Enrichment of Location Databases

Maria Sofia Almeida
maria.d.almeida@tecnico.ulisboa.pt
Instituto Superior Técnico
Portugal

Abstract

Nowadays, with the massive amount of data available online, there is a growing need to integrate different types of data into unified systems, specifically in a geographical context. Integrated tools that provide both geographical and entity related data already exist, however, the information they contain is often unreliable, not current or has restricted access. Therefore, in this work, we propose a method to enrich current location databases with information related to their entities. Our work concerns two problems, web extraction and classification, so we base our architecture on studies of those fields. The system was built with the *OpenStreetMap* project's data and uses several web scraping and slot filling methods, in order to extract an entity's attributes from the Web and gather them in a single information tool. To ensure the data is current and trustworthy, the system extracts each entity's information from its official website. This data is then processed by the system, which uses regular expressions, language models and machine learning techniques to classify each possible attribute value. The system's evaluation is done with metrics, such as Precision, Recall and F-measure. Our results show that the proposed system performs better when using regular expressions or a mixed approach of regular expressions and machine learning algorithms, depending on the attribute.

Keywords: Web Scraping, Slot Filling, Natural Language Processing, Geographical Information System, OpenStreetMap

1 Introduction

With the explosive growth of the World Wide Web, each day, more and more information is accessible online. This increase of data at one's disposal leads to a rising need of processing and making it available quickly and freely so that this information can be used by other user oriented platforms and services. There is a considerable number of this type of services, mainly data analysis applications, that benefit greatly from it.

Specifically in a geographic context, there is a need for integration and validation of information, which the Web can bridge. Thus this work's purpose, which is to evaluate and improve the data contained in Geographic Information Systems (GIS), such as *OpenStreetMap* (OSM). According to Chang[3], GIS are computer systems that capture, store, query, analyze and display geospatial data, which is data that describes both the locations and characteristics of spatial features.

Finding official information about an entity through existing geographical location sources is a somewhat limited process. Limited in the sense that these sources mostly only focus on an entity's geographical knowledge, containing only its name and geographic coordinates and no other type of data (e.g. opening hours or phone number), which are available on the Web through search queries. Even when they do take these types of data into account, they often provide display limitations and require the entity itself to update such data on the platform. Those services which do not include such restrictions might have other issues, like reliability or currency, which is OSM's case as its information is inserted by random users at random times.

Therefore, the challenge addressed in this work is the development of an automated tool that complements geographical location sources, more specifically, the OSM project, with reliable data taken from the Web. The solution to the problem in question will involve the application of Slot Filling (SF) techniques to data extracted from the Web, through Web Scraping (WS) techniques. The proposed system's objective is to finally obtain current detailed information from a subset of geographical entities. However, since the SF task is highly application and domain dependent[2], in this case, a set of selected attributes for the SF task was chosen: addresses, opening hours and phone numbers, so the system will only concern the extraction and processing of these.

1.1 Organization of the Document

This article is organized in 5 chapters, as follows: Section 2 that briefly explores the state-of-the-art of the subject, Section 4 concerns a detailed description of our solution to the WS and SF problems, which is applied in a geographical context, and Section 5 describes the system's most relevant results and experiments. Finally, Section 6 delves into the focal points of this work.

2 Related Works

This work covers two main areas: Web Extraction (more specifically Web Scraping) and Slot Filling. WS can be described as the extraction of useful information from web pages, while SF is defined as the extraction of the values of certain types of attributes for a given entity.

Some of the most popular methods for Web Scraping include Text mining, Text grepping, HTTP programming, DOM parsing and HTML parsing[10, 12]. To solve Natural Language Processing (NLP) tasks, such as Slot Filling (SF), there are also different types of techniques: rule based[4, 11], machine learning based[1] and deep learning based[6, 8] approaches.

A work very similar to our own is *Leopard*. In their work, Speck and Ngonga Ngomo[11] introduce Leopard as a Regular Expressions (RE) based approach to SF. *Leopard* was created in the context of the 2017 SWC KBP with the objective of predicting and validating attributes from a predefined dataset. The theme of that year's competition was organizations' attributes, such as the countries they work in, the phone numbers of its' offices, the date they were founded, etc.

The system starts by collecting the given website URLs from the task's provided data for each organization. With the collected websites, *Leopard* crawls them to store their contents and then crawls its subpages (URL links found in the web page with the same domain) by the order they were found. The number of subpages explored was limited to a fixed number of URLs. For the phone number attribute, *Leopard* searched a web page's contents for hyper-references starting with the keyword "tel" for phone number and chose the phone number that occurred with the highest frequency. This system also made use of a library to parse, format and validate international phone numbers. Finally, in case multiple phone numbers had the same frequency in a web page, the system chose the one that occurred at the first place on a website.

Another attribute Leopard analyzed was the date an organization was founded. To discover the values for this attribute, *Leopard* used regular expressions to choose the smallest 4 digit number in the interval [1900, 2018]. Another method they used was searching for the keyword "founded" in the text of a website, by traversing its HTML markup, to extract the year behind this keyword.

3 Datasets

Throughout this project six datasets were used, which were crucial for this work's development and simultaneously one of this work's biggest challenges, due to the lack of annotated data in this context. Three datasets were used in the *training* phase (OSM Dataset, Official Address Dataset and Yellow Pages Dataset) while the other three datasets were used in the *testing* phase (Annotations Dataset, Truth Base Dataset and OSM Attribute Dataset). The Annotations Dataset was manually built as there were no data sources that complied with this work's requirements, reason why it is described below. The remaining datasets are described thoroughly in the main document.

