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

Co-Advisor

Dr. Nome Cognome

Academic Year 2016/2017

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: December 29, 2017

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.

Specifically, 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. Matteo, Alessandro, Lorenzo, Alessio, Gabriele [etc]. Along with them, I want to mention a couple of colleagues from Codemotion, Massimo and Bruno, who made me really understand that creativity and technology weren't mutually exclusive concepts, and with whom I had a productive friendship that made me grow up []

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

Contents

1 Introduction **1**

2 Related works **3**

3 Collecting Data **5**
3.1 Amazon 5
3.2 The challenges 5
3.2.1 Amazon API 5
3.3 Scraping 6
3.3.1 Scraping a single page 7
3.3.2 Dynamic pages 8
3.4 Scraping on a larger scale 10
3.4.1 Identifying bottlenecks 10
3.4.2 Multithreading 10
3.5 Tampering requests 11
3.5.1 Proxies 11
3.5.2 Crafting user agents 12
3.6 Flow diagram of the scraper 13
3.7 amazon-scraper 14
3.8 Shape of the collected data 14

4 Processing the information **21**
4.1 Natural language processing 21
4.1.1 Pattern recognition in strings 22
4.1.2 Named entity recognition 22
4.1.3 Language use differences in reviews and questions 23
4.2 Questions and Answers 23
4.2.1 Open-ended questions and closed-ended questions 23
4.2.2 Detecting the object of a question 25
4.2.3 Differences in Q/A language for different product categories 26
4.3 Reviews 26
4.3.1 Sentence structure 29
4.3.2 Subjects, verbs and objects 29
4.3.3 Relations 29

5 Ontology extraction and population 31
5.1 Ontologies . . . 31
5.1.1 Ontology Learning . . . 32
5.2 Choosing the formal language to represent the ontology . . . 32
5.3 Translating the data into an OWL ontology . . . 33
5.3.1 RDF/XML representation of a product . . . 34
5.4 Taxonomy of classes . . . 34
5.5 Populating the ontology via python . . . 35
5.6 dataProperties and objectProperties . . . 35
5.7 Querying the ontology . . . 35

6 Results 37

7 Future work 39

8 Conclusioni 41

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, 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, judjing 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.

[sellers buying reviews]
quotes on trust system

Since reviews are so relevant, they have been the object of a series of studies. They are, infact, made of unstructured informations about a product: the data they contain, if succesfully extracted, may help in identifying precisely the features of a product.

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 project presented in this thesis is aimed at extracting and structuring the information contained in user generated content regarding products on online marketplaces.

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

Chapter 2

Related works

In 2012, S.Z. Haider [4] 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.

Biemann [2] published in 2005 a survey of methods to study the different approaches at extracting data from unstructured text in order to build an ontology.

Julian McAuley and Alex Yang [6] used an approach based on machine learning to classify questions and answers in the Questions/Answers section of Amazon.

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

- include information from Questions/Answers sections as well as reviews
- include a much bigger range of products and categories of products
- extract the structure of the ontology from the data

Chapter 3

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 aren't so straightforward. They are, infact, 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.

3.1 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 [5], Amazon is accountable for 43% of online sales in the US in 2016.

TODO: insert more data about amazon, explain why amazon is important

3.2 The challenges

3.2.1 Amazon API

Amazon offers an API that makes available some of the information I wanted to examine. Unfortunately, not the request limits imposed to API users nor the

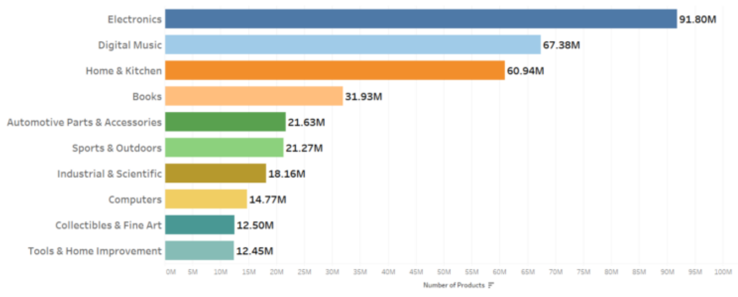


Figure 3.1. a comparison between different product categories on Amazon

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.

3.3 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.

3.3.1 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 successfull, 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 parsetree.
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 [1]. Processing the page in this way allows me to navigate HTML easily and query the contents.

