TÉCNICO LISBOA

Enrichment of Location Databases

Maria Sofia Barros Feio de Almeida

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pável Pereira Calado
Prof.ª Maria Luísa Torres Ribeiro Marques da Silva Coheur

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. Pável Pereira Calado
Member of the Committee: Prof. Nuno João Neves Mamede

January 2021

Acknowledgments

First of all, I would like to thank my Dad for always listening, for giving the best advice and always pointing me in the right direction. Thanks to you I can now work on a field that I am passionate about and I would never have reached this point of my life without you. I am immensely grateful for everything you always do for me and for all the support you give me. You are my inspiration.

To my sister and little brother, thank you for always making me smile and cheering me on. Having you on my side made all the difference and made every moment worthwhile.

To my aunt, uncle, cousins and grandparents, thank you for always being on my corner and for always believing and cheering for me. Your encouragement throughout all these years always pushed me to do and be better.

I would also like to thank my boyfriend for all the long night discussions, for all the revisions, for all the late night singing to cheer me up, for helping me through the toughest parts of life, for always bringing a smile to my face, for always putting up with my sour moods, for getting me out of the house whenever I felt discouraged, for making the silliest jokes just to make me laugh, for coming up with the coolest nicknames, for sharing your ideas and always giving me your honest opinion, for being the greatest and the kindest and for supporting me in everything I do. Thank you for always being there for me.

To my dissertation supervisors, Prof. Pável Calado and Prof. Luísa Coheur, thank you for the many meetings and discussions. Without your knowledge and advice, none of this would be possible.

Last but not least, thank you to all my friends and colleagues for the many long nights of work, for always having my back and for pushing me to be a better person.

Thank you.

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.

Resumo

Atualmente com a grande quantidade de dados disponíveis online existe uma necessidade crescente de integrar diferentes tipos de dados num sistema único, especialmente num contexto geográfico. Ferramentas que integram tanto informação geográfica como dados relacionados com entidades já existem, contudo a informação que contém muitas vezes não é confiável, não está atualizada ou é de acesso restrito. Por isso, neste trabalho, propomos um método para enriquecer bases de dados de localização com informação relacionada com as suas entidades. O nosso trabalho foca-se em dois problemas, extração web e classificação, por isso baseamos a nossa arquitetura em trabalhos destas áreas. O sistema foi construído com informação do projeto *OpenStreetMap* e usa várias técnicas de *web scraping* e *slot filling* de forma a extrair os atributos de uma entidade da Web e a combiná-los numa única ferramenta. Para assegurar que a informação é atual e confiável, o sistema extrai os dados de cada entidade dos seus websites oficiais. Esses dados são então processados pelo sistema, que usa técnicas baseadas em expressões regulares, modelos de língua e aprendizagem automática para classificar cada possível valor de atributo encontrado. A avaliação do sistema é feita através de métricas como Precision, Recall e F-measure. Os nossos resultados mostram que o sistema proposto obtém melhores resultados ao usar expressões regulares ou ao usar uma abordagem mista de expressões regulares e aprendizagem automática, dependendo do atributo.

Palavras Chave

Web Scraping, Slot Filling, Processamento de Língua Natural, Sistemas de Informação Geográfica, OpenStreetMap.

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
CPLP	Comunidade dos Países de Língua Portuguesa (Community of Portuguese Speaking Countries)
CSS	Cascading Style Sheets
CTT	Correios, Telégrafos e Telefones - Correios de Portugal (Portuguese Postal Services)
DOM	Document Object Model
DT	Decision Tree
FN	False Negative
FP	False Positive
GIS	Geographical Information Systems
HEL	HTML Extraction Language
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDF	Inverse Data Frequency
KBP	Knowledge Base Population
KG	Knowledge Graph
LM	Language Model
LR	Logistic Regression
ML	Machine Learning
MLE	Maximum Likelihood Estimation
NER	Named Entity Recognition
NB	Naive Bayes
NLP	Natural Language Processing

NLTK	Natural Language Toolkit
NN	Neural Networks
NNLM	Neural Network Language Models
OSM	OpenStreetMap
PDF	Portable Document Format
RE	Regular Expression
RF	Random Forest
SF	Slot Filling
SVM	Support Vector Machine
SWC	Semantic Web Challenge
TAC	Text Analysis Conference
TF	Term Frequency
TN	True Negative
TP	True Positive
URL	Uniform Resource Locator
VT	Vocabulário Toponímico (Toponymic Wordlist)
WCM	Web Content Mining
WDE	Web Data Extraction
WS	Web Scraping
W4F	World-Wide Web Wrapper Factory
XML	Extensible Markup Language
XPath	XML Path Language
YP	Yellow Pages

1

Introduction

Contents

1.1 Context

With the explosive growth of the World Wide Web, each day, more and more information is accessible online [1]. 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 the extraction of information from the Web, like applications which require the extraction of information regarding its competitors' price list (to stay ahead of them) or user information displayed in a web page (to insert it in an organization's internal database), or the transformation of data presented in a website to excel spreadsheets, among many other applications of this field [2].

One of the user oriented services that could massively benefit from leveraging the Web's knowledge and power are geographical systems. There are many people who find it useful to have a system at hand that can quickly inform them about their current location, what surrounds them and even provide directions to other locations of interest, which just shows how relevant this subject is. This geographical and entity related information is very important for both business and personal reasons as they can sway consumers' decisions (e.g. if a shop has opening hours available while another does not, both in the same proximity range, a customer will be inclined to go to the one she is sure will be open) and help users organize their time and travels better (e.g. if a user needs to arrive at place A at 14h he can use a Geographical Information Systems (GIS) to predict when he will have to leave his house). According to Chang [3], GIS are computer systems that capture, store, query, analyze and display geospatial data. Geospatial data describes both the locations and characteristics of spatial features (e.g. a road can be described by its location and its characteristics: length, name, speed limit, and direction). Due to this field's importance, there is a clear need for fast, simple and trustworthy location tools. GIS are also particularly useful in the tourism sector for informed sustainable development [4] and urban planning, in the natural wildlife protection field for land use planning, and many other fields [3].

In response to this growing need for detailed geographical information, several location search tools appeared such as Google Maps[1], Bing Maps[2], Waze[3], etc. Some of these geographical search tools, like *Google Maps* and *Bing*, provide a user information regarding entities, as well as driving/walking directions to a specific location, while other tools, like *Waze*, specialize in providing a user with driving directions to a specific location (*Waze* specifically does not support the search of a location by an entity's name, only by street). The majority of these tools allow the user to insert an origin (e.g. their current location) and a destination as well as some path preferences. Then, the system will calculate the fastest route to get there according to the inserted restrictions (e.g. the method of traveling, time of arrival,

[1] `https://www.google.pt/maps`
[2] `https://www.bing.com/maps`
[3] `https://www.waze.com/en-GB/livemap`

the time of departure, etc.). Some of these systems have better algorithms than others and take into account more interesting factors, such as traffic jams, road accidents, speed controls, etc.

Nonetheless, one important issue with most of these geographical information systems is that the information they provide regarding geographical entities themselves is rather lacking or inaccurate, due to a varied number of reasons. Some of these reasons are the lack of updates, inserting or updating wrong data, no possibility to insert rotatory or changing schedules (e.g. in a tool, where the schedule of an entity must be fixed weekly, a restaurant that closes at 16h every Friday except for each last Friday of the month, time when it closes at 15h, can only state it closes at 16h on Fridays). It is precisely because of this lack of simplicity and flexibility, that a user would much more value a system that besides geographical information also provides accurate and current detailed information on relevant entities. Besides, the integration of linked data into a single system will avoid unnecessary user queries made to search engines to capture the same information [5], which reduces the user's search time, reason why it is a very desirable characteristic for a geographical system.

In order to merge the Web's knowledge with GIS' so as to improve its contents' relevance, the Web first has to be traversed and its data extracted. To extract information from the Web a diverse set of techniques appeared throughout the years. One of those approached was Web Scraping (WS) which is a set of techniques often used for simple automatic web extraction [6]. Since most websites do not offer the functionality of saving a copy of the data contents displayed in a user's computer, the only option for a user to record those contents is to manually duplicate this information to a local file on his computer. This process can be strenuous and take a considerable amount of time to complete as Mahto and Singh [7] point out in their work. Due to this, the authors believe the use of *Web Scrapers* is expected to increase significantly in the future as these systems open up an entire world of anonymous retrieval of information, without the need for Application Programming Interface (API) (software intermediaries that allow two applications to communicate with each other like *Facebook*, weather apps, etc.). Besides, WS is a technique known to provide interesting results and have multi-domain usage [8]. Some of the many areas of application of Web Scraping are social research [9], competition research and web usage patterns [7, 8], etc.

With the increased focus in processing data from the Web dealing with different data formats becomes an issue. This happens due to a language's property, called variation, that states that there is more than one way of saying/writing the same thing. Thus, in order to extract the full meaning from a web page's continuous text, a system must be able to understand all possible *linguistic variations* (e.g. the addresses *Rua Dona Maria, 13, 2ºDireito* and *R. Dona Maria, 13, 2 D.* are the same and yet a system would have trouble flagging them as identical due to their different formats). This same problem is identified in Gracia and Mena [10], "Because of the Web's open nature, however, online semantics can be defined by different people, for different domains, and can vary significantly in expressiveness,

richness, coverage, and quality, leading to increasing semantic heterogeneity".

Besides, in the Web, typically, information is displayed in an unstructured format since humans can easily process information of this type. However, machines can not process unstructured data as easily. Therefore, the automatic analysis and extraction of text involves a reasonable understanding of natural language by machines, which is an issue that Natural Language Processing (NLP) systems have tried to tackle since their beginning [1].