3.1 Annotations Dataset

This dataset needed to have a similar structure to a web page, besides containing all 3 attributes, since the system's objective is to attribute labels to sentences in HTML web pages. The search for a dataset with these requirements was quite extensive, there were some options available, however, many of them either did not comply with the base requirement (containing all attributes), their information was not 100% reliable or they were not freely available, reason why this dataset was manually annotated.

Thus, the *Annotations Dataset* consisted in several manually annotated documents containing text extracted from HTML web pages (30 documents, one for each entity). Since every classification task requires an annotated dataset, this dataset was created, so that the system's correctness could be evaluated. With this objective in mind, 30 entities were chosen from all available entities. The selection criteria for an entity to be chosen was:

- The entity's online website must contain values for all three attributes (address, opening hours and phone number);
- The entity's website must be successfully extracted through Scrapy (e.g. the website has to be a valid URL for that entity);
- The entity's website needs to have the attributes information in Portuguese.

Only 30 entities were chosen as not many fulfilled the criteria and 30 was a reasonable enough number considering all available entities. After collecting all 30 entities' HTML web pages with Scrapy, they were all manually annotated, which is a time-consuming task. The annotation procedure for the collected web pages consisted in the following manual steps:

- Eliminate all content that is in a language other than Portuguese;
- Assign a value of 0 for all sentences that are of no interest (e.g. "Sinceramente, 0" and "Saboreie uma vida diferente. 0");
- Assign a value of 1 for all sentences that are a valid Portuguese address (e.g. "Praça da Figueira, 12A 1" and "1100-241 Lisboa 1");
- Assign a value of 2 for all sentences that are a valid Portuguese opening hour, more specifically all cases where an hour indication was present, preceded or followed by a week day or week interval reference. (e.g. "SEG - QUI 2" and "09h30 - 20h00 2", but "Segunda fechamos 0" was not annotated as a valid opening hour, since it has a complex meaning and implies an opening hour reference indirectly. On the other hand "Segunda-feira: 14h-17h30 2" was positively annotated as it is a concrete and obvious opening hours reference);

- Assign a value of 3 for all sentences that are a valid phone number, even if not Portuguese (e.g. “+351 218 862 859 3" and “924 775 913 3");

This dataset presents both categorical and numerical data, it also contains 38222 records, where 542 are addresses, 933 opening hours, 558 phone numbers and 36189 instances that are not attribute values, which makes it an imbalanced dataset. Imbalanced since 70% of all entities are negative, while only 30% of the instances are positive (contain any type of attribute). This high percentage of negative instances is to be expected since the average entity's website contains plenty of varied types of information regarding the entity and attributes like address, opening hours and phone number are often displayed in few lines, thus being a mere fraction of it.

This dataset is used for the ML variation of the system, reason why it should be balanced in order to avoid affecting classifiers' which are sensitive to imbalanced classes. Thus, the dataset was analyzed regarding the existence of positive instances (addresses) and negative instances (not addresses). Out of the 542 addresses, only 112 were found to be unique. Since there were only 112 positive unique examples, in order to have a balanced dataset, 112 negative records were randomly selected out of the existing 37680 negative instances (combination of phone number, opening hours and non attribute sentences of the dataset). To randomly select the negative instances, the *random.sample numpy*'s function was used from the *random* module, which returns k (in this case 112) number of random elements from the input array. After shuffling and joining both the positive and negative instances, the now balanced dataset was composed of 224 annotated records for the address attribute.

4 System Architecture

In order to develop this work, several WS and SF techniques were used. Therefore, this work focuses in two types of methods: methods that extract information from the Web and convert it to a tree like structure, in order to traverse each web page's extracted nodes, and methods that recognize relevant attributes of entities in continuous text, based on RE rules, LM and ML techniques. As can be seen in Figure 1, the system's architecture consists of a:

- *Web Scraper*, that extracts content from the Web (e.g. for the entity “ShariSushi Bar", extracts its official website from OSM, goes to its official website *https://www.sharisushibar.pt/*, requests its HTML web page and extracts it to a local machine)
- *Slot Filling Component*, that selects candidate attribute strings based on several methods. These methods are:
  - *Regular Expression* rules, that filter the extracted text and try to match pre-defined strings with it (e.g. for the entity “ShariSushi Bar", its HTML document is traversed and each line is tested for a match with the REs defined for each attribute. The line “3030-193 Coimbra" is an example of a positive match for this entity since it matches the address' REs);
  - *Language Model* algorithms, that first learn a language's vocabulary and then try to recognize it in the extracted text and attribute to each analyzed sentence a probability score describing the likeliness of belonging to the vocabulary the LM recognizes;
  - *Machine Learning* algorithms, that analyze an attribute annotated dataset with labels for each candidate and, based on the labels viewed, decide if they are valid or not (e.g. for the entity “ShariSuhi Bar", according to a Support Vector Machine classifier trained with a balanced dataset, “3030-193 Coimbra" is labeled as an address).

That being said, the proposed system will consider three attributes to be analyzed for each entity: *address*, *phone number* and *opening hours*. These three attributes were deemed relevant for any GIS, as they are the most useful details regarding geographical entities from a user's perspective. One particularity of the system, regarding the mentioned attributes, is that, in the SF component, not all techniques focus on all attributes. More specifically, the RE rules focus on all attributes, while the LMs and ML algorithms focus solely on the address attribute. The reason for this distinction will be clarified later in this section.