In the code example above, at 3.3.1, retrieving the title of the received page would be written like this:

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

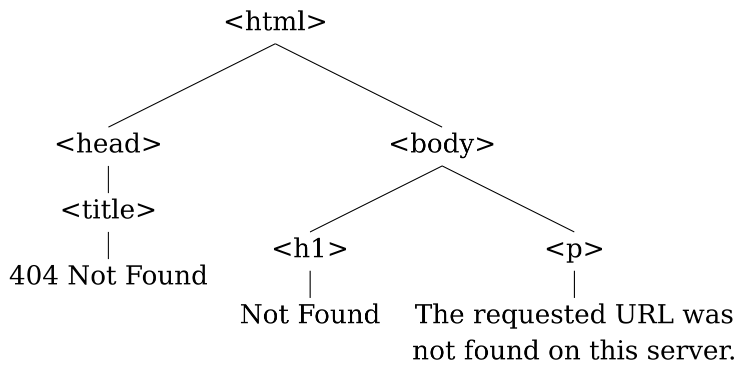


Figure 3.2. The parsetree generated from 3.3.1

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 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 3.3, 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.

3.3.2 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.

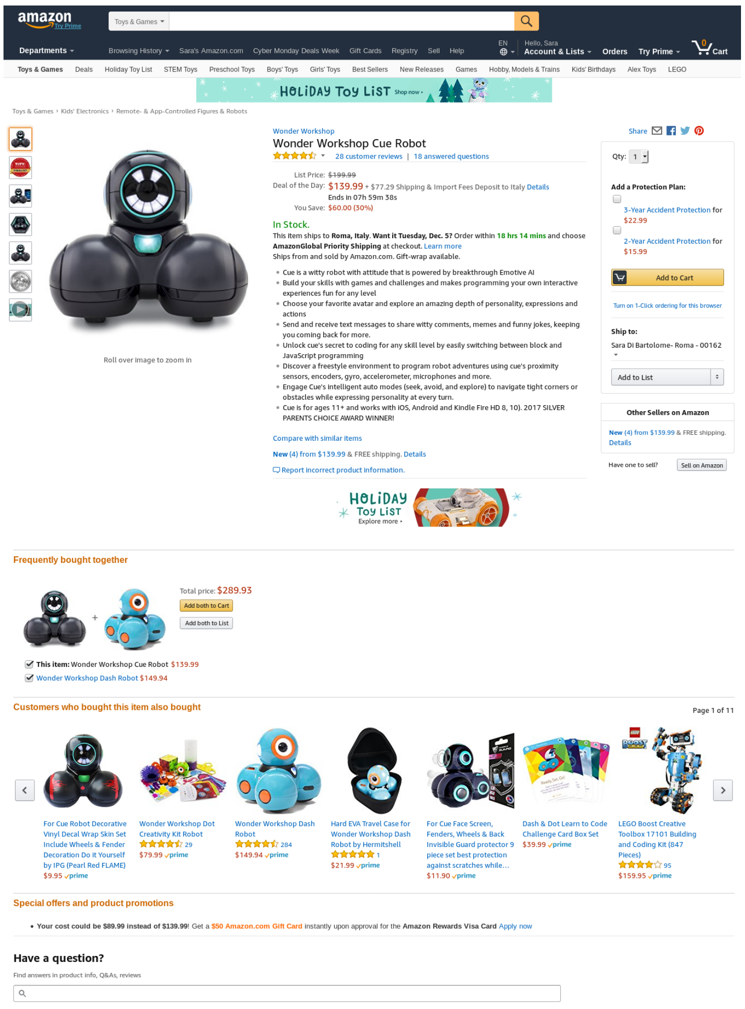
Think of facebook: each user sees the same website at the same address in a unique way, depending on the user's account, the cookies he has stored in its website, the times and types of content that the user's friends posted.

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: [example]

Therefore, a web server does not store a single web page for each possible



Wonder Workshop

Wonder Workshop Cue Robot

28 customer reviews | 18 answered questions

List Price: ~~$199.99~~
Deal of the Day: $139.99 + $77.29 Shipping & Import Fees Deposit to Italy Details
Ends in 07h 59m 38s
You Save: $60.00 (30%)

In Stock.
This item ships to **Roma, Italy. Want it Tuesday, Dec. 5?** Order within 18 hrs 14 mins and choose **AmazonGlobal Priority Shipping** at checkout. Learn more
Ships from and sold by Amazon.com. Gift-wrap available.

- Cue is a witty robot with attitude that is powered by breakthrough Emotive AI
- Build your skills with games and challenges and makes programming your own interactive experiences fun for any level
- Choose your favorite avatar and explore an amazing depth of personality, expressions and actions
- Send and receive text messages to share witty comments, memes and funny jokes, keeping you coming back for more.
- Unlock cue's secret to coding for any skill level by easily switching between block and JavaScript programming
- Discover a freestyle environment to program robot adventures using cue's proximity sensors, encoders, gyro, accelerometer, microphones and more.
- Engage Cue's intelligent auto modes (seek, avoid, and explore) to navigate tight corners or obstacles while expressing personality at every turn.
- Cue is for ages 11+ and works with iOS, Android and Kindle Fire HD 8, 10). 2017 SILVER PARENTS CHOICE AWARD WINNER!