Despite having evolved greatly in the last decades, NLP techniques still have some limitations. Mainly due to two factors, it is quite difficult for a machine to perceive logical and correct semantic understanding from continuous text, as humans do when faced with words and sentences, and, besides, most languages contain inherent linguistic issues, which are difficult for a system to discern. According to Indurkhya and Damerau [11], often when a system tries to clarify these issues, it can even amplify them or generate additional ones. These issues can either be due to *linguistic ambiguity*, which is a linguistic property concerned with the same word having different meanings (e.g. "get" can mean both to obtain or to understand), or due to *linguistic variability*, which is a property describing how the same thing can be written/said in many different ways (e.g. "st." and "street" have the exact same meaning). Much of the challenge involved in NLP tasks consists of resolving both these issues [11], but the main focus of this work lies in the latter, linguistic variability.

One technique that addresses this difficulty of machines in understanding natural language text is Semantic Slot Filling (SF), which plays an important role in NLP since it helps a system extracting meaning from continuous text [12]. NLP is a field concerned with language (e.g. written data) comprehension and with making decisions by analyzing it. The reason why SF is crucial for NLP is that it is a task concerned with extracting knowledge from an utterance on a word-level by identifying the slot type of a word or phrase (e.g. *Lisboa* can be both a *City* or an *Address* slot type), through the use of methods like Machine Learning (ML). SF can often be confused with Named Entity Recognition (NER), as they are somewhat related techniques with similar objectives [13]. However, Jurafsky and Martin [12] state that while NER focuses solely on finding entities with names (e.g. cities, countries, people, etc.), the SF task is also concerned with finding other types of data (e.g. phone number of a restaurant, address of a museum, age of a person, etc.). This work focuses primarily on discovering attributes of geographical entities as will be explained further in Section 1.2, thus, the techniques used are mainly SF techniques. Currently, the SF task is commonly treated as a sequential labeling task and can be applied in many NLP tasks like intent detection [14–16], spoken language understanding [13, 17–19], etc.

Despite its importance, SF is an area still in need of vast improvement. This becomes evident in the SF track of the Text Analysis Conference (TAC) Knowledge Base Population (KBP). TAC KBP is a competition whose goal is to promote research in the extraction of knowledge from large amounts of information and then filling incomplete elements of a knowledge base, which is precisely an SF problem.

In the 2014 overview of the KBP English track, presented by Surdeanu and Ji [20], the maximum score of the participating systems increased slightly from previous years, even though the test queries were more complex. While this is a very positive improvement, it is clear that SF systems are still far from human performance on this task. According to Surdeanu and Ji's [20] results, the top system in 2014 achieved merely 52% of human performance. In later years, the competition's format suffered some changes, mainly the addition of more languages to the challenge among other variations. This, of course, led the results of the top systems in the 2015 KBP [21] to decrease slightly as multi-lingual support created more challenges. However, the number of participating systems and their results for multi-lingual queries improved the following year [22] even as new requirements were added. This reinforces the relevance of this vast area of research as well as the need for more SF scientific breakthroughs.

In conclusion, in this work we propose a system that analyzes and processes information regarding certain geographical entities, and captures specific slot types from the collected data (mainly the slots *address*, *opening hours* and *phone number*) by leveraging the data power of the Web.

1.2 Motivation

A myriad of geographical apps that provide accurate information regarding its nodes already exist. These tools usually provide the name and official website for each of their nodes/entities. The OpenStreetMap (OSM) project is one of these tools and it distinguishes itself from others because the knowledge it contains is completely free to access and use by any type of user, as there are no copyrights over its data. Besides, OSM is frequently updated by its vast community, unlike some of its competitors [23, 24]. These characteristics establish it as a very complete and resilient system on which many users and contributors rely on.

However, OSM may not always contain the most current data as it is fully dependent on its contributors update schedules and correctness. This is why there is a significant interest in complementing the data it makes available with data accessible online. This data should preferably be information contained in official websites, as it is more trustworthy than data entered by random users [25]. This would turn the OSM project into an even more powerful, free and all-around available information tool, since it could provide both geographical and essential information regarding existing entities. Therefore, to ensure the availability of correct entity related information, in OSM, such as addresses, opening hours and phone numbers, we opted to integrate reliable content taken from the entities' official websites.

Despite this, some argue the issues far outweigh the advantages such an integration of data involves, due to data privacy challenges [5]. Others, argue that since the original driving force of the Web was to potentiate collaboration, it is no wonder it is used as a medium of communication between people and sharing knowledge [26]. It is clear, however, that the integration of the Web's shared knowledge into a

single free system is useful for all types of users since it simplifies the access to valuable and useful information.

This need for automatic integration is the reason why the objective of the system we propose is to allow the access of important and current information easily and aggregated by relevant content to all types of user. This objective aims to suppress the shortcomings of other available geographical tools, that either require entities to manually insert their data (e.g. *Google Maps*), or impose data display restrictions (e.g. *Bing Maps*), or that do not contain all the information needed by a user (e.g. *Waze*), etc.

The existence of these limitations in geographical services and the existence of large amounts of untapped relevant data in the Web are the drivers behind the development of this work. Graham and Shelton [27] reflect on this issue specifically, the impact of the age of big data on geographic services. Their work also delves into how this area might be a useful gateway through which to understand big data as a social phenomenon, since trust in geographical technologies is crucial as it is those very platforms that shape how, where, and why consumers move, purchase, act, and interact. This same trust forces these systems to assume the supply of geographical information in a correct and transparent form as a task of utmost priority, which is precisely the goal of this work.

1.3 Goals and Challenges

Finding official information about an entity through existing geographical location sources is a somewhat limited process, as mentioned before. 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. This is the case of Waze and *Maps.me*[4], which focus solely on geographical information. 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, like Google Maps and Bing Maps. Those services which do not include such restrictions might have other issues, like reliability or currency, which is OSM's[5] case as its information is inserted by random users at random times.

Therefore, the challenge addressed in this master thesis is to develop an automated tool that complements geographical location sources, more specifically in this work, the OSM project, with reliable data taken from the Web. More specifically, the problem in question will involve the application of Slot Filling techniques to data extracted from the Web (through Web Scraping techniques) to obtain current detailed information from a subset of geographical entities. Since the SF task is highly application and domain dependent [28], in this case, a set of selected attributes for the SF task was chosen: addresses,

[4] `https://maps.me/#gsc.tab=0`
[5] `https://www.openstreetmap.org`

opening hours and phone numbers.

The Web has the potential to increase GIS information reach and relevance, however, the use of the Web as corpora is also one of the main challenges of this project, due to the Web's inherent heterogeneity. Despite language redundancy and ambiguity being some of the greatest issues when using the Web as corpora, they will not be covered in this thesis. The reason behind this option lies in this work's specific context (extracting data like addresses and opening hours), which more often presents *linguistic variability* issues than word ambiguities. This is due to the fact that a web page's content can be represented in many different ways as there is no specified global standard for the presentation of data (addresses may contain abbreviations - Avenue or Av. or even av, phone numbers might include the country code, opening hours might be defined as 8 PM or 20h, etc.) [10]. This diversity in data format poses a challenge to the platforms that must interact with it [29], as they must take into account all possible formats. In case they fail to do so, these platforms are faced with the risk of missing relevant data and thus producing inaccurate or even incorrect results.

Another crucial challenge in this work, is finding a unique dataset that contains annotations for the three attributes being considered: address, opening hour and phone number. Some of the most popular methods to extract knowledge from text require annotated datasets. Methods like classification which are a supervised learning technique and thus need to know if each sentence of the text contains for example an address, opening hour, etc. Finding these annotations can become a challenge in some contexts where this type of data is lacking, which was this thesis' case. Since no dataset could be found, the data for this work had to be manually annotated. Manual annotations are time-consuming and prone to errors [12, 30], which may have some impact on the results obtained. Also, since the process was fully manual, the size of the produced dataset was not very large.

In short, the goal of this work is creating a tool for the extraction and processing of geographical entities' related information from the Web, while dealing with the challenges of the Web's format and heterogeneity. The tool resorts to Slot Filling and Web Scraping techniques to extract the information and has the purpose of evaluating and improving the data contained in geographical tools such as the OSM project.

1.4 Contributions

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. This system will then allow for the confirmation or rectification of the data contained in the OpenStreetMap platform, further explained in Section 4.3, thus checking its information's currency and trustworthiness. For the first technique, Section 3.4.1, the system obtained f-measure scores of 0.816 for the address, 0.681 for the opening hours and 0987 for the phone number attribute. For the second

technique, Section 3.4.2, without using a filter, the system obtained f-measures of 0.086 and, with the filter, a maximum score of 0.940, for the address. Finally, for the third technique, Section 3.4.3, the system obtained for the address attribute a maximum f-measure of 0.928. When using a mixed approach, the system reached a maximum f-measure of 0.957.

As consequence of the creation of a system that uses supervised learning techniques there was a need for annotated datasets to train the classifiers, but due to the scarcity of semantic annotations in several domains, this was a challenging task. In this work, as no annotated dataset with all required attributes was found, we also produced a dataset of HTML web pages for 30 OSM entities - Annotations Dataset - where each sentence was annotated, as either containing the attributes. In total 38222 sentences were annotated, where 542 contained addresses, 933 opening hours, 558 phone numbers (36189 sentences contained no attribute value whatsoever). More details regarding the system itself can be found in Chapter 3 and regarding the annotations in Chapter 4.

This work also contributed with a working demo of the system available in `https://gitlab.com/sophiealm/enrichment-of-location-databases-demo`. This demo allows a user to find all attribute values for new entities, by providing the tool their official URLs.

1.5 Organization of the Document

This thesis is organized in 5 chapters, as follows: Chapter 2 explores the state-of-the-art of the subject and its main challenges and advances, Chapter 3 concerns a detailed insight into the solution of this thesis' SF problem, which is applied in a geographical context, and Chapter 4 describes the validation and evaluation method for the developed system. Finally, Chapter 5 delves into the focal points of this work and future work suggestions will be presented.