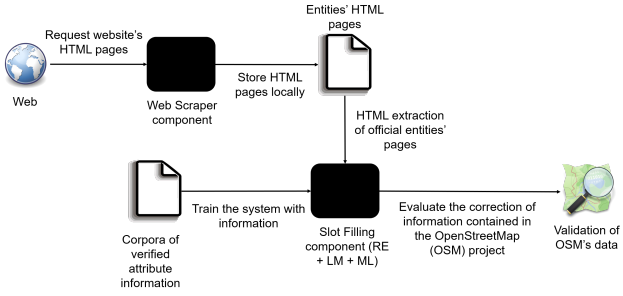

Figure 1. Architecture Representation

In 1, it is possible to see a more concrete representation of the developed system. Firstly, the system interacts with the Web, by extracting from the *OpenStreetMap*[1] project the official web page links for all chosen entities. More concretely, the chosen entities are all Portuguese entities registered in OSM with the *Amenity*[2] key. In order to be chosen, these entities have to contain the selected attributes: address, opening hours and phone number, and have the Amenity tag in OSM. This extraction step results in a list of official websites for all OSM entities with the 3 attributes in question.

[1] https://www.openstreetmap.org/
[2] https://wiki.openstreetmap.org/wiki/Key:amenity

With the collection of official web page links of all entities, our Web Scraper travels the Web in search of each link's HTML web page, as can be seen in 1. The crawler was developed using *Scrapy*[3] and it extracts the available HTML web pages for each website, while respecting all the websites' *robots.txt* file restrictions.

From the collection of HTML pages, the Slot Filling component of our system then extracts all possible candidates for each attribute. This component possesses three types of techniques it can use to analyze the extracted HTML files. The first method is based on RE rules, called the *Rule-Based Extractor*, which is a method that given a text, in this case an HTML web page, uses regular expression rules to extract possible fillers for each attribute. The second method is based on LMs, called the *Language Model Analyzer*, which is a technique that learns a grammar from a large corpus and then. given a string, outputs the probability of the sentence belonging to its grammar. The third method is based on ML algorithms, called the *Machine Learning Classifier*, which is a technique that trains several classifiers with an annotated corpus and then classifies the extracted text to distinguish valid attribute instances from invalid ones. The Slot Filling component's methods are used depending on the attribute to be evaluated, which will be explained further still in this section.

After these two steps are executed, the system finally returns the valid fillers of all attributes for each entity. There can be multiple valid results and, if that is the case, the system returns them all.

4.1 OSM Extraction

The OpenStreetMap data is organized into countries, tags/keys, entities and attributes. To extract the official information from OSM, a free download server called Geofabrik was used. *Geofabrik*[4] is provided free of charge by Geofabrik GmbH and allows users to access data extracts from the OpenStreetMap project in *.osm.pbf* and *.osm.bz2* file formats. The granularity of the data extracts can be selected via country or city of interest and the extracts are updated on a regular basis.

After downloading the *.bzip* file corresponding to Portugal from the Geofabrik website, the file was unzipped and processed. After a first manual analysis, it was clear that the relevant entities for this work, contained in the data extract, belonged to the Amenity tag (a key used in OSM to describe useful and important facilities for visitors and residents like, toilets, restaurants, cafes, telephones, banks, pharmacies, prisons, schools, etc.). Therefore, the downloaded file was processed with *Osmosis*[5], which is a java command-line tool to interact with *.osm* files, to select only entities with the Amenity tag and insert them into a XML file.

However, not all entities belonging to the Amenity tag had all three attributes, so another filtering step had to employed. Thus, the XML file was traversed and entities lacking at least one of the required attributes were discarded. Finally, each remaining entity had their official website, contained in OSM, registered in a *.txt* file. This file was meant to be later used by the web crawler to extract each entity's web page. The methods for filtering and extracting information from OSM are further explained in the main document.

4.2 Web Extraction

After the extraction of all Amenity entities' websites there was a need to restrict its number due to the extremely elevated running time. Thus, to test the system's relevance and accuracy only 30 were chosen, from the existing Amenity entities that had a valid website and all 3 attributes present in the OSM platform.

The Web Extraction phase consists in retrieving from the Web the HTML web pages of the selected OSM entities. To this end, a web scraper was developed to extract an HTML web page given a certain website's URL. After an extensive evaluation of currently available tools for Web Scraping, mentioned briefly in 2 and more thoroughly in the main document, Scrapy was chosen. Scrapy is a scraping framework with the main advantages of respecting *robots.txt* requirements and scheduling and processing requests asynchronously, instead of sequentially executing all requests as most tools do which is time-consuming.

Scrapy starts its crawling process by making requests to the URLs defined in the *start_urls* attribute or through the *start_requests* method. In this work's crawler, the process started with the *start_requests* function, which fetched the OSM's URLs obtained through Osmosis and processed them. This processing step started by excluding any URL with bad file terminations (*.jpg*, *.pdf*, *.png*, *.gif*), as well as URLs that include uninteresting keywords (*tel:*, *mailto:*, *maps*, *youtube*, *facebook*, *javascript*), since URLs referring to images, PDFs, *Youtube* videos or *Facebook* pages, and emails or telephone hyperlinks are not HTML web pages and therefore irrelevant in this work's context. After this filtering step, the Scrapy's URL *request* method is invoked (meaning that the HTML web page is requested to the website), after which the default callback method of Scrapy, *parse*, is called to process the HTML page obtained.

In the *parse* function, our scraper first fetches the HTML content inside the received *response* object and saves it to a text file. After storing the HTML web page, the system checks at which depth level the current URL is. The depth level of a URL represents the level of indirectness a URL has (e.g. a URL1 extracted from OSM has a depth level of 1, while a URL2 extracted from the web page of URL1 will have a depth of 2, and so on). If the current depth level is

[3] https://scrapy.org/
[4] https://download.geofabrik.de/
[5] https://wiki.openstreetmap.org/wiki/Osmosis

above the threshold, predefined as 5, then Scrapy stops the exploration on the current link and goes to the next link in the queue. On another hand, if the current depth level is below the threshold, then the scraper will loop through the web page's sublinks.