Compare with similar items

New (4) from $139.99 & FREE shipping. Details

Report incorrect product information.

Qty: 1

Add a Protection Plan:
☐ 3-Year Accident Protection for $22.99
☐ 2-Year Accident Protection for $15.99

Add to Cart

Turn on 1-Click ordering for this browser

Ship to:
Sara Di Bartolome- Roma - 00162

Add to List

Other Sellers on Amazon
New (4) from $139.99 & FREE shipping. Details

Have one to sell? Sell on Amazon

Frequently bought together

Total price: $289.93

✓ **This item:** Wonder Workshop Cue Robot $139.99
✓ Wonder Workshop Dash Robot $149.94

Customers who bought this item also bought

For Cue Robot Decorative Vinyl Decal Wrap Skin Set Include Wheels & Fender Decoration Do it Yourself by IPG (Pearl Red FLAME) $9.95 prime

Wonder Workshop Dot Creativity Kit Robot 29 $79.99 prime

Wonder Workshop Dash Robot 284 $149.94 prime

Hard EVA Travel Case for Wonder Workshop Dash Robot by Hermitshell 1 $21.99 prime

For Cue Face Screen, Fenders, Wheels & Back Invisible Guard protector 9 piece set best protection against scratches while… $11.90 prime

Dash & Dot Learn to Code Challenge Card Box Set $39.99 prime

LEGO Boost Creative Toolbox 17101 Building and Coding Kit (847 Pieces) 95 $159.95 prime

Special offers and product promotions

- **Your cost could be $89.99 instead of $139.99**! Get a $50 Amazon.com Gift Card instantly upon approval for the **Amazon Rewards Visa Card** Apply now

Have a question?

Find answers in product info, Q&As, reviews

Figure 3.3. a product page on amazon 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.

[Luckily, Amazon uses javascript to lazy-load pictures on product pages]

3.4 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]

3.4.1 Identifying bottlenecks

Some of the problems associated with large-scale scraping are related to I/O bottlenecks and memory management.

3.4.2 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]

3.5 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.

3.5.1 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 aren't 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.

3.5.2 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:

```
{
        '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.

3.6 Flow diagram of the scraper

The flow diagram at 3.4 descibes an overview of in what sequence the scraper program applies the techinques described in the previous chapters.

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.

3.7 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.

3.8 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",
```