2

State of the Art

Contents

This master thesis project will cover two main areas of focus: Web Extraction and Slot Filling. Web Extraction or *Web Content Mining* consists of "extracting complex, semantically and visually distinguishable information, such as paragraphs or whole articles from the Web" as stated in Kowalkiewicz et al. [31]. Other authors, like Johnson and Gupta [32], describe this task as the extraction of useful information from web pages. However, from all existing Web Content Mining (WCM) techniques, this project will only be interested in covering techniques corresponding to *Web Data Extraction* and *Web Scraping*, but mainly the latter. After applying these techniques to extract the desired content from Web, the system needs to capture relevant information from it. Thus the system is faced with a *Slot Filling* problem where the system's objective is to "extract the values of certain types of attributes for a given entity from a large collection of source documents", as described by Huang et al. [33].

2.1 Web Content Mining

Before Web Scraping systems, the only way to extract information automatically from an application was through Screen Scraping systems. *Screen Scraping* systems were the precursors of WS systems and, according to Baumgartner et al. [34], consisted of programs that extracted screen formatted information from the display output of another program or application. As a later technique, Web Scraping is a technique that allows a software to extract data from human-readable web pages to computationally process it. Despite having many advantages, the authors also point out some of its main limitations as sometimes this approach lacks the logic necessary to define highly structured output data, as opposed to mere textual chunks. WS systems are also usually unable to collect and integrate data from different related sources, define stable (unchanging) extraction patterns and transform the extracted information into a specific output format. For this reason, extensive research on dealing with dynamically changing web pages was needed, which led to the creation of the Web Data Extraction field.

Web Data Extraction (WDE), a descendant research field of WS, is a field that attempts to build software systems that automatically and repeatedly extract data from web pages with changing content and deliver the extracted data to a database or other applications [34]. WDE is a broad area of research that can be dated back to the early nineties and uses techniques from fields like Information Extraction and Natural Language Processing. The developed system, however, applies mostly WS techniques as they are simpler to implement and fulfill this work's objectives with reasonable enough accuracy as showcased in Chapter 4. Besides, dynamic and changing pages are not a main concern since most of these entities' web pages do not change often or drastically, so a simplified solution was found to be the best approach to the problem.

More information on these fields can be found in, for WCM, in Johnson and Gupta [32]; for WDE, in Baumgartner et al. [34] and Ferrara et al. [35]; and, finally, for WS, in Mahto and Singh [7], Glez-Peña et al. [36] and Sirisuriya [37].

WS's main objective is to locally store a website's web pages. This process is usually performed by crawlers. *Web Crawlers*, also called *Web Spiders*, are internet bots created to continuously explore the World Wide Web. According to Srinivasa and Bhatt [38] and Vargiu and Urru [6], a crawler's purpose is mainly to periodically explore the Web, collect and index its web pages, so that crawler-based search engines can maintain an index of words, phrases and a list of all their occurrences in the web. Further insight into the crawling topic can be found in Srinivasa and Bhatt's [38] work, as well as a brief summary of the challenges WCM faces.

As a variant of web crawlers, Web Scrapers are a type of crawlers aimed at looking for certain types of information or formats in the Web and then extracting and aggregating them into documents or into new web pages [6]. This area can be present in many data contexts, like Bioinformatics [36], advertisements [39], tasks such as online price comparison and weather data monitoring [37], etc. One of the reasons why its techniques are applicable in so many contexts, according to Glez-Peña et al. [36], is that, although they depend greatly on the HTML structure of the web pages, they are often very easy to implement. This is one of the reasons why Web Scrapers are a very good method of extracting information from web pages, since they are easy to implement and extract and process data quickly so that it can be stored in databases or provide interesting conclusions [37]. This explains why a lot of software tools for web scraping already exist, be it libraries for programming languages, frameworks or desktop-based environments [36]. Their biggest challenge being that web scraping deeply depends on the underlying object structure of HTML web pages, which is why any change to its structure will have an impact on the scraper's results.

Some of the most popular methods for Web Scraping include Text mining, Text grepping, HTTP programming, DOM parsing and HTML parsing. *Text mining* relies on linguistics and statistic algorithms to extract relevant data from text files. *Text Grepping* consists of using regular expressions to find information that follows certain patterns. According to Kaur [40], a Regular Expression (RE) is a sequence of characters that define a search pattern and it is often used in NLP tasks like finding text within a larger body of text, validating that a string conforms to a desired format, replacing or inserting text and splitting strings. *HTTP programming* is a protocol used to interact with websites. *Automatic pattern discovery* is a type of approach based on CSS or XPath expressions to recognize patterns and extract information based on them. CSS is a language used for describing the presentation of a document written in a markup language (e.g. HTML), while XPath is an XML selection language that can be used to match elements (it can also be used with HTML files). *DOM parsing* is an algorithm that can parse both XML and HTML pages into tree/object structures, called DOM documents, wherein each tree node is an element of the HTML document. *HTML parsing* is an algorithm that has two functions, creating an editable structure to represent the HTML data and fix invalid HTML portions. More complex approaches include, for example, *Computer vision analysers* [30], which use machine learning and computer vision techniques

to identify and extract information by interpreting web pages visually like a human being would. Malik and Rizvi [41], Vargiu and Urru [6] and Sirisuriya [37] explore these scraping methods and argue the importance of this field of study.

In their work, Gunawan et al. [42], compare three of these WS approaches - Text Grepping, HTML DOM parsing and Automatic pattern discovery (XPath) - in terms of memory and time performance. The authors' results conclude that HTML DOM parsing presents the best average execution time but the worst memory usage percentage, while regular expressions and XPath present a very low memory usage in comparison but slightly higher average times.

2.1.1 HTML Parsing

Many of WS techniques that allow for a simple extraction rely on the structure of the web page itself (e.g. knowledge of HTML structure or DOM tree structure) to scrape data. Kadam and Pakle [2] present a survey of HTML dependent web scrapping techniques, some of which will be later approached, such as *DeLa* [43].

In their work, Vargiu and Urru [6] apply techniques like HTML parsing techniques to a web page, through the Python libraries, *HTMLParser*[1] and *BeautifulSoup*, to gather suitable ads to a given web page. From a collection of initial URLs, their proposed system first looks for each web page's first 10 backlinks (e.g. all pages that link to that one page) by relying on the *Yahoo! Site Explorer*. Yahoo! Site Explorer was a service that allowed users to view information on websites in Yahoo!'s search index, like the full, timely backlink reports of any website. This service is currently integrated into *Bing Webmaster Tools*[2] since 2011. Then, their approach relies on WS techniques to extract banner ads from the collection of web pages found. The reason they use WS methods is due to their need to access HTML tags as object members, access tag attributes in a dictionary-like manner and find tags by matching their names, contents or attributes according to certain criteria. Thus they resort to WS tools such as the Python libraries, HTMLParser, which parses an HTML web page, and BeautifulSoup, which transforms an HTML web pages into tree structures, where each node is an HTML element. From the parsed tree, the system looks for anchor tags <a> in each node and selects those which are images only (have an <img> tag), since the authors exclusively want to create an ads collection. After this step, they apply several policies in order to choose a reduced amount of ads to showcase in the final web page (e.g. randomly select three out of all ads in the web page).

In conclusion, their method is highly dependent on a web page's markup. If there are changed to the HTML structure, the results of this technique may vary greatly. The simplicity of this method is one of its main advantages, however it becomes lacking when the system's objective is more complex than

[1] `https://docs.python.org/3/library/html.parser.html`
[2] `https://www.bing.com/toolbox/webmaster`

finding all ads in a web page. If the task required looking at an HTML element's content this approach would require the combination of other techniques, such as Automatic Pattern Discovery.

2.1.2 Automatic Pattern Discovery (XPath/CSS)

Since document markup is often volatile, each change to the markup of a given website requires a manual update of the scraper [39]. This is taxing from an implementation point of view in the sense that the scraper must be often updated to remain effective. To avoid this issue, tools like *SiteScraper*, which are less dependent on the website's markup became quite popular. SiteScraper, presented in Penman et al. [39], is a system that automatically learns XPath based patterns to identify where a user-defined list of strings, *text chunks*, occurs in a given web page set, *seed documents* (e.g. if the user requires the temperatures' information contained in a website, the text chunks could be temperature examples displayed on the website - (22, 23, 27) - and the seed documents could be some regions' web pages taken from a weather website).

To generate the XPath query, first the user must provide the system with the seed documents and the location of the desired text chunks (that the user wishes to extract) on those same web pages. Then, the system uses the *lxml*[3] and BeautifulSoup Python libraries to parse the HTML page. With the parsed web page, SiteScraper creates, for each seed document, a hashtable (a structure that can map keys to values) where the keys are all strings in that document and the values are those strings' element paths (or the HTML tag sequences indicating where those string can be found in the document, e.g. "/html/body/div"). Next, the system queries each table for the initial *text chunks* by using a string similarity algorithm as a way to find matches in the table entries. This algorithm uses a scoring mechanism based on the output of the Longest Common Subsequence algorithm, by using the Python *difflib*[4] module to find all the matching and non-matching substrings. *difflib* provides classes and functions for comparing text sequences. If the algorithm's score is closer to -1, it shows that two strings are highly unrelated, while if the score is closer to 1, it shows the strings have more matching substrings than non-matching and therefore are highly similar.