Too look for a web page's sublinks, this work took inspiration in Vargiu and Urru's[12] work, which performs a similar task when looking for all ads contained in a web page. However, instead of looking for an ad in an image format (an <img> associated with an <a>), our system looks for subpages' hyperlinks on the current website. Since Scrapy already provides access to XPath selectors and Gunawan et al.[5]'s experiments show that using XPath patterns is a good option to extract information from web pages in terms of running time and memory usage, they were found to be a reasonable approach. The authors' results state that XPath are the slowest approach (compared with REs and DOM parsing) but not by much and that they do not use much memory, like REs (compared to DOM parsing, which uses a lot of memory). For these reasons, XPath expressions were selected as the method to find and select elements of the HTML web pages. One concern, however, with using XPath selectors is that the majority of this work' entities comprise cafes, restaurants, etc. and these entities often do not have a database-backed website, since these types of websites can insert unnecessary complexity in their websites' development. Knowing this, if XPath queries were used the same way as in *Sitescraper*[9] (with very specific queries), our results might quickly become irrelevant if the website's contents changed drastically. For this reason, the XPath expressions selected were relative. They only look for <a> generic tags, instead of looking for specific tags in the elements' tree, precisely to account for this possibility. Kowalkiewicz et al.[7]'s results also support this relative approach to XPath with their system, called *myPortal*.

Instead of XPath, CSS selectors could also be used as they are also available in Scrapy, but since they are equivalent in this work's context and XPath can additionally analyze the contents of an element while CSS can not, we opted for the first. Finally, since the way to define a hyperlink in an HTML web page is to use the <a> tag, in order to discover a page's sublinks, the Scrapy's XPath selectors looked only for the *href* attribute of all existing <a> HTML tags. For these newly obtained links to be later analyzed by the scraper, they had to fulfill the following conditions:

- No bad file terminations (e.g. *.png*, *.pdf*, etc);
- No uninformative keywords (e.g. *Youtube*, *Facebook*, etc);
- The domain must be the same as the origin web page (e.g. *https://www.tripadvisor.com.br/* is found in the website of "ShariSushi Bar" but since they do not share the same domain, the link is rejected);
- The new link can not have been visited before;
- The new link must include a set of selected keywords, if it is not a first depth level sublink (e.g. *contato*, *informacao*, etc). If it is a first depth level and fulfils the remaining conditions, then it is visited.

The selected keywords considered are: *contato*, *contate*, *informacao*, *sobre*, *atendimento*, *home*, *horario*, *index*, etc, as well as other possible variations (e.g. *contacto*, *about*, *informacoes*).

For the sublinks which fulfill all above conditions, the scraper then schedules another request using the same *parse* method as callback. In case, the new sublink is a relative link (link that points to a location in the same website), the scraper tries to append it to the <base> tag of the web page if it exists, otherwise it appends it to the current web page URL, and then schedules the new request. The <base> HTML tag specifies a default URL for all links on a web page.

Throughout the *start_requests* and *parse* phases, there is one very important internal structure that controls the dimension of this work's scraping process. The structure is a dictionary that tracks the depth of link exploration. More specifically, this structure registers how many subpages were visited already and their depth level (e.g. the entity "Shari Sushi Bar", with the URL *https://www.sharisushibar.pt/*, has sublink at depth 1 *https://www.sharisushibar.pt/contato-e-reservas* and a sublink of depth level 2 *https://www.sharisushibar.pt/menu*, that can be reached through the link with depth 1, etc.). This structure was needed to contain the crawling process to a reasonable level, since not all entities' sub-pages have to be visited in order to find all information regarding the attributes. The idea behind the creation of this limit came from Leopard's[11] use of a crawling limit to the number of a website's sub-pages visited, which avoids unnecessary page crawls and decreases the running time of its scraping process.

One challenge of limiting the crawling depth was URL redirection, also called URL forwarding, which is a World Wide Web technique for making a web page available under more than one URL address. When a web browser attempts to open a URL that has been redirected, a page with a different URL is opened. Similarly, domain redirection or domain forwarding is when all pages in a URL domain are redirected to a different domain (e.g. "wikipedia.com" and "wikipedia.net" are automatically redirected to "wikipedia.org"). This becomes problematic when the system needs to keep track of which websites lead to other websites, especially when the redirection leads to a different domain, which happened to several entities in this work. This was another advantage of choosing Scrapy as it solves this issue by storing all the URLs involved in the process (even if the URLs are redirected). Therefore, at the end of each request, both the URL received in the website's response and all the URLs processed since the original request was made are saved in the metadata of the *response* object. This object is then passed to the *parse*

function, which makes it possible to know which URLs lead to the current one. Thus keeping track of the URL crawling sequence, which was necessary in order to limit the crawling depth of the system.

Another structure that was experimented with in this phase was a dictionary that kept a record of all the URL domains visited and how many times they were visited. Its objective was the same as the previous structure. Although the scraping with this structure provided faster results (about 2 minutes), it proved to be too strict of a limit. Too strict in a sense that when used, it returned a maximum of 10 web pages per domain, which, after a manual analysis, was concluded to not be enough to collect all attributes' information. Since there were web pages in the dataset with more than 20 relevant inner links, a small limit by domain quickly became a problem (e.g. 20 web pages of an entity could contain selected keywords but if the *Contacts* web page with all attributes' data was not among the 20 first pages, then it would be skipped). Due to the fact that the system's running time without this structure was still reasonable (around 90 minutes) and provided far better results, the structure was discarded. The times provided here correspond to the crawling process of the 30 entities selected, as mentioned before. For more entities, this time difference becomes quite relevant and then this structure might need to be reconsidered.