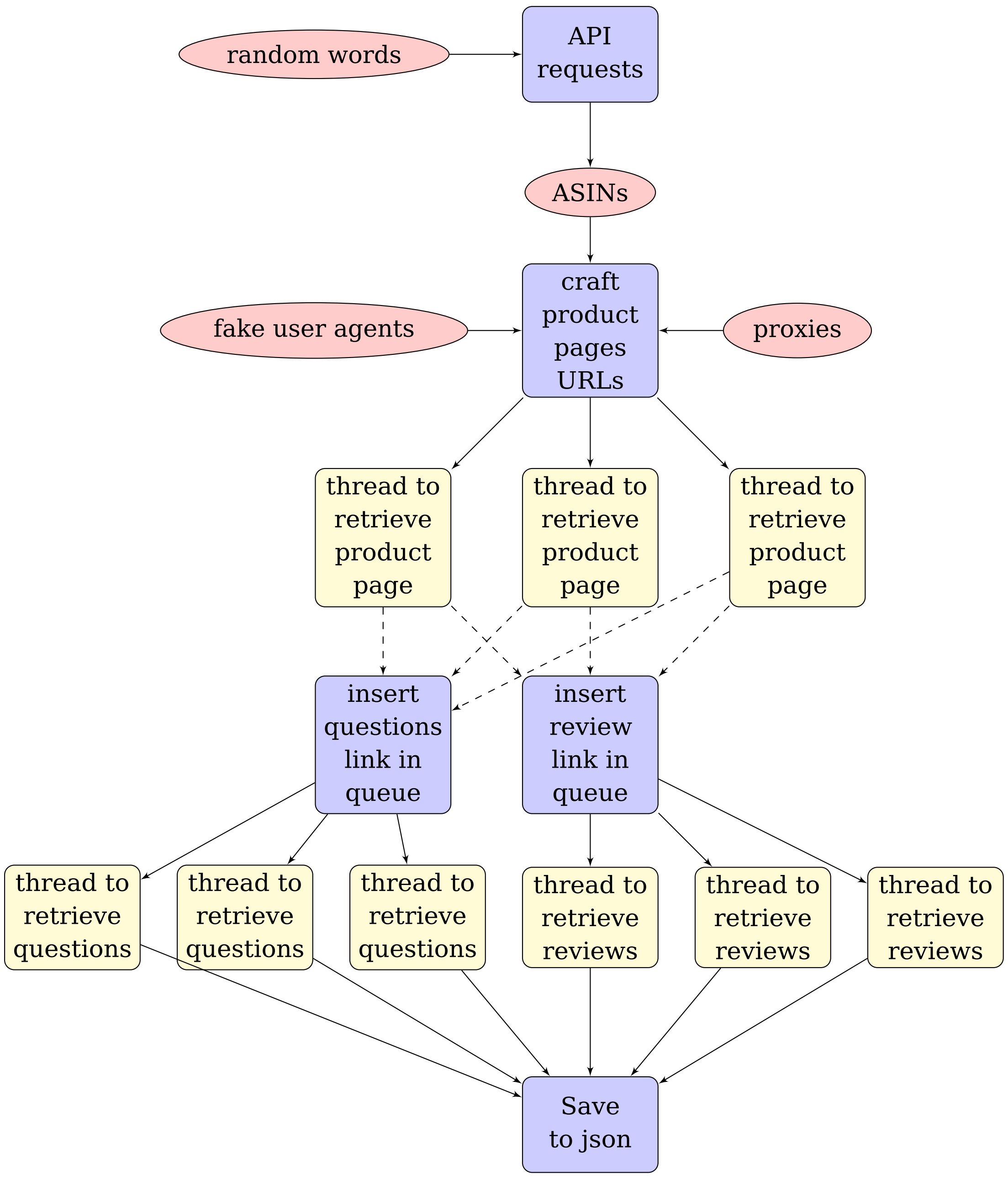

random words
API requests
ASINs
fake user agents
craft product pages URLs
proxies
thread to retrieve product page
thread to retrieve product page
thread to retrieve product page
insert questions link in queue
insert review link in queue
thread to retrieve questions
thread to retrieve questions
thread to retrieve questions
thread to retrieve reviews
thread to retrieve reviews
thread to retrieve reviews
Save to json