After discovering the table entries corresponding to each *text chunk* provided by the user, SiteScraper fetches their corresponding values in the tables, thus obtaining the *text chunks*' element paths in the *seed documents*. With the strings' locations, SiteScraper generates an XPath query based on them. These XPath queries are constructed by aggregating all HTML tags that are traversed in order to reach the user string (e.g. */html/body/span[1]*, if the string is contained in the body's first span of the HTML file). However, to find the desired type of string in any web page with a similar structure, while including all related information (e.g. information of all temperatures on a weather website) and not just the user

[3] `https://lxml.de/`
[4] `https://docs.python.org/3/library/difflib.html`

specific examples (e.g. 22, 23, 27), the system must abstract/generalize the found XPath patterns. An abstraction of an XPath pattern consists of relaxing the specificity of the XPath to increase coverage by defining a bounding box (e.g. go up one node in the HTML's tree hierarchy to include more context). Once SiteScraper has determined all the potential abstractions it must choose the set that matches the maximum number of strings. These generalizations are necessary since the relevant information might be presented dynamically (different formats across different web pages) and, it can happen that all user *text chunks* examples are in the same line but the user does not want all values of that line, which must be accounted for.

The more documents and user strings are provided, the better SiteScraper is at generalizing candidate patterns, as the authors observe. The results obtained were quite good, SiteScraper's evaluation metrics were an average precision and recall of 1.00 and 0.97, respectively, over 700 analyzed web pages. In conclusion, the main advantage of SiteScraper is that if a web page changes its global layout but not its contents (e.g. only changes the address information from the bottom to the top of the web page), then the model can easily be retrained correctly without human intervention thanks to the use of XPath queries [39]. The main concerns in turn are that it can only reliably be applied to invariant data. This assumption comes at a risk, however, since websites can change their structure and content completely in a matter of days, rendering SiteScraper's previous results irrelevant, but as Penman et al. [39] state, "Fortunately database-backed websites that inject their content into a common template are popular, so this restriction is not unreasonable".

Another interesting work that mentions web extraction and integration techniques based in XPath patterns is *myPortal*, introduced by Kowalkiewicz et al. [31]. In their paper, the authors present an overview of a simple solution for the integration of information and content on the Web. Their main objective was to develop a system that enabled non-professional users to easily extract and integrate information taken from the Web, to simplify their day-to-day information management routines.

Initially, *myPortal* works as a Web browser, allowing the user to navigate web pages freely, while storing some information regarding its context (such as cookies, HTTP request headers, HTTP variables, POST and GET data, etc). While the user navigates the web pages, the system provides an option for the user to formulate his information needs in the form of queries, through a point-and-click interface. This means that the user only needs to point, click or draw a bounding box around the relevant content or structures he wants and the system automatically generates a query to the selected element (by recording the element's id or position in the web page's DOM tree).

Then, the system translates the user queries generated into XPath expressions over the document tree and converts the HTML document to XML form. Similarly to SiteScraper [39] the XPath queries are formed by aggregating all HTML tags that are traversed to reach the element selected by the user. In this phase, *myPortal* uses two different approaches. The first, similar to SiteScraper, consists of

defining absolute XPath queries, over the whole DOM tree (making the first element always be the root and the last, the selected element, e.g. */html/body/div*), while the second approach considers relative XPath queries, which span from the user selected reference element instead of the DOM root node (e.g. */div*). By comparing both approaches, Kowalkiewicz et al. [31] verified that relative XPath queries outperformed absolute queries in terms of robustness, since relative XPath expressions are immune to minor document structure changes (like advertisement introduction or content repositioning) that often occur on Web portals, unlike absolute expressions. Evaluating these XPath expressions on the documents returned a part of the document tree or a set of elements.

This approach presents some disadvantages mainly that the patterns it defines can be rendered irrelevant in the presence of a website's drastic changes (just like SiteScraper) and that the queries provided by the user might not always give unique answers, thus the XPath patterns found might also not be unique. However, in this approach, if a web pages' structure changes slightly due to repositioning of content or insertion of advertisements, the XPath patterns remain correct, since relative XPath expressions find elements in the document tree through their names or ids, not by their position.

By analyzing the results of Kowalkiewicz et al. [31] and Penman et al. [39], it is possible to conclude that XPath expressions for WCM are a quite reasonable technique, whose main risk lies in extracting content from web pages with changing structures (e.g. changes in names of classes or element's ids, etc), which can be avoided mostly by using relative expressions. Another interesting alternative to XPath expressions, and simpler perhaps, is CSS, which can be used to select styled elements. CSS selectors are known to be simpler to use, when dealing with types of elements (e.g. "div"), class and tag names, since they are shorter and easier to read. CSS selectors are however, incapable of matching content within an element, so for more complex queries, XPath selectors are still the best approach as they provide a broader set of selection possibilities, which the found literature corroborates.

2.1.3 Wrappers

HTML content-aware web scraping techniques are methods that scrape information from the web by utilizing knowledge of the HTML or DOM tree structure of the web pages. Several techniques, based on Wrappers, like this exist and are explored in Sahuguet and Azavant [44], Wang and Lochovsky [43] and Zhai and Liu [30]. A wrapper is a program that extracts data from a Web site or page and put them in a database, thus converting it into a structured format [30].

Wang and Lochovsky [43] developed a system named *DeLa* or Data Extraction and Label Assignment, which extracts data by querying websites through HTML search forms and assumes that websites are generated through scripts that fetch data stored in a database. This approach uses a data model based on nested types (e.g. in a restaurants website and corresponding database, a *restaurant* is a nested type with several attributes like *address*, *opening hours*, etc). Using this data model, *DeLa* pro-

cesses the HTML web page and creates a regular expression wrapper capable of extracting all nested type instances of a web page.

In order to create the RE wrapper, *DeLa* starts by extracting the HTML web pages through a crawler based on HTML search forms. From that collection of web pages, the system creates a RE wrapper to describe the extracted web pages by inducing REs from their HTML tag structure. They do so since in predefined template websites supported by databases the tag structure might appear repeatedly if the page contains more than one instance of the same data object (e.g. restaurantA and restaurantB both have the same format and are the same type of nested structure, *address*, but their attribute values differ). The repeated HTML tag substrings are identified and from them RE are created to capture them, while following the hierarchical relationships present in the original web page.

Before the REs are induced from the HTML tag structure, an algorithm is used to identify relevant information sections of the website, from the HTML full tag sequence. Only then does it compare each pair of web pages from the same website to discover similar sections or data objects (e.g. in a restaurants website context, *restaurant* objects). This comparison is done by building DOM trees based on nested structures for each pair of relevant web pages and, afterwards, doing a depth-first traverse on both, while comparing them node by node, keeping only those that do not have identical sub-trees at the same depth. This eliminates sections such as repeated headers and footers.

Another wrapper baed on DOM tree parsing is introduced in Sahuguet and Azavant [44], called World-Wide Web Wrapper Factory (W4F) which is a toolkit to generate web wrappers. Any wrapper created with W4F first retrieves the HTML document from the Web according to certain retrieval rules. Then the document is parsed by a HTML parser capable of recovering from badly formatted documents. After creating the DOM tree from the HTML document, extraction rules are applied to it and the extracted information is stored.

More specifically, to retrieve the HTML documents from the Web, W4F uses an extraction wizard, which helps the user select the extraction rules. Each HTML document is provided to W4F through the wizard, which returns the document to the user with invisible annotations (that describe each element's location). Then, when the user points to a certain element of the page it gets highlighted and the extraction rules (annotations) pop up. These extraction rules are expressed in HTML Extraction Language (HEL). HEL is a DOM type of language that interacts with an HTML document as if it was a labeled graph or a parse tree. This language also allows for two types of navigation, one based on a depth-first traversal of the document and another that follows the hierarchy of the document through its tags. The language also offers extraction features based on regular expressions (like match and split). Moreover, the language has been designed to extract complex structures (nested type structures such as Wang and Lochovsky's [43]) from HTML documents and not just isolated pieces.

Although wrapper focused works have interesting conclusions and results, they often make strict

assumptions, like Wang and Lochovsky [43], which consider websites with HTML search forms only, and like Sahuguet and Azavant [44] and Wang and Lochovsky [43], which look mostly for complex (nested) structures in the HTML web page (e.g. information displayed in tables or nested information, where there is an entity with attributes associated to it), that are not always applicable. Despite their strict assumptions, similarly, to other approaches seen in this section, both Sahuguet and Azavant [44] and Wang and Lochovsky [43] rely on converting HTML web pages into a DOM format to simplify the process of traversing web pages to select/extract relevant information. Zhai and Liu [30] suggest a slightly different wrapper method that automatically identifies data records in a page by relying on visual cues to build a DOM tree of the document. In conclusion, DOM parsing is quite a useful technique when a system needs to inspect specific elements of an HTML web page and there are several free to use tools (e.g. BeautifulSoup) that perform this transformation step and provide many other features to select specific content from the DOM tree. These will be further explored in the next section.

2.1.4 Web Scraping Tools

Despite there being plenty of web scraping tools available online, not all of them abide by the Web's restrictions, made explicit on the website's *robots.txt* file. This file exists to communicate with crawlers (ex: search engine crawlers) and inform which pages or files the crawler can or can not request from a website, according to the *Robots Exclusion Standard*. These restrictions exist to avoid overloading a website with requests from scrapers and crawlers and since WS systems' objective is to extract information without harming the website they should always respect their restrictions.

urllib

The *urllib*[5] library is a Python package that aggregates several modules for working with URLs. The modules are *urllib.request* that reads and opens URLs, *urllib.error* that has the exceptions *urllib.request* throws, *urllib.parse* that parses a URL into its components and *urllib.robotparser* that parses a robots.txt file, which means it answers questions about whether or not a particular user agent can fetch a URL on the website that published a certain robots.txt file.

Requests

Requests[6] is an elegant and simple Python library for handling HTTP requests. This library allows a user to easily send HTTP/1.1 requests, it doesn't require the manual addition of query strings to URLs, provides digest/basic authentication for encrypted and non encrypted credential communication, respectively, it supports international domains and URLs, etc. The simplicity however is also one of its disadvantages as it does not provide methods to parse the extracted HTML pages, and it can neither handle purely Javascript based websites or extract non-static content from a web page [45].