Other issues when scraping web pages are the existence of dynamic web pages, which may display different content depending on certain conditions (e.g. the page may change with the time of day, the user that accesses it, etc.). These pages are often controlled by *JavaScript* or other scripting languages, which determine the way the HTML in the received page is parsed and loaded. Not all WS tools can handle this types of scripts and web pages, which is the case of Scrapy. Therefore, web pages of this type will be ignored in this work.

Finally, at the end of this Web Extraction phase, the system had access to a set containing all the HTML information successfully extracted with Scrapy from the Web for the 30 selected entities.

4.3 Slot Filling Component

In this work, the SF task consists of trying to extract information regarding certain attributes, like address, opening hours and phone number, from the official web pages of some selected entities (e.g. restaurants, bars, etc). The opening hours attribute mentioned throughout this work concerns both the opening and closing hours of an entity. Also, the official web page of an entity is considered, in this case, a collection since it can contain multiple candidates for each of these attributes.

4.3.1 Regular Expression Rules.

One method typically used in text classification is handwritten rules. There are many areas where this technique is considered to either be the state-of-the-art solution or part of the state-of-the-art solution. Since handwritten rules are used in several NLP tasks, like web search, word processing to find and replace words, validation of fields in databases (e.g. dates, email address, URLs) and information extraction (e.g. people or object names), they appeared to be a reasonable approach to experiment with, in this work. They were applied mainly through the use of regular expressions, in order to evaluate how well handwritten rules performed in this specific context.

When the scraper finished collecting the HTML web pages of all selected entities, the system then scanned each page in order to extract possible candidates for each attribute (e.g. address, opening hours, phone number). Possible candidates in this context mean possible values the attribute can have for a specific entity (e.g. possible candidates for the attribute address of the entity *Museu Calouste Gulbenkian* are "Av. de Berna 45A, 1067-001 Lisboa" and "Av. António Augusto de Aguiar 31, 1069-413 Lisboa", despite only the first one being the correct entity's address). The rules were not concerned with finding the correct unique slot values, but all possible values for each entity's attribute.

This first SF approach focused on the use of REs to select possible attribute candidates. The reason why REs were chosen as a possible approach to this problem were due to the conclusions reached in Leopard[11] and DeFacto[4]. Both system use REs to find temporal clues and achieve very high results in the SF task of the 2017 SWC competition. Also, in DeFacto, the use of REs specifically helps generalizing search patterns of the fact the system needs to validate. In our case, REs will help when looking for attributes' values (e.g. addresses), which can have many different formats.

The use of REs in this work is done with the *re*[6] Python library. While, in one hand, using REs simplifies the discovery of the fillers, which are possible values of a slot in the SF task, on another, it requires the execution of an extensive lookup to find a good pattern. The desired pattern is a pattern that must be general enough to uncover the correct type of information and yet strict enough to find the exact relevant data for that slot (e.g. a possible and correct candidate for the address attribute of the entity *Museu Calouste Gulbenkian* is "The main entrance is at Avenida de Berna, 45A.", an incorrect candidate would be "10:00 – 18:00" since it is not an address but an opening hour).

To execute this phase, each entity's web pages were traversed and each HTML element or node's text was extracted through the use of the Python library *BeautifulSoup*[7] (used in both Penman et al.[9] and Vargiu and Urru[12]). The construction of a DOM parsing tree from each visited website, to allow for the traversal of that website's markup was based on the Leopard's[11] and SiteScraper's[9] approach, as well as Vargiu and Urru's[12] work. On the latter approach, the

[6] https://docs.python.org/3/library/re.html
[7] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

authors are concerned with extracting image ads so they only focus on the HTML <a> tag, but since this work assumes not an advertisement context but a geographical one, tags such as , <p> and could include relevant information so they are also processed. The selection of this method was also supported by Gunawan et al.[5]'s experiments, that evidenced HTML DOM parsing as the fastest technique (compared to RE and XPath based techniques) to traverse the various components of an HTML web page. After the creation of the DOM tree, each node (each paragraph) was analyzed by the regular expressions and in case it matched it would be registered as a correct value for that attribute. Below, all REs used for each attribute and extra considerations regarding them are explained.

Address

For the address attribute, it is clear that regular expressions are not able to completely identify all possible values as expected since it is a complex attribute with lots of word variations. However, the rules are still able to retrieve the majority of them, which is an indicator of this technique's great performance when the selected rules are adequate.

The rules chosen for the address attribute search for a specific address combination, which means that, the string being evaluated must contain a common starting word for a Portuguese address (e.g. "Rua", "Avenida", etc.), followed by a zip code reference composed by digits (e.g. "2345-678"). The regular expressions to catch these cases are:

```
"\bRUA\b.*[0-9]+|\bR\. .*[0-9]+|\bR\b.*[0-9]+|\b
PRAÇA\b.*[0-9]+|\bP\. .*[0-9]+|\bP\b.*[0-9]+|\b
PRAÇETA\b.*[0-9]+|\bLARGO\b.*[0-9]+|\bLUGAR\b.*
[0-9]+|\bQUINTA\b.*[0-9]+|\bQ\. .*[0-9]+|\bQ\b
.*[0-9]+|\bS[ÍI]TIO\b.*[0-9]+|\bAVENIDA\b.*
[0-9]+|\bAV\. .*[0-9]+|\bAV\b.*[0-9]+"
```

```
"[0-9]{4}-[0-9]{3} "
```

These regular expression patterns were selected after a careful evaluation of the available datasets and taking into consideration general facts regarding Portuguese toponymy. Since the possible composition of addresses in Portuguese is so diverse, it is simpler to look at the start words of an address since they present a smaller set of unique possible words, like "Rua", "Praça", "Avenida", etc. The regular expressions and the phrases being evaluated were all converted to upper case to eliminate casing issues. The reason why these REs were chosen was that despite the enormous variety of words a street name can include in Portuguese, these are the most common formats.