Figure 3.4. 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 3.5. `amazon-scraper` usage

```
        "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 4

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 infact 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 didn't 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 didn't 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.

[add content here]

4.1 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*)
- Assignign to the structure a semantical meaning

[decision trees for tagging vs machine learning techniques]

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

The 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.

4.1.1 Pattern recognition in strings

4.1.2 Named entity recognition

[explain better how named entity recognition works]

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`, has been used. `spacy` features a default model that maps entities to []

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

> Linus Benedict Torvalds $^{\text{Person}}$ (born December 28, 1969 $^{\text{Date}}$) is a Finnish $^{\text{NORP}}$ software engineer who is the creator, and for a long time, principal developer of the Linux $^{\text{ORG}}$ 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.

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

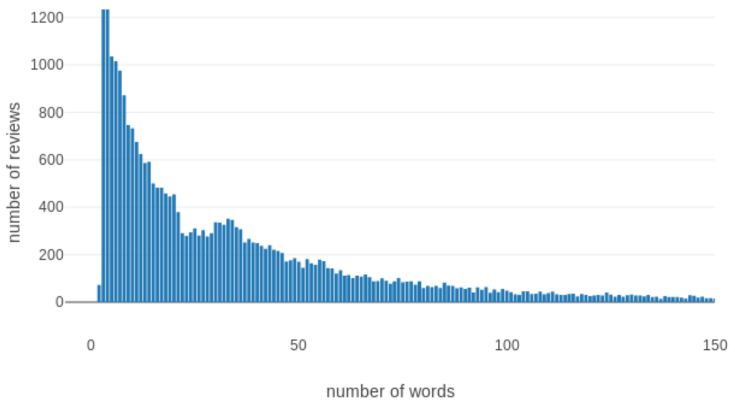


Figure 4.1. Frequency distribution of the number of words used in reviews

4.1.3 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 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 4.2 and 4.1, 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.

Another indicator of a more complex use of the language in reviews is [variance in frequency distribution of words]

[insert average question length for type of question] [insert word frequency distribution for each category]

4.2 Questions and Answers

4.2.1 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

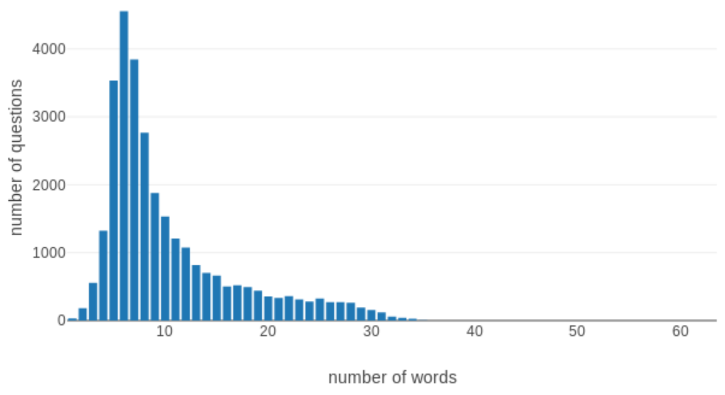


Figure 4.2. Frequency distribution of the number of words used in questions/answers

differences in how information is expressed.

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

Based on the work of [insert link], I used a simple Naive Bayes classifier to discern closed-ended from open-ended questions. The classifier has been trained on a pre-labeled dataset of [N] questions and answers. The labels were "open" or "Y/N".

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.

4.2.2 Detecting the object of a question

[better] Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on. 5.1 lists some of the more commonly used types of NEs. These should be self-explanatory, except for "Facility": human-made artifacts in the domains of architecture and civil engineering; and "GPE": geo-political entities such as city, state/province, and country.

ORGANIZATION	Georgia-Pacific Corp., WHO
PERSON	Eddy Bonte, President Obama
