[7] José Ignacio Fernández Villamor et al. “A semantic scraping model for web resources-Applying linked data to web page screen scraping”. In: (2011).

[8] Thomas R. Gruber. “A translation approach to portable ontology specifications”. In: *Knowledge acquisition* 5.2 (1993), pp. 199–220.

[9] Syed Zeeshan Haider. *AN ONTOLOGY BASED SENTIMENT ANALYSIS: A Case Study*. 2012. URL: `http://www.diva-portal.org/smash/record.jsf?pid=diva2:552748` (visited on 03/22/2017).

[10] Jing He and Decheng Dai. “Summarization of yes/no questions using a feature function model”. In: *Asian Conference on Machine Learning*. 2011, pp. 351–366.

[11] Marti A. Hearst. “Automatic acquisition of hyponyms from large text corpora”. In: *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 1992, pp. 539–545.

[12] B. I. Intelligence. *Amazon accounts for 43% of US online retail sales*. URL: `http://www.businessinsider.com/amazon-accounts-for-43-of-us-online-retail-sales-2017-2` (visited on 11/19/2017).

[13] Sabina Jeschke et al. “Information extraction from mathematical texts by means of natural language processing techniques”. In: *Proceedings of the international workshop on Educational multimedia and multimedia education*. ACM, 2007, pp. 109–114.

[14] Joerg-Uwe Kietz, Alexander Maedche, and Raphael Volz. “A method for semi-automatic ontology acquisition from a corporate intranet”. In: *EKAW-2000 Workshop “Ontologies and Text”, Juan-Les-Pins, France, October 2000*. 2000.

[15] L. Liu, C. Pu, and W. Han. “XWRAP: an XML-enabled wrapper construction system for Web information sources”. In: *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. 2000, pp. 611–621. DOI: `10.1109/ICDE.2000.839475`.

[16] Alexander Maedche and Steffen Staab. “Learning ontologies for the semantic web”. In: *Proceedings of the Second International Conference on Semantic Web-Volume 40*. CEUR-WS. org, 2001, pp. 51–60.

[17] Alexander Maedche and Steffen Staab. “The text-to-onto ontology learning environment”. In: *Software Demonstration at ICCS-2000-Eight International Conference on Conceptual Structures*. Vol. 38. sn, 2000.

[18] Julian McAuley and Alex Yang. “Addressing Complex and Subjective Product-Related Queries with Customer Reviews”. en. In: ACM Press, 2016, pp. 625–635. ISBN: 978-1-4503-4143-1. DOI: `10.1145/2872427.2883044`. URL: `http://dl.acm.org/citation.cfm?doid=2872427.2883044` (visited on 12/03/2017).

[19] Robert Meersman. “The use of lexicons and other computer-linguistic tools in semantics, design and cooperation of database systems”. In: *STAR* 1999.02 (2005).

[20] Ion Muslea, Steven Minton, and Craig A. Knoblock. “Hierarchical wrapper induction for semistructured information sources”. In: *Autonomous Agents and Multi-Agent Systems* 4.1 (2001), pp. 93–114.

[21] Roberto Navigli, Paola Velardi, and Aldo Gangemi. “Ontology learning and its application to automated terminology translation”. In: *IEEE Intelligent systems* 18.1 (2003), pp. 22–31.

[22] Primal Pappachan et al. “Building a Mobile Applications Knowledge Base for the Linked Data Cloud.” In: *MoDeST@ ISWC*. 2015, pp. 14–25.

[23] Brian Roark and Eugene Charniak. “Noun-phrase co-occurrence statistics for semi-automatic semantic lexicon construction”. In: *arXiv:cs/0008026* (Aug. 2000). arXiv: cs/0008026. URL: `http://arxiv.org/abs/cs/0008026` (visited on 01/05/2018).

[24] Vincent Schickel-Zuber and Boi Faltings. “Using hierarchical clustering for learning theontologies used in recommendation systems”. In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 599–608.

[25] *spaCy - Industrial-strength Natural Language Processing in Python*. URL: `https://spacy.io/index` (visited on 12/09/2017).

[26] Michael Thelen and Ellen Riloff. “A bootstrapping method for learning semantic lexicons using extraction pattern contexts”. In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002, pp. 214–221.

[27] Paola Velardi, Stefano Faralli, and Roberto Navigli. “Ontolearn reloaded: A graph-based algorithm for taxonomy induction”. In: *Computational Linguistics* 39.3 (2013), pp. 665–707.

[28] Raphael Volz et al. “Pruning-based identification of domain ontologies”. In: *J. UCS* 9.6 (2003), pp. 520–529.

[29] David Yarowsky. “Unsupervised word sense disambiguation rivaling supervised methods”. In: *Proceedings of the 33rd annual meeting*

on Association for Computational Linguistics. Association for Computational Linguistics, 1995, pp. 189–196.

[30] Lina Zhou and Pimwadee Chaovalit. “Ontology-supported polarity mining”. en. In: *Journal of the American Society for Information Science and Technology* 59.1 (Jan. 2008), pp. 98–110. ISSN: 1532-2890. DOI: 10.1002/asi.20735. URL: http://onlinelibrary.wiley.com/doi/10.1002/asi.20735/abstract (visited on 01/05/2018).