Although these expressions find most cases, they cannot find all possible addresses. For example, *Beco* is also a possible start word in an address, however it is incredibly uncommon and resulted in more incorrect than correct results when first tested, which is why *BECO* was not considered as a possible RE in this system. This makes the system unable to catch instances of these rarer address types, which is one of this method's shortcomings.

Opening Hours

For the opening hours attribute, regular expressions are not able to completely identify them, similarly to the address attribute, but they are quite useful for narrowing down the number of possible opening hour's values.

The selected regular expressions caught instances that fulfilled 3 major restrictions:

- The instance contained a day of the week or a similar reference (e.g. "Segunda"), by matching one of the following expressions:

  ```
  "(2ª|3ª|4ª|5ª|6ª|\bSEG\b|\bTER\b|\bQUA\b|\b
  QUI\b|\bSEX\b|\bSÁB\b|\bSAB\b|\bDOM\b|\bS\b
  |\bT\b|\bQ\b|\bD\b|\bSEGUNDA([ -]{0,1}FEIRA
  {0,1}|)\b|\bTERÇA([ -]{0,1}FEIRA{0,1}|)\b|
  \bQUARTA([ -]{0,1}FEIRA{0,1}|)\b|\bQUINTA
  ([ -]{0,1}FEIRA{0,1}|)\b|\bSEXTA([ -]{0,1}
  FEIRA{0,1}|)\b|\bSÁBADO\b|\bSABADO\b|\b
  DOMINGO\b|\bDIAS [ÚU]TEIS\b)";
  ```

- The instance contained an hour interval (e.g. "12:00-15:00", "14h", etc.), by matching the following expression:

  ```
  "(\b[0-9]{1,2}H{0,1}[ :]{0,3}[0-9]{0,2}H
  {0,1}[ ]*([-/>]|ÀS){1}[ ]*[0-9]{1,2}H{0,1}
  [ :]{0,3}[0-9]{0,2}H{0,1}\b)";
  ```

- The instance could not be an address, according to the REs for the address attribute (e.g. if the instance was "Rua das Flores", it would be rejected as it is considered by the RE to be an address).

Similarly to the address attribute, the opening hours had important context stored in the neighboring lines of the string being evaluated. Therefore, besides the current string, the two previous lines and the next one were added to the opening hour's analysis step. After this change, 4 situations can now lead to a valid opening hour according to the system's RE:

- The previous line contains a day of the week or similar reference, the current line contains an hour interval and neither are an address;
- The second previous line contains a day of the week or similar reference, the current line contains an hour interval and neither are an address;
- The next line contains an hour interval, the current line contains a day of the week or similar reference and neither are an address;
- The current line contains both an hour interval and a day of the week or similar reference and it is not an address.

The chosen regular expressions proved to be surprisingly reasonable when they started to take into account the addresses' REs as well, since the reason why they performed

poorly at first was that several address instances matched with the opening hours rules (e.g. “Quinta da Banda Alegre, 6" is an address but it also matches the opening hours’ REs). So, the assumption that was made in the system was that if an instance matched an address it most certainly was not an opening hour and this improved greatly this attributes’ results.

Phone Number

For the phone number attribute, regular expressions are more than sufficient since they are composed of a limited combination of numbers and symbols. All Portuguese phone numbers are composed of an optional identifier (+351) followed by a combination of 9 digits and a varying number of spaces. Therefore, a regular expression that catches these examples will be quite effective. As such the chosen RE catches 4 different cases:

- When a phone number has an optional identifier and the 9 following digits are broken in groups of 3 (e.g “+351 212 987 654"). The regular expression is:

  ```
  "[+]{0,1}[(]{0,1}\b[0-9]{0,3}\b[)]{0,1}[ ]{0,1}\b[0-9]{3}[ ]{0,1}[0-9]{3}[ ]{0,1}[0-9]{3}\b"
  ```

- When a phone number has an optional identifier and the 9 following digits are broken in groups of 2 and 3 (e.g “+351 21 298 76 54"). The regular expression is:

  ```
  "[+]{0,1}[(]{0,1}\b[0-9]{0,3}\b[)]{0,1}[ ]{0,1}\b[0-9]{2}[ ]{0,1}[0-9]{3}[ ]{0,1}[0-9]{2}[ ]{0,1}[0-9]{2}\b"
  ```

- When a phone number has an optional identifier and the 9 following digits are broken in groups of 1 and 2 (e.g “+351 21 29 87 65 4"). The regular expression is:

  ```
  "[+]{0,1}[(]{0,1}\b[0-9]{0,3}\b[)]{0,1}[ ]{0,1}\b[0-9]{2}[ ]{0,1}[0-9]{2}[ ]{0,1}[0-9]{2}[ ]{0,1}[0-9]{2}[ ]{0,1}[0-9]{1}\b"
  ```

However, these expressions will still catch bad examples like, for example, they will catch any 9 digit sequence without spaces, which matches phone numbers, but can also match other types of information, like driver licenses, id numbers, filenames, HTML elements’ content, etc. (e.g. “...o seu número único de matrícula e pessoa coletiva *513 049 967*...", where *513 049 967* is a driver license number and not a phone number). In this context, these situations are scarce, which is why the RE’s results for the phone number attribute are near excellent.

Rules perform quite well in certain contexts, however, as data can change over time and the effectiveness of the rules depends greatly on the data they are applied to, REs can be considered a fragile or temporary solution. The authors also argue that due to this limitation, most cases of classification in language processing are instead solved via other techniques like language modelling and supervised machine learning.