LOCATION	Murray River, Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 million Canadian Dollars, GBP 10.40
PERCENT	twenty pct, 18.75%
FACILITY	Washington Monument, Stonehenge
GPE	South East Asia, Midlothian

The goal of a named entity recognition (NER) system is to identify all textual mentions of the named entities. This can be broken down into two sub-tasks: identifying the boundaries of the NE, and identifying its type. While named entity recognition is frequently a prelude to identifying relations in Information Extraction, it can also contribute to other tasks. For example, in Question Answering (QA), we try to improve the precision of Information Retrieval by recovering not whole pages, but just those parts which contain an answer to the user's question. Most QA systems take the documents returned by standard Information Retrieval, and then attempt to isolate the minimal text snippet in the document containing the answer. Now suppose the question was Who was the first President of the US?, and one of the documents that was retrieved contained the following 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.

Analysis of the question leads us to expect that an answer should be of the form X was the first President of the US, where X is not only a noun phrase, but also refers to a named entity of type PERSON. This should allow us to ignore the first sentence in the passage. While it contains two occurrences of Washington, named entity recognition should tell us that neither of them has the correct type.

4.2.3 Differences in Q/A language for different product categories

4.3 Reviews

[better]

- Information extraction systems search large bodies of unrestricted text for specific types of entities and relations, and use them to populate well-organized databases. These databases can then be used to find answers for specific questions.
- The typical architecture for an information extraction system begins by segmenting, tokenizing, and part-of-speech tagging the text.
- The resulting data is then searched for specific types of entity. Finally, the information extraction system looks at entities that are mentioned near one another in the text, and tries to determine whether specific relationships hold between those entities.
- Entity recognition is often performed using chunkers, which segment multi-token sequences, and label them with the appropriate entity type. Common entity types include ORGANIZATION, PERSON, LOCATION, DATE, TIME, MONEY, and GPE (geo-political entity).
- Chunkers can be constructed using rule-based systems, such as the RegexpParser class provided by NLTK; or using machine learning techniques.

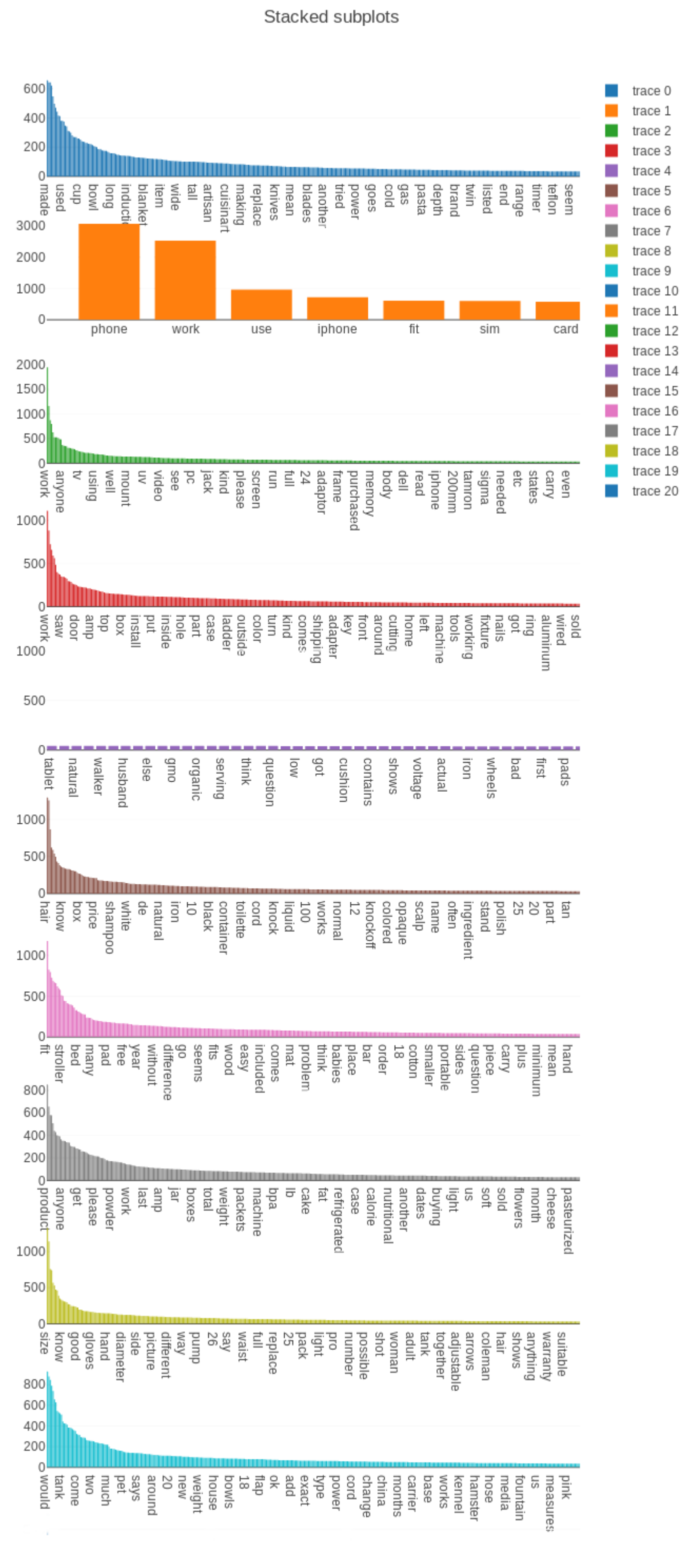


Figure 4.3. A tree of chunks of a sentence

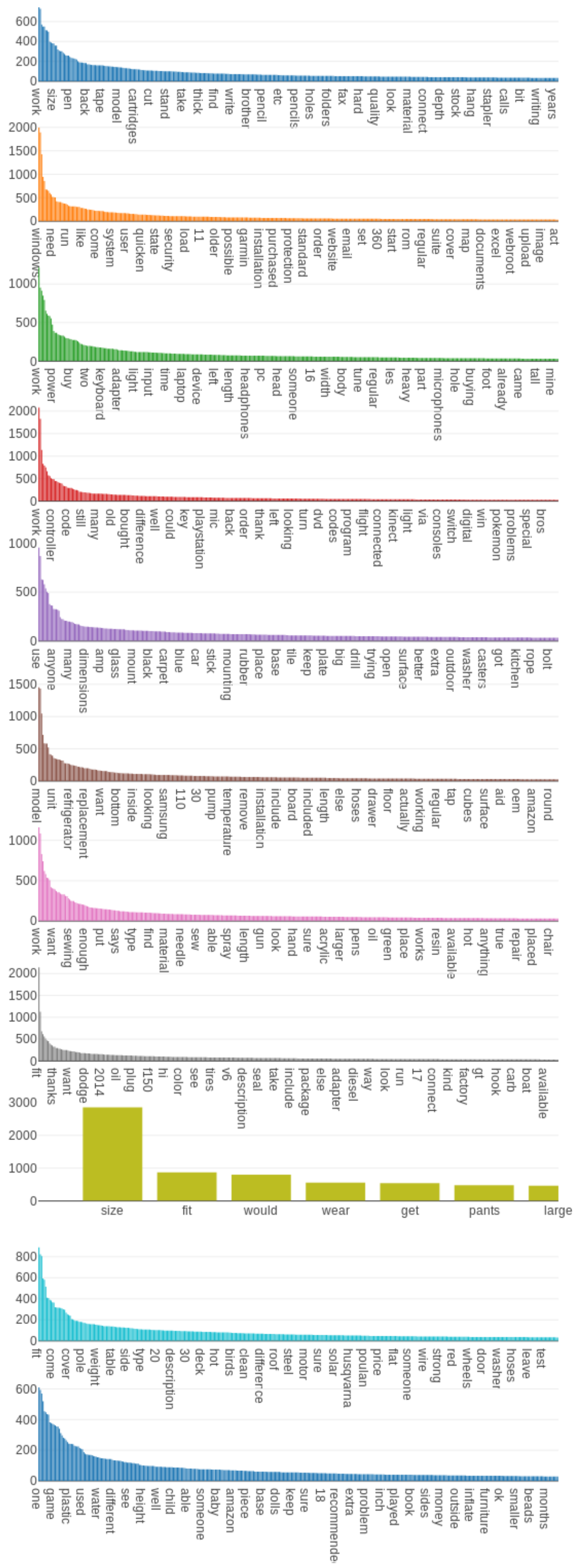


Figure 4.4. A tree of chunks of a sentence

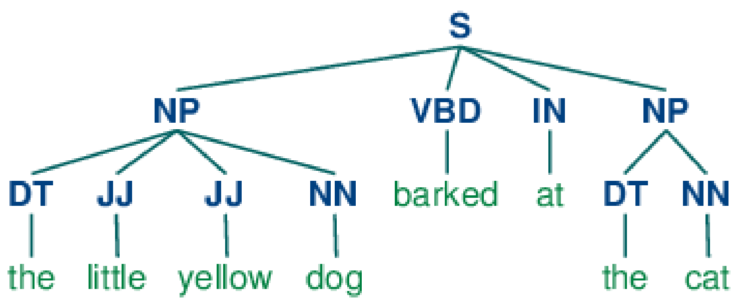


Figure 4.5. A tree of chunks of a sentence

In either case, part-of-speech tags are often a very important feature when searching for chunks.