[5] https://docs.python.org/3/library/urllib.html
[6] https://requests.readthedocs.io/en/master/

lxml

lxml is a high performance, fast HTML and XML parsing Python library written in *C*. This library is usually used together with Requests that extracts the web pages, while lxml parses their HTML orHTML XML content into a tree representation. Some of its main advantages are that it is a light-weight and extremely fast parser. However, its documentation is not the simplest for beginners and the tool itself has some difficulty processing poorly formulated HTML.

BeautifulSoup

According to Vargiu and Urru [6], *BeautifulSoup*[7] is a Python library for parsing HTML and XML data and uses RE to traverse the tag soup it creates from the original document. It provides simple methods for navigating, searching, and modifying a parse tree, and can even automatically detect the encoding of a document and convert it to Unicode, while creating the parse tree. This feature allows BeautifulSoup to automatically convert incoming documents to Unicode and outgoing documents to *UTF-8*, depending on the user's needs. This library's advantages are that it is very user-friendly, provides a lot of methods to analyze the parsed files, comes with clear and extensive documentation, automatically detects a file's encoding through its metadata and it deals well with poorly formed HTML web pages. However, this library is significantly slower than its counterpart, lxml, which is one of its main downsides.

Scrapy

Scrapy[8] is an open source and collaborative web crawling and scraping framework written in Python that allows for a simple HTML extraction from websites. Essentially, it is a framework for writing web spiders that crawl websites and extract data from them. It has become a very popular library since it is extremely simple to use, possesses extensive documentation available on its functionalities and supports the extraction of specific data from HTML document through the use of XPath and CSS expressions. In a way, these selectors can be considered as parsing, since they are used to travel the parsed content by certain tags. Besides, as Hajba [45] states in his work, perhaps some of its best features are that it can make multiple HTML requests asynchronously in a fast manner (the spider even automatically adjust its crawling speed) and has the possibility of forcing the crawler to always respect *robots.txt* files. Despite its many advantages, Scrapy is not recommended for smaller scraping projects since in those cases it can be slower than other tools, like *Requests*, and its learning curve is steeper than that of other scraping libraries.

Tool Overview

Sirisuriya [37] provide a comparative study of several Web Scraping tools, where it is concluded that most scrapers are often quite generic and mostly designed to perform common, simple tasks, making them less universal than expected. Also, only 2 out of the 13 scrapers reported in the article

[7] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`
[8] `https://scrapy.org/`

are free to use, *WebExtractor360* and Scrapy, the latter being our final choice as the first relied too heavily on regular expressions for the extraction process. This dependence on REs would require a manual analysis of each HTML web page to identify patterns and find relevant elements' identifications to extract, which would counter this thesis' objective of building a simple and automatic scraping tool. On another hand, in his work, Hajba [45] takes a more practical approach to the WS challenge and delves into some of these software's advantages, limitations and how to combine them depending on the scraper's main objectives.

Tools	Performance	Extract web pages	Parse URLs	Parse HTML	Parse XML
urllib	Fast	Yes	Yes	No	No
Requests	Fast	Yes	No	No	No
lxml	Fast	No	No	Yes	Yes
BeautifulSoup	Slow	No	No	Yes	Yes
Scrapy	Fast	Yes	No	Yes	Yes

Table 2.1: Comparison of available tools for Web Scraping

2.2 Slot Filling

SF is the task concerned with extracting values of certain attributes (slots) for a given entity from a vast collection of documents [33]. Three main approaches exist to solve this problem: *Rule-Based* Approaches, which turn to Context Free Grammars and Regular Grammars to associate tags with the components of the input string; *Machine Learning* Approaches, which use collections of data to create models that yield probabilities to the association of tags and specific words. Finally, *Deep Learning* Approaches, which are a case of ML Approaches, except for using NNs as the feature extraction method. Each of these approaches are further described below. A Neural Networks (NN), according to Otter et al. [46], is a collection of connected nodes (artificial neurons), which roughly model the neurons in a biological brain. Each node receives a signal, processes it and signals its neighbouring nodes. The signal in this context is a real number and the output of each artificial neuron is computed by a non-linear function of the sum of its inputs.

2.2.1 Rule-Based Approaches

Some approaches to the SF task often include the use of RE rules as they allow for a fast and simple development. In their work, Speck and Ngonga Ngomo [47] introduce a RE based approach called *Leopard*. Leopard was created in the context of the 2017 Semantic Web Challenge (SWC) KBP with the objective of predicting and validating attributes from the given 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 the *libphonenumber*[9] library, which parses, formats, and validates 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. Then to discover the values for the dates when the organizations were founded, 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.

The techniques used by Leopard to solve the SF task provided useful insights, mainly because Leopard applies quite simple techniques and yet placed second for attribute prediction with an F1-score of 53.42% in that year's SWC. Thus, this not only proves that REs are a simple and flexible approach to solve the SF problem in multiple domains, but also produce reliable results against other state of the art approaches.

Another participant system of SWC 2017 which is quite similar in objectives and techniques is *DeFacto*. In their work, Gerber et al. [48] introduce DeFacto, a framework created to validate knowledge facts in a specific temporal interval by extraction of documents from the Web that confirm them.

The system starts by receiving a user fact in a triplet or textual format (e.g. the fact "Exiles Memorial Center is located in Av. Marginal 7152A" can be represented by a triplet ("Exiles Memorial Center", *is located in*, "Av. Marginal 7152A")). Then, DeFacto retrieves all relevant web pages associated with the fact, by querying a regular search engine. These queries are formulated by verbalizing the input fact using multilingual natural language patterns extracted by a pattern extraction framework, further introduced in Gerber et al. [49]. A drawback of this tool is that it can not detect similar patterns where the relation type is the same but has different names (e.g. in "CompanyA Lisbon's address is AddressA" and "CompanyA Porto's address is AddressB", the relation type is the same, *location's address*, even if written differently) [48]. From the search engine queries' results, the highest ranked web pages are retrieved and then evaluated by DeFacto to check for the fact's presence. Besides checking for the fact, the system also checks for its temporal validity and it does so by looking for temporal clues next to it. Thus, DeFacto uses manually defined regular expressions, more specifically with the RE expression "[1-2][0-9]{3}", to find year references or pairs of references, while using a context window to recognize if the temporal data found is related to the fact in question. This application of REs allowed the system

[9] https://github.com/google/libphonenumber

to better generalize facts in order to both select the most relevant web pages for the user fact and to find specific attribute references like temporal clues.

One other characteristic that benefited DeFacto was its capability of combining evidence from multiple languages. This allowed it to account for the multilingualism of the Web, which is one of Web Extraction's biggest challenges. The importance of this feature is evidenced in DeFacto's results, where the experiment with multiple language queries, instead of only English queries, performed much better. Another interesting feature of DeFacto is its use of token distance to select relevant excerpts of the fact in the extracted web pages. *Token distance* defines the distance between the fact's two entities found in an excerp, which DeFacto limits to a maximum of 10 tokens. This setting inspired a possible technique variation, which is further discussed in Section 5.2.

2.2.2 Language Models

The field of NLP encompasses a variety of topics involving the computational processing and understanding of human languages [46] and one of its main pillars are models such as Language Model (LM). They were first proposed as a form of providing models to compute the probability of a given sentence or sequence of words [50]. Or, according to Chen and Goodman [51], "A language model is usually formulated as a probability distribution *p(s)* over strings *s* that attempts to reflect how frequently a string *s* occurs as a sentence". LMs are often used in speech recognition, machine translation, part-of-speech tagging, parsing, information retrieval, etc. In speech recognition, for example, they are combined with acoustic models to predict the next words in a dialogue. In machine translation tasks, which is the automatic translation from one language to another, LMs are used to evaluate probabilities of the system's outputs so as to improve the translation's fluency in the target language [50].

According to Wu et al. [52], in the 1950s, linguistic structure models were the state-of-the art. In these models linguists built grammar rules' to describe languages and they were often assumed as the combination of probabilities and finite state machines. A finite state machine is a system where particular inputs cause particular changes in state. In these models, a sentence would be run through a finite state machine, where each state was a word and each state transition had a probability (of translating into the next state) associated, thus forming a Markov Language Model. According to Jurafsky and Martin [12], a *Markov Model* assumes that the probability of some future state can be predicted without looking too far into the past, in other words, that future states depend only on the current state.

Later, based on the *Markov Assumption* appeared the *n*-gram model, which is a model approximation that calculates the probability of a word by assuming it depends only on a fixed window of previous words (*n* words) [50]. An *n*-gram is a contiguous sequence of *n* items from a given sample of text. These models calculated probabilities by considering bigrams (looking one word into the past), trigrams (looking two words into the past), and so on until the *n*-gram (looking $n-1$ words into the past). Both the

bigram and trigram models are used in the scope of this thesis. An example of the *n*-gram model is, for *n* = 2, the probability of observing the word *Mendes* knowing it is preceded by *Rua Fernão Lopes* can be approximated to:

P("Rua Fernão Lopes Mendes") = P("Mendes" | "Rua Fernão Lopes") ≈ P("Rua" | "<s>") x P("Fernão" | "Rua") x P("Lopes" | "Fernão") x P("Mendes" | "Lopes")

Each of the model's estimates are obtained through the Maximum Likelihood Estimation (MLE), which is done by getting counts from a corpus and normalizing the counts, so that they lie between 0 and 1. Simply put, the MLE calculates the probability of a specific word appearing after a given sequence of words, based on the number of times that sequence followed by the specific word appeared in the training data, divided by the number of times that given sequence appears in the training data [12, 53].

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{c(w_{i-1})}$$

One of the biggest problems, however, when building language models is data sparsity. *Data sparsity* is the property of not observing all possible word sequences in training, which might result in new strings returning low probabilities in the LM when they are in fact valid instances of the language. This happens because some data in the testing set was never seen in the training phase and to handle these unknown words a simple language model can just assign those *n*-grams a probability of 0 [12]. However, just because a string has never been observed in the training data that does not mean it cannot occur in the test data or that it is not possible in the language [51]. In order to force the LM to assign some probability to unseen events, so that they are not automatically assigned a probability of 0, various LM smoothing techniques appeared. Some of these smoothing techniques are *Laplace*, *Witten-Bell* and *Kneser-Ney* and will be further explained below.