REs provided very good results for the phone number attribute, but the results were not as good for the address and opening hours, as can be seen in Section 5. Between both attributes, since the opening hours attribute was quite constant in format and no dataset was easily available for it, the address attribute was chosen for further exploration. Besides, having the correct address available in OSM was more crucial as it is a GIS system. For these reasons, all the LM and ML approaches described in this chapter, specifically in the following chapters, focus solely on the address attribute.

4.3.2 Language Models.

The LM implementation used in this work was the NLTK implementation, more specifically the Language Model submodule, called *lm*. First of all the system starts by converting the original corpus to *n*-grams to be learned later by the LM. Thus, the functions *word_tokenize* and *sent_tokeni ze* from NLTK convert the continuous text collected from an addresses only dataset (further explained in the main document) to tokens that are later converted to *n-grams*. Then, the function *padded_everygram_pipeline* from the NLTK’s package *nltk.lm.preprocessing* separates all tokenized sentences into k-grams (in this case, bigrams or trigrams), depending on the chosen LM. The lm module also makes available several smoothing techniques, which allow the language model to deal with unknown words, which is often a problem in language modelling, like *Laplace*, *WittenBellInterpolated* and *KneserNeyInterpolated*. The k-grams obtained are then fitted into one of the smoothed LMs and are ready to test sentences.

For the testing phase, the dataset used is the Annotations Dataset and for each sentence in it, the label is first removed and the remaining sentence is converted to *n*-grams and then analyzed by the *score* function of the selected LM and given a score. To classify the sentence the score given by the LM is compared to a predefined threshold (e.g. 10^{-30}, 10^{-50}, etc.). If the obtained score is below it then it is classified as an address value. The threshold values were manually tested in order to obtain relevant results.

Unknown words become a problem when predicting language model’s probabilities, since no information on them was collected during the training phase. That is why the use of a LM requires a vast training dataset in an effort to decrease the chances of unseen words happening. Of course, this is an extremely difficult factor to ensure when available training data it limited and the word variability for that specific language is high. For the attribute *address* specifically, the chance of finding unknown words in the test set is very high, since the geography context comprises a vast vocabulary. The Portuguese Toponym Vocabulary has nearly 70000 standardized vocables and includes every toponym (general term for a proper name of any geographical feature) corresponding to an administrative division in the participating CPLP countries. Of course the VT includes several different types of toponyms, while in this work we will only

be interested in urbanonyms, which are the proper names of urban elements like streets, squares etc. But, the sheer number of toponyms registered in this platform is already a good indicator of the word variability this type of information has.

Knowing that the linguistic variability for the *address* attribute is high, if the language models were derived solely through the frequency count of the *n*-grams there will be valid sentences with a zero probability when the model finds unknown *n*-grams. This is why some form of smoothing is necessary in order to assign some of the total probability mass to unseen words, so that sentences with unknown *n*-grams might have some probability score attributed to them instead of an immediate 0 (e.g. “Rua das Flores" is a valid address, however in the training set the bigram “Rua das" was never seen while “das Flores" is a high probability bigram. If there is no smoothing, then the LM would return a null score for this sentence, while if there is some form of smoothing, then P(“das"|“Rua") would receive some probability score and the sentence would return a high enough score to be considered an address). There are various LM smoothing techniques, those used in this work are Laplace, Witten-Bell and Kneser-Ney and will be further explained below.

The Smoothing implementations used were the NLTK's *Laplace Smoothing*[8], *Witten-Bell Interpolated Smoothing*[9], and *Kneser Ney Interpolated Smoothing*[10]. All implementations were provided in the *nltk.lm.models* module and both bigram and trigram models were considered to obtain the results with these smoothing techniques.

4.3.3 Machine Learning Techniques.

In order to use Machine Learning approaches, such as Support Vector Machine (SVM), Logistic Regression (LR), Naive Bayes (NB), Decision Tree (DT) and Random Forest (RF), in a textual classification task, the sentences and words must first be converted into vectors of features with the help of vectorizers and encoders, which will be better explained later in Section 5. The vectorizers' implementations considered to translate the textual data into numeric feature vectors were the *Scikit-Learn*'s *CountVectorizer*[11], *HashingVectorizer*[12] and *TfidfVectorizer*[13].

All classifiers start by looking at the data with which they will be trained, the Annotations Dataset. The dataset is then stripped from the annotations so it can be processed by the classifiers. The resulting dataset without annotations is shuffled and partitioned in two groups at random (via the Scikit-Learn function, *train_test_split*), 70% for training and 30% for testing. Then, the training dataset is vectorized and fitted to each classifier. All classifiers are parameter-tuned through the *GridSearchCV* function of Scikit-Learn in order to find the best parameters for each classifier. After the training phase is finished, the classifiers analyze the test set and produce predicted labels for each processed string. These are then compared with the dataset's original annotated labels to calculate metrics that evaluate the classifier's accuracy, as shown in 5.

The implementations used in this work were, for SVMs, the Scikit-Learn's *SVC*[14] from the *sklearn.svm* module; for LRs, the Scikit-Learn's *LogisticRegression*[15] from the *sklearn.linear_model* module; for NBs, the Scikit-Learn's *MultinomialNB*[16] from the *sklearn.naive_bayes* module; for DTs, the Scikit-Learn's *DecisionTreeClassifier*[17] from the *sklearn.tree* module; and for RFs, the Scikit-Learn's *RandomForestClassifier*[18] from the *sklearn.ensemble* module.

5 Evaluation

The metrics used to evaluate and compare the proposed system in this work were Precision, Recall and F-measure.