- Although chunkers are specialized to create relatively flat data structures, where no two chunks are allowed to overlap, they can be cascaded together to build nested structures.
- Relation extraction can be performed using either rule-based systems which typically look for specific patterns in the text that connect entities and the intervening words; or using machine-learning systems which typically attempt to learn such patterns automatically from a training corpus.

4.3.1 Sentence structure

4.3.2 Subjects, verbs and objects

4.3.3 Relations

Chapter 5

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] [3],

> 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.

5.1 Ontologies

[better] The main aim of ontology is to provide knowledge about specific domains that are understandable by both the computers and developers. It also helps to interpret a text review at a finger granularity with shared meanings and provides a sound semantic ground of machine- understandable description of digital content. Ontology improves the process of information retrieval and reasoning thus results in making data interoperable between different applications (Zhou, Chaovalit, 2007) According to Meersman(2005), most of the ontologies in the community of information systems are known as data models that are mainly used for structuring a fairly narrow application domain. He claimed that ontologies that includes lexicons and thesauri may be a useful first step in providing and formalizing the semantics of information representation. According to Meersman (2005), in near future these ontologies will act as a semantic

domain for the information systems and will be very useful.He also predicted with authenticity that:

> “ 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”

(Meersman,2005,p.02) According to Meersman, (2005), a lexicon is defined as a language-specific ontology, for e.g English, Polish. Whereas thesaurus is defined as either a domain-specific ontology or an application specific ontology. Manufacturing, Laptop-manufacturing, Naïve physics, corporate law, Ontology theory are examples of domain specific ontologies, whereas Inventory Control, Airline Reservations, Conference Organization are examples of application specific ontology. Domain specific ontology and application specific ontology can be distinguished intuitively as the differences between the two types are not distinct. There is difference between ontologies and conceptual schemas but both are intimately related and are used to represent commonly perceived reality. Mathematically, ontology is the 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.

5.1.1 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.

5.2 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 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.

5.3 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.

5.3.1 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.