Laplace Smoothing

Laplace Smoothing (also known as Lindstone or add-one smoothing) is a technique used to smooth categorical data. Categorical data is a collection of information that can be divided into groups where there is no intrinsic order between categories.

Typically, a LM calculates a bigram's probability by normalizing each sequence of counts by the unigram count. This ratio is often called a relative frequency and its use as a way to estimate probabilities is an example of MLE [12]. The MLE is therefore a method to calculate the probability distribution over *n*-grams from a corpus based on the following formula (for bigrams):

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{c(w_{i-1})}$$

In order to apply the Laplace smoothing principle which is one of the simplest smoothing techniques, the bigram count has to be increased by one and the count for the unigram needs to be increased by

the number of total word types in the vocabulary, which can be generalized for larger *n*-grams [12]:

$$P_{Laplace}(w_n|w_{n-1}) = \frac{c(w_{n-1}w_n) + 1}{c(w_{n-1}) + V}$$

Thus, assuming that *V* represents the vocabulary, all counts that were zero before the smoothing will then have a very small but not null probability. According to Jurafsky and Martin [12], although this technique does not perform as well as others to be used in modern day *n*-gram models, it provides a useful and simple baseline, and is very useful for text classification.

Laplace smoothing is a very simple approach and often produces reasonable results, however it does assume that the prior probabilities (previous history or context) of all events are equal. Another issue with Lindstone smoothing is that to give a non-zero probability to unseen events it must reduce the probability of all seen events (in comparison with their MLE probability). This can result in a sharp change in the result's probabilities because too much probability mass is being moved to cases where there was a 0 [12]. Other than that, it is a simple and easy to implement solution, that eliminates zero count probabilities due to unknown words.

Witten-Bell Smoothing

Witten-Bell smoothing is a smoothing technique that can be considered an instance of the *Jelinek-Mercer* smoothing, according to Chen and Goodman [51]. According to the Laplace Smoothing, if two different bigrams have never been seen before then they will have exactly the same probability, 1/|**V!**|. However, by intuition, if one bigram has words that usually appear more times than the other bigram's words, it should have a higher probability in the model. To apply this intuition which counters Laplace's assumption, there comes a need to interpolate the mentioned bigram model with a unigram model. The unigram model reflects merely the frequency of each word in the training text and will provide information to the bigram model regarding which bigram's words are more common and therefore which bigram should be more probable. The MLE unigram model formula is:

$$P_{MLE}(w_i) = \frac{c(w_i)}{\sum_{w_i} c(w_i)}$$

To interpolate the bigram and unigram model, the following formula can be used, where $0 < \lambda < 1$:

$$P_{interpolated}(w_i|w_{i-1}) = \lambda P_{MLE}(w_i|w_{i-1}) + (1 - \lambda)P_{MLE}(w_i)$$

The interpolated model combines both the Laplace's assumption and the unigram intuition just mentioned, which is why the bigram with the most common words will be the one with the highest probability according to the LM. In general, it is useful to interpolate higher-order *n*-gram models with lower-order models since if the data is insufficient to estimate a probability in the higher-order model, than more

often that not the lower-order model can provide very useful information. A lower-order model is a model of a lower *n* number of grams (e.g. a trigram model's lower-order model is a bigram model or a unigram model) while a higher-order model is a model with a higher *n* number of grams (e.g. a unigram model's higher-order model is a bigram model, trigram model, etc.) [51]. In particular, according to Jelinek-Mercer the nth-order smoothed model is defined recursively as:

$$P_{interpolated}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P_{MLE}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{interpolated}(w_i|w_{i-n+2}^{i-1})$$

Similarly to the Jelinek-Mercer smoothing approach but with a different $\lambda_{w_{i-n+1}^{i-1}}$ calculation, the *n*th-order smoothed model in the Witten-Bell smoothing is defined similarly as:

$$P_{WB}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P_{MLE}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{WB}(w_i|w_{i-n+2}^{i-1})$$

$\lambda_{w_{i-n+1}^{i-1}}$ is computed by using $N_{1+}(w_{i-n+1}^{i-1}\cdot)$, which corresponds to the number of unique words that follow the history w_{i-n+1}^{i-1} with a count of 1 or more. While $1 - \lambda_{w_{i-n+1}^{i-1}}$ represents the probability of finding a new word (a word not observed after the history w_{i-n+1}^{i-1}, in the training data, occurs after that history). To calculate the probability of finding these novel words, the training data is traversed and the number of times the word following the history w_{i-n+1}^{i-1} differs from the words in all such previous events is counted, $N_{1+}(w_{i-n+1}^{i-1}\cdot)$. With this definition we can approximate $\lambda_{w_{i-n+1}^{i-1}}$ to the following expression, for Witten-Bell smoothing:

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\cdot)}{N_{1+}(w_{i-n+1}^{i-1}\cdot) + \sum_{w_i} c(w_{i-n+1}^{i})}$$

By substitution, P_{WB} becomes:

$$P_{WB}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\cdot) P_{WB}(w_i|w_{i-n+2}^{i-1})}{\sum_{w_i} c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\cdot)}$$

The interpolation formula states that the higher-order model is used with probability $\lambda_{w_{i-n+1}^{i-1}}$ and the lower-order model is used with probability 1 - $\lambda_{w_{i-n+1}^{i-1}}$. If the current *n*-gram occurs in the training data then the higher-order model is used, since the bigger the known context (learned in the training phase) of the *n*-gram in question the better. Otherwise the algorithm backs off to the lower-order model since the present sequence context has not been seen in training as it appears (e.g. "Rua Azul" occurs in the training data so higher model is used, "Rua Amiga" does not occur so lower model is used, respectively bigram and unigram models). Witten-Bell is known as a reasonable and fast smoothing technique, being quite good in tasks like text compression [51], since it proves to build models very efficiently and incrementally.

Kneser Ney Smoothing

Kneser Ney, according to Jurafsky and Martin [12], is a smoothing method primarily used to calculate the probability distribution of *n*-grams in a document based on their histories. It is widely considered the most effective method of smoothing due to its use of *absolute discounting* by subtracting a fixed value from the probability's lower-order terms to omit *n*-grams with lower frequencies.

The intuition behind Kneser Ney can be explained in a simple example. Given the sentence "I cannot read without my reading", the following word is clearly "glasses", however the unigram model might say "Kong" is the most probable candidate as it is more frequent in the training data than "glasses". However if we take into account the probability of both words to appear as the continuation of the word "reading", then it is clear that "glasses" is undoubtedly the word that should have the highest probability. This is what Kneser Ney tried to achieve. [12]

The Interpolated Kneser Ney starts by applying absolute discounting, which consists in subtracting a fixed number *D* from all *n*-gram counts, instead of using a factor λ like in Witten-Bell. This is done in order to move some probability mass from the MLE to the unseen *n*-grams while retaining a probability distribution that sums to 1. The adjusted count of an *n*-gram after the absolute discounting phase, where w_k represents a word in position *k* and c represents the number of times that *n*-gram was counted, is:

$$c_{adjusted}(w_1, ..., w_n) = c(w_1, ..., w_n) - D$$

After allocating some probability mass for the unknown *n*-grams, the technique uses *lower-order models* to do the estimations. The logic behind this step is that even if a higher-order *n*-gram was never seen, a smaller order *n*-gram with the same words might have been seen and thus have positive counts, giving some training data context. Therefore, the creation of a language model, where K is the highest order, implies that it estimates probabilities for all *n*-grams where n $\in$ 1,..., K.

With a *recursive interpolation* that combines the probabilities of all lower-order models to get the probability of an *n*-gram, the following formula can be reached:

$$P(w_n|w_i, ..., w_{n-1}) = \frac{c(w_i, ..., w_n) - D}{\sum_{w' \in L} c(w_i, ..., w')} + \gamma(w_i, ..., w_{n-1}) P(w_n|w_{i+1}, ..., w_{n-1})$$

Where γ is the back-off weight, which is the amount of probability mass left for the next lower-order model:

$$\gamma(w_i, ..., w_{n-1}) = \frac{D \cdot |(w_i, ..., w_{n-1}, w') : c(w_i, ..., w_{n-1}, w') > 0|}{\sum_{w' \in L} c(w_i, ..., w')}$$

If after the probabilities' interpolation, a sequence has any *n*-gram suffix present in the corpus, it will not have a null probability.

Finally, the technique has to take into account the *continuation probability*, since if a word appears

after a small number of contexts then it should be less likely to appear in novel contexts. This is why an estimation of how likely a unigram is to continue a novel context should be computed. This unigram Kneser Ney probability is the number of unique words the unigram follows divided by all bigrams, which can be extended to *n*-grams as long as n is not *K* (the highest *n*-gram order).

$$P_{KN}(w) = \frac{N_{1+}(\bullet w)}{N_{1+}(\bullet\bullet)}$$

Where N_{1+} evokes the number of words that have one or more counts and the $\bullet$ is meant to evoke a free variable that is summed over:

$$N_{1+}(\bullet w_1, ..., w_k) = |(w', w_1, ..., w_k) : c(w', w_1, ..., w_k) > 0|$$

If the previous formulas are combined with the previous interpolation formula then it is true for lower-order models (lower *n*-grams) that:

$$P_{KN}(w_n|w_i, ..., w_{n-1}) = \frac{N_{1+}(\bullet w_i, ..., w_n) - D}{N_{1+}(\bullet w_i, ..., w_{n-1}, \bullet)} + \gamma(w_i, ..., w_{n-1}) P_{KN}(w_n|w_{i+1}, ..., w_{n-1})$$

While for highest *N*-grams the formula to use is the previous interpolation formula as there is no data available on the words' past contexts for them.