For the RE method variation, the system obtained an f-measure score of *0,816* for the address, *0,681* for the opening hours and *0,987* for the phone number attribute. For the LM variation, the system obtained f-measures of *0,086* and, when combining the LM model with a previous RE filter, a maximum score of *0,940* was obtained for the address. Finally, for the ML variation, the system obtained for the address attribute a maximum f-measure of *0,928*. The best result however came from using a mixed approach of a ML classifier and RE rules, where the system reached a maximum f-measure of *0,957*.

The final validation phase allowed us to infer that for the *address* and *phone number* attributes, OSM has mostly current and valid information, while for the *opening hours* the results are mainly outdated or incorrect. Also, from the results we reached the conclusion that our system finds the great majority of phone numbers available on a web page, but fails to find a considerable amount of values for the *address* and *opening hours* attributes as they have many distinct

[8] https://www.nltk.org/_modules/nltk/lm/models.html#Laplace
[9] https://www.nltk.org/_modules/nltk/lm/models.html#WittenBellInterpolated
[10] https://www.nltk.org/_modules/nltk/lm/models.html#KneserNeyInterpolated
[11] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer.transform
[12] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html#sklearn.feature_extraction.text.HashingVectorizer
[13] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer
[14] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
[15] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[16] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[17] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[18] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

representations and it is difficult for a system to capture all of them.

6 Conclusion

In this work, we proposed an approach to enrich current location databases, such as OSM, with reliable entity related content. To build such a platform, the system is composed of two modules that contribute to finding all attribute (address, opening hours and phone number) values for an entity. The first module of the system is a Web Scraper, that extracts content from the Web. The returned HTML web pages are then passed to the final module, the Slot Filling Component, that selects candidate attribute strings based on several methods. These methods are: Regular Expression rules, that filter the extracted text and try to match predefined strings with it; Language Model algorithms, that first learn a language's vocabulary and then try to recognize it in the extracted text and attribute to each analyzed sentence a probability score describing the likeliness of belonging to the vocabulary the LM recognizes; and Machine Learning algorithms, that analyze an attribute annotated dataset with labels for each candidate and, based on the labels viewed, decide if they are valid values.

Thus, this work's main contribution is the creation of a system for the extraction and processing of certain geographical entities' information (address, opening hours and phone number) from their official online websites. Its results allowed for an analysis of the data contained in the OSM platform, mainly its currency and correctness. Additionally, this work also produced an annotated dataset (for the address, opening hour and phone number attributes) for 30 OSM entities - Annotations Dataset. In total 38222 sentences were annotated, where 542 contained addresses, 933 opening hours, 558 phone numbers (36189 sentences contained no attribute value whatsoever).

Although this field of study lacks diverse annotated datasets, the results obtained in this work were quite encouraging, which shows the potential a tool like ours could have to improve current location databases.

References

[1] Heike Adel, Benjamin Roth, and Hinrich Schütze. 2016. Comparing convolutional neural networks to traditional models for slot filling. In *2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2016 - Proceedings of the Conference*. 828–838. https://doi.org/10.18653/v1/n16-1097 arXiv:1603.05157

[2] Charu C. Aggarwal. 2018. *Machine Learning for Text*. Springer International Publishing. 493 pages. https://doi.org/10.1007/978-3-319-73531-3

[3] Kang-Tsung Chang. 2019. *Introduction to Geographic Information Systems 9th edition*. McGraw-Hill Education. 461 pages.

[4] Daniel Gerber, Diego Esteves, Jens Lehmann, Lorenz Bühmann, Ricardo Usbeck, Axel Cyrille Ngonga Ngomo, and René Speck. 2015. DeFacto - Temporal and multilingual deep fact validation. *Journal of Web Semantics* 35 (2015), 85–101. https://doi.org/10.1016/j.websem.2015.08.001 arXiv:arXiv:1011.1669v3

[5] Rohmat Gunawan, Alam Rahmatulloh, Irfan Darmawan, and Firman Firdaus. 2019. Comparison of Web Scraping Techniques: Regular Expression, HTML DOM and Xpath. In *Proceedings of the 2018 International Conference on Industrial Enterprise and System Engineering (IcoIESE 2018)*. 283–287. https://doi.org/10.2991/icoiese-18.2019.50

[6] Kun Jing and Jungang Xu. 2019. A Survey on Neural Network Language Models. (2019). arXiv:1906.03591

[7] Marek Kowalkiewicz, Maria E Orlowska, and Tomasz Kaczmarek. 2006. Towards more personalized Web: Extraction and Integration of Dynamic Content from the Web. In *Zhou X., Li J., Shen H.T., Kitsuregawa M., Zhang Y. (eds) Frontiers of WWW Research and Development - APWeb 2006*. Vol. 3841. 668–679.

[8] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. 2020. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems* (2020), 1–21. https://doi.org/10.1109/tnnls.2020.2979670 arXiv:1807.10854

[9] R. Penman, T. Baldwin, and D. Martinez. 2009. Web Scraping Made Simple with SiteScraper. (2009), 1–10.

[10] S.C.M. de S Sirisuriya. 2015. A Comparative Study on Web Scraping. *8th International Research Conference KDU* (2015), 135–140.

[11] René Speck and Axel Cyrille Ngonga Ngomo. 2019. Leopard — A baseline approach to attribute prediction and validation for knowledge graph population. *Journal of Web Semantics* 55 (2019), 102–107. https://doi.org/10.1016/j.websem.2018.12.006

[12] Eloisa Vargiu and Mirko Urru. 2012. Exploiting web scraping in a collaborative filtering- based approach to web advertising. *Artificial Intelligence Research* 2, 1 (2012), 44–54. https://doi.org/10.5430/air.v2n1p44