```xml
<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 didn't 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.

5.4 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":

```xml
<owl:Class rdf:about="#Wearable_Technology">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#
     Thing"/>
  <rdfs:subClassOf rdf:resource="#Electronics"/>
</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 5.1, a part of the discussed taxonomy extracted from the documents is shown, about the product category "Electronics".

5.5 Populating the ontology via python

[Populating the ontology via python has been done via the owlready library.]

5.6 dataProperties and objectProperties

In OWL, we have two possible kinds of properties that a member of the ontology can have:

- DataProperty
- ObjectProperty

5.7 Querying the ontology

[Having an ontology like this one allows us to query the contents] Protegé supports DL queries on the knowledge base.

`color value black and brand value Samsung`

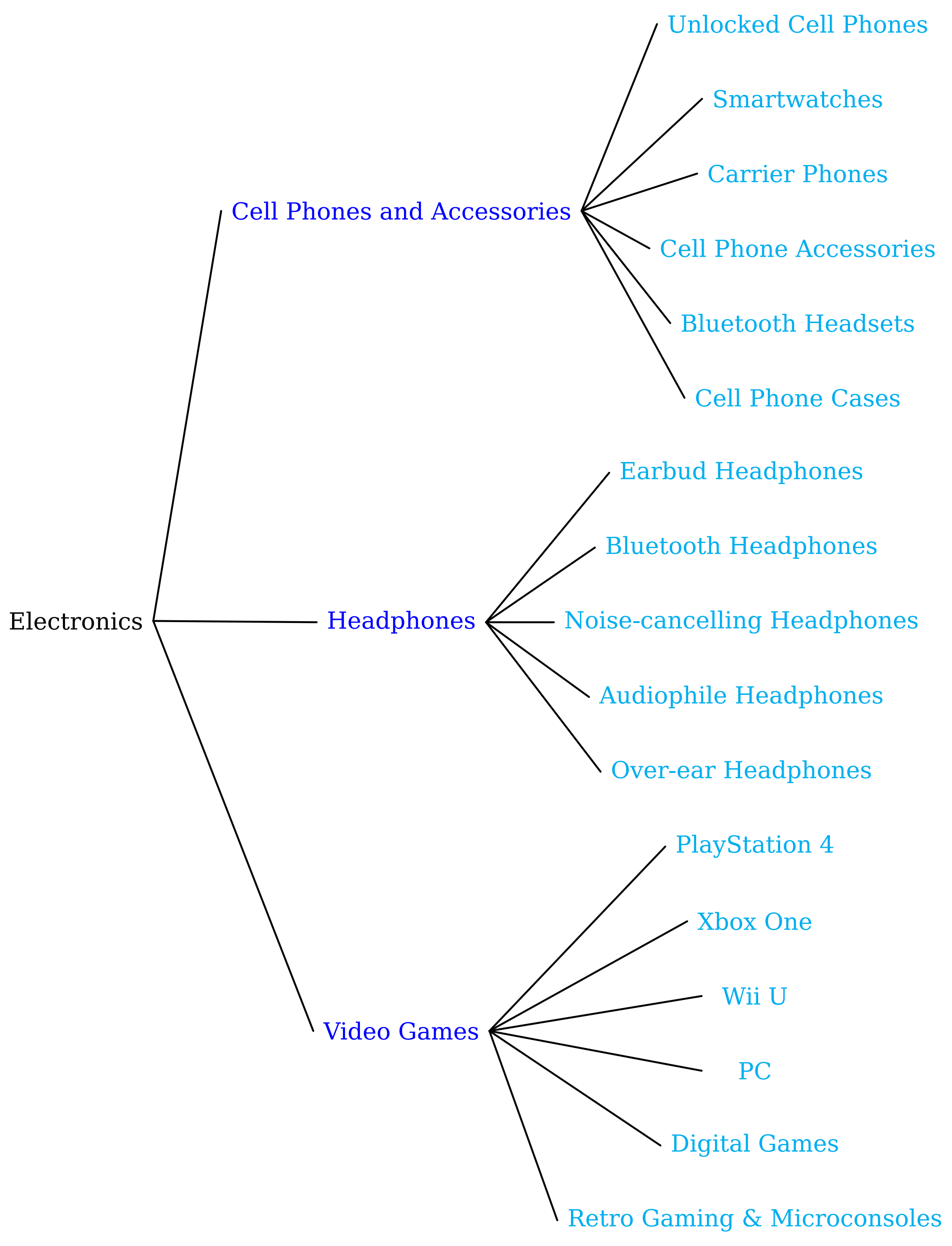


Figure 5.1. A small extract from the taxonomy of classes about the category "Electronics"

Chapter 6

Results

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.

At the end of the data collection step, we had access to [] reviews, [] questions and answers, and [] product pages. The natural language analysis part produced

Chapter 7

Future work

Selection bias: users who post reviews may be a different subset than users who do not

Errors in classifying items or naming

Chapter 8

Conclusioni

Bibliography

[1] Beautiful Soup: We called him Tortoise because he taught us. Available from: `https://www.crummy.com/software/BeautifulSoup/`.

[2] Biemann, C. Ontology learning from text: A survey of methods. In *LDV forum*, vol. 20, pp. 75–93 (2005).

[3] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge acquisition*, **5** (1993), 199.

[4] Haider, S. Z. *AN ONTOLOGY BASED SENTIMENT ANALYSIS: A Case Study* (2012). Available from: `http://www.diva-portal.org/smash/record.jsf?pid=diva2:552748`.

[5] Intelligence, B. I. Amazon accounts for 43% of US online retail sales. Available from: `http://www.businessinsider.com/amazon-accounts-for-43-of-us-online-retail-sales-2017-2`.

[6] McAuley, J. and Yang, A. Addressing Complex and Subjective Product-Related Queries with Customer Reviews. pp. 625–635. ACM Press (2016). ISBN 978-1-4503-4143-1. Available from: `http://dl.acm.org/citation.cfm?doid=2872427.2883044`, `doi:10.1145/2872427.2883044`.