Despite the recognition which LMs based in the Markov assumption gathered, some renowned linguistics still rejected this kind of statistical approach to language modelling by saying that languages (more specifically English) could not be described by a finite state machine. Their reasoning lied in that no probabilistic model can adequately differentiate sentences based on gramatical correctness, nor give insight into the basic issues of syntactic structure (e.g. while the sentence "Colorless green ideas sleep furiously" is gramatical, "Furiously sleep ideas green colorless" is not, and a Markov Model is incapable of capturing this gramatical issue and would consider both sentences to be included in the language) [52].

Recent advances in computational power and parallelization allowed for the application of *Deep Learning* techniques, which use NNs, to solve the LM problem by exploiting NN's ability to automatically learn the vector of features which characterize all instances of the data. A NN is a collection of connected nodes (artificial neurons), which roughly model the neurons in a biological brain, according to Otter et al. [46], where a node receives a signal, processes it and signals its neighbouring nodes. A signal in this context is a real number and the output of each artificial neuron is computed by a non-linear function of the sum of its inputs. Neural Network Language Models (NNLM) have the advantage of being able to handle longer contexts, not needing smoothing and having a much higher predictive accuracy than *n*-gram models, for certain training set sizes, according to Almeida and Xexéo [54]. However, there is a cost for the improved performance, NNLM are significantly slower to train and require larger training

datasets to achieve reasonable results [46, 54]. For all the mentioned reasons this approach was not pursued in this work. Besides, for many tasks an *n*-gram language model is still considered the right tool [12]. In conclusion, NNs are indeed considered as the current state-of-the-art in language modeling, however they require large amounts of information to produce reliable and unbiased results. In contexts where only small datasets are available, like this thesis project, *n*-gram models remain the state-of-the-art solution as they can provide reasonable enough results in the presence of smaller datasets. To gain further insight in this field, two surveys on Deep Learning applied to the language modeling context are available in Jing and Xu [50] and Otter et al. [46].

Finally, in order to evaluate LMs, as Jurafsky and Martin [12] elaborate in their work, metrics like entropy and perplexity can be used. *Entropy* is a measure of information that calculates the average level of information inherent in a certain variable's possible outcomes. In the language model context, a LM can be considered a source of information and the entropy can be the average amount of surprise when a new word happens. On the other hand, *Perplexity* is an evaluation metric that weights the amount of randomness in a model, or the average number of possible words that can follow any word. If the perplexity is 3 (per word) then that means the model had a 1/3 chance of correctly guessing (on average) the next word in the text, which makes a low perplexity be desirable. Also, if the perplexity of a model is lower than another it indicates that the first is a better model, according to Wu et al. [52]. However in practice it is not always the case as a model can have a very small perplexity and yet only be able to construct repetitive or incoherent phrases because it has a small vocabulary, which does not make for a good language model.

Despite the reasonable results LMs obtain in certain contexts, they do have some limitations, mainly derived from data sparsity as mentioned previously, since the number of words and valid word combinations is in principle infinite and most language models can only give reliable probability estimates for frequent combinations. That is why, one more technique was experimented, in hopes of better learning the possible values of the address attribute, ML algorithms. In supervised ML techniques, there is an initial dataset where each instance of it is associated to a correct output. Then, the goal of the algorithm is to learn how to map from a new observation to a correct output.

2.2.3 Machine Learning

ML can be defined as the study of computer algorithms that improve through experience, without the need for human intervention. Another common definition of ML, more closely related to this work, is the extraction of useful insights from text through various types of statistical algorithms (also referred to as text mining or machine learning from text) [28,55]. ML algorithms can be divided in two areas, supervised and unsupervised approaches. In supervised approaches, there is training data present, which provides the algorithm guidance, just as a teacher supervises her student towards a specific goal. On the other

hand, as the authors explain, unsupervised approaches receive no guidance whatsoever. Thus, these algorithms aim to partition data into similar groups by analyzing each data instance's characteristics.

One very common supervised algorithm is *Classification*, which Jurafsky and Martin's [12] describe as being at the heart of both human and machine intelligence. This type of technique is present in many tasks such as language modeling. When modeling a language, each word or sentence can be thought of as a class and the task of predicting the next word based on previous words or predicting a new sentence based on previous sentences can be thought of as classification. The goal of classification is to take a single observation, extract some useful features, and thereby classify the observation into one of a set of discrete classes. Thus, after learning from a set of labeled instances, most existing supervised learning systems learn to associate class labels to sets of unlabeled instance.

One issue with classification, however, is that the majority of ML algorithms are incapable of learning from pure textual information [12]. Therefore, in order for ML algorithms to process this type of information, the data has to be first converted to word embeddings, in a transformation step called *text categorization*. *Word Embeddings* are fixed length vector representations of words and can be categorized into two types: prediction based models and count or frequency based models [54].

- Prediction based models: models that use local information, more specifically they leverage the word's context. Thus, modeling the probability of the next word given a sequence of words (as is the case with LMs), which is why these models are generally reminiscent of NNLMs, as they were initially created to be applied in that context.

- Count based models: models that use global information regarding the whole corpus and consider global statistics (e.g. word counts and frequencies in the document).

The *count based models* are very often represented as word-context matrices and typical examples of these models are Count vectorization and Tf-Idf vectorization. Count based models were selected, instead of prediction based models, because they are both the simplest method and are available in the Scikit-Learn library that also provides the necessary ML algorithms for this thesis. Also, unlike their counterpart, predictive models do not require a reasonable amount of training information to provide accurate, reliable results, which is why count models are often preferred.

The *CountVectorizer* class converts a collection of text documents to a matrix of token counts, where the number of features chosen is equal to the vocabulary size found in the data. For example, for the phrases "Rua Dona Conceição" and "Largo Dona Maria", the CountVectorizer matrix would be the following:

The *HashingVectorizer* converts a collection of text documents to a matrix of token occurrence counts, possibly normalized. This vectorizer uses the hashing trick to find the token string name to index map its features with integers. The hashing trick consists in applying a hash function to the fea-

Table 2.2: CountVectorizer Example

Phrase / Feature	Rua	Dona	Conceição	Largo	Maria
Rua Dona Conceição	1	1	1	0	0
Largo Dona Maria I	0	1	0	1	1

tures and using their hash values as indices directly, instead of looking the indices up in an associative array. An associative array is an abstract data type composed of paired keys and values, like maps, dictionaries, etc. Since, the hash function determines which indice is corresponding to each feature, there is no need to maintain a structure with all existing features like in CountVectorizer. This is why this approach is low memory scalable to large datasets since it does not store a vocabulary dictionary in memory. Otherwise, this approach is quite similar to the CountVectorizer, as can be seen in the results, since the indexing is done by features also. The only disadvantage is if there are collisions different words can eventually map to the same feature's index.

The *TfidfVectorizer* converts a collection of raw documents to a matrix of TF-IDF features. The TfidfVectorizer is equivalent to applying a CountVectorizer followed by a *TfidfTransformer*. The later one converts a count matrix into a normalized TF-IDF representation. TF-IDF stands for Term Frequency (TF) and Inverse Data Frequency (IDF).

The Term Frequency corresponds to the number of times a word happens in a document (a document can also be a sentence) divided by the number of words in the document. Each document will have a specific term frequency. This definition can be converted to the formula below, where *i* is the index of the word w_i in the document *j* and where *k* is the number of documents available in the dataset.

$$TF_{i,j} = \frac{w_{i,j}}{\sum_0^k w_{i,j}}$$

The Inverse Data Frequency corresponds to the logarithm of the number of documents divided by the number of documents containing the word *w*. IDF, therefore, gives the weight of rare words in the dataset across all documents. This definition translates into the formula below, where w_t is the word and *t* its index, *N* the number of documents in the dataset and DF_t is the number of documents containing the word *w* exists:

$$IDF_{w_t} = log(\frac{N}{DF_t})$$

This leads to the TF-IDF definition which takes into account both the term frequency and the inverse data frequency of a word and is calculated for each word present in the dataset.

$$TF - IDF_{w_{i,j}} = TF_{i,j} * IDF_{w_i})$$

The advantage TfidfVectorizer has over the CountVectorizer alone is that it takes into account that some words are rare, which allows the classifier to use the relative importance of the words instead of just their occurrences.

In their work, Almeida and Xexéo [54] provide an extensive survey on the main strategies for building representations for words and present comparisons between several existing models for both categories of word embeddings. The authors also mention how there is a cost for the improved performance of NNs, they are significantly slower to train and require larger training datasets to achieve reasonable results [54], reason why they were not explored further.

On the other hand, Jurafsky and Martin [12] and Aggarwal [28] delve further into describing existing classification models such as SVM, LR, NB, DT and RF.

Support Vector Machine

Support Vector Machine (SVM) is a set of supervised learning methods, used for many tasks, such as classification, that determine the best decision boundary between vectors that belong a given group/-category or not. SVMs are, therefore, based on the idea of finding a hyperplane that best divides a dataset into two classes [28]. Support vectors are the data points nearest to that hyperplane, the points of a dataset that, if removed, would alter the position of the dividing hyperplane. This is why they are considered the most critical elements of a data set. A hyperplane can be though of as a line that clearly divides a dataset. The further from the hyperplane the data points are, the more certainty there is that they have been correctly classified. The goal of SVMs is having the data points on the correct side of the hyperplane, as that will be the category they will be classified as, and the furthest away from the hyperplane, to make sure that they are correctly classified. The distance between the hyperplane and the nearest data point from either set is known as the margin.

When there is no clear linear hyperplane, there is a need to represent the mapping of data into a higher dimension, so that instead of having a line dividing the data, there will be a plane separating the different classes' data points. This transformation is known as kerneling. The data will continuously be mapped into higher dimensions until a hyperplane can be formed to separate it. There are several kernel variations that can be used in SVMs like linear kernel, polynomial kernel, radial basis function kernel and sigmoid kernel. The best choice for the kernel type always depends on the input dataset and how it can be separated into dichotomic classes. If the dataset can be divided by a line then the linear kernel is the best option, etc. SVMs are memory efficient since they use a smaller set of training points in the decision function, versatile since many different kernels can be used and work very well for smaller cleaner datasets. SVMs are not suited for larger datasets as the training time can be high and they are less effective on noisier datasets.

Logistic Regression

Logistic Regression (LR) is a discriminative classifier, which means the classifier learns enough

features so as to clearly distinguish both classes. This means the LR does not necessarily learn all features that describe each class, as generative models do. Therefore, LR will assign a high weight to features that directly improve its ability to discriminate between possible classes, making it incapable of generating new examples of a class, according to Jurafsky and Martin [12]. The type of Logistic Regression considered in this work is the Binary LR, where all instances can only be classified as 0 or 1. The function used by Logistic Regression to discriminate between classes is a sigmoid function which produces an *S*-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Where $t = \beta_0 + \beta_1 x$ and when applied to the formula becomes:

$$f(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Where $f(x)$ is the predicted output, β_0 is the bias or intercept term and β_1 is the coefficient for the single input value x. The parameters β_0 and β_1 for each instance must be learned from the training data, which is done by using a loss function, called conditional maximum likelihood estimation, that expresses, for a certain instance, how close the LR output is to the correct output. The intuition for this loss function is to seek values for the coefficients (β values) that minimize the error between the probabilities predicted by the model and the real probabilities of the data. After calculating the classifiers, the LR uses the sigmoid function with the calculated coefficients as a threshold to predict which class a data instance belongs to. This threshold or *decision boundary* helps to differentiate probabilities into positive and negative class and it can be linear (e.g. like for linear regression) or non-linear (e.g. like for logistic regression).

A Logistic Regression classifier generally works better on larger datasets and is quite robust to correlated features [12]. It is also efficient and does not require high computation power, it is easy to implement and used widely. Also, discriminative models often tend to perform better than generative models, like NB.

Naive Bayes

Naive Bayes (NB) is known as a generative type of model since it builds a model of how a class can generate some input data. They also, given an instance (e.g. string), return the most likely class to have generated it. NBs are also part of the family of probability classifiers, which uses the Bayesian theorem [12]. The *Bayes Theorem* describes the probability of an event by taking into account prior knowledge of conditions that might be related to the event in question. This theorem provides a way for calculating the posterior probability $P(A|B)$ from $P(A)$, $P(B)$ and $P(B|A)$:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

In the above formula, if $P(A|B)$ is the probability of a fruit being a banana knowing that it is yellow, then $P(B)$ is the prior probability of the predictor (fruit being yellow), $P(A)$ is the prior probability of the class (fruit is a banana) and $P(B|A)$ is the likelihood (or posterior probability) which is the probability of predictor given class (being yellow knowing it is a banana).

Naive Bayes assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Therefore, Naive Bayes requires a rigid independence assumption between input variables, which is not always true, as in text classifications features can be correlated (since the appearance of a word in a sentence can be triggered by previous words). This can lead to the model's bad performance, in the case where there is some dependence between features/predictors (while as we have seen before LR do not present this same issue).

The algorithm starts by converting the dataset into a frequency table (in this case through a vectorizer, e.g. a frequency vectorizer), then it creates the likelihood table which has the probabilities of all events happening based on the frequency table's values. Finally it uses the Naive Bayes equation to calculate the likelihood (posterior probability) for each class. The outcome of the prediction class will be the highest posterior probability (the highest $P(A|B)$). In this work, the multinomial naive bayes classifier is used since it is a Bayesian classifier that makes a simplifying (naive) assumption about how the features interact. The classifier represents a document as if it were an unordered set of words (thus, ignoring its positions), keeping only their frequency in the document. [12]

The main advantages of this approach are that it is a very simple method to implement, fast to train and works extremely well with small datasets [12]. On the other hand, the disadvantages of the Naive Bayes approach are that if an instance has a feature in the test data set, which was not observed in the training phase, then the model will assign a zero probability to it and will be unable to make a valid prediction. Another limitation is that it assumes independent predictors which in real life is almost never the case.

Decision Trees

Decision Tree (DT) is a predictive modelling approach used in statistics, data mining and machine learning, that uses a decision tree composed of nodes and branches to classify previously observed instances. A DT is built by recursively evaluating all features (e.g. words) and choosing the one that best splits the training data at each moment. More specifically, the algorithm first calculates a metric (e.g. gini index or entropy) for all variables/features and all their thresholds and chooses the one with the lowest value. Then it separates the training data in two based on that threshold, each branch will have one side of the threshold. For the next iteration, the algorithm repeats the first step but for both branches and it does so until the maximum tree depth parameter value or minimum reduction in the error metric

is reached.

To predict values, the algorithm starts at the root node (the first node of the tree), looks at the threshold of the feature evaluated by it, and depending on that value go to the left or right child node. Then, the process is repeated until a leaf node is reached. Finally, the instance to be predicted is classified as belonging to the same class of the leaf node.

DTs are very easy to interpret, fast to construct, tolerant to missing values, do not require large amounts of training data, are immune to the effects of predictor outliers and they produce interpretable models (if the trees are small). DTs inherently perform internal feature selection and are resistant to the inclusion of irrelevant predictor variables. These properties of decision trees are largely the reason that they have emerged as the most popular learning method for data mining [56].

However, DTs are very prone to overfitting (classifier performs much better in the training data than in the testing dataset because they learned from noisy training data and become incapable of generalizing well for the testing dataset) and are a weak learners [56]. A weak learner is a classifier only slightly correlated with the true classification, which means it can label examples better than random guessing, but not much better, which makes it a classifier with relatively bad prediction accuracy. However, Decision Trees can be combined into ensemble models quite powerful like Random Forests.

Random Forests

Random Forest (RF) is an ensemble model composed of several DT classifiers where, instead of simply averaging the results/predictions of each tree, it also randomly samples the training data when it build the trees and considers random features' subsets when splitting nodes. Ensemble modeling is a process where multiple diverse models are created, by for example using different training data sets, to predict an outcome. RF's objective is to average many noisy but approximately unbiased models, and hence reduce the variance, also since trees are notoriously noisy, they benefit greatly from the averaging [57].

Thus, the main idea behind this approach is that a single source can not be trusted and neither can a dataset that provides both valid and invalid information. The solution then passes by providing each tree with random different subsets (sampling) of the training dataset in hopes of cancelling the effects of noisy information in the training phase and then pool the votes of each tree and average them. This is why RFs combine hundreds or thousands of DTs, train them on random samples of the data and split nodes in each tree considering a limited number of features (e.g. if there are 16 features available for a tree at each node, the algorithm will only randomly consider 4). Typically the limit of features is set to the quadratic root of the number of features). Then, the final prediction are the average of each individual trees' prediction.

A model is said to have high variance/flexibility when the learned parameters are extremely dependent on the training data, thus generalizing poorly to new information. While inflexible models are said

to have high bias because they systematically make erroneous assumptions (are biased towards pre-conceived ideas regarding the data). Therefore, a good classifier should keep a reasonable balance between both. RFs employ a number of techniques to reduce variance in predictions (thus reducing the effects of noisy data in the training phase) while maintaining the low variance characteristic of DT, state Hastie et al. [57]. RFs do this primarily by averaging together a number of very weakly correlated (if not completely uncorrelated) trees. According to the authors, When the number of features in the RF is large, but the fraction of relevant ones small, random forests are likely to perform poorly, since at each split the chance that the relevant variables will be selected can be small.

2.3 Knowledge Graphs

Most of the evaluated approaches to the SF problem, which use the Web as corpora, resort to KGs. A Knowledge Graph (KG) is a specific kind of database that collects, organizes, shares, searches and utilizes information to store knowledge in a machine-readable format. KGs usually describe knowledge as multi-relational data represented as triple facts (head entity, relation, tail entity), indicating the relation between two entities [58] (e.g. the Exiles Memorial Center is located in "Av. Marginal 7152A, 2765-192 Estoril", an representation of this relation in a KG would possibly be (Exiles Memorial Center, *is located in*, "Av. Marginal 7152A, 2765-192 Estoril")). Examples of widely known existing KGs are *DBpedia*[10], *Google's Knowledge Graph*[11] and *Freebase*[12], the latter having been recently incorporated into Google's KG. Access to Google's KG can be done through the use of APIs like the *Places API*[13], which is a Google service that returns information about places via HTTP requests.

Knowledge Graphs are considered to be a vast source of structured information, regarding entities and their relations, and are mainly used in tasks such as NER. The largest KGs are usually maintained by several contributors and leverage a gigantic source of knowledge with a dimension of millions of entities/records. However, even if KGs possess enormous amounts of knowledge they can still only cover around 15 to 20% of all information available on the Web [49].

Considering the specific case of DBpedia, this KG aims to collect information from *Wikimedia* projects, such as *Wikipedia*. However these projects often do not include volatile entities (entities with a short life span, change name often, etc.), which are the type of entities this work mainly considers (e.g. schools, hospitals, restaurants, etc.). Therefore, even if a KG like DBpedia includes some more well-known entities (e.g. *Fundação Calouste Gulbenkian*), it can not be used in this work, as most of the relevant entities considered are volatile and thus not contained in it. Unfortunately, like DBpedia most KGs present the same domain problem or, sometimes, even if they do include the relevant entities they do not present

[10] https://wiki.dbpedia.org/
[11] https://developers.google.com/knowledge-graph
[12] https://developers.google.com/freebase
[13] https://developers.google.com/places/web-service/overview

the necessary relations for this problem (e.g. DBpedia has information regarding *Fundação Calouste Gulbenkian*'s location as "Lisbon", but does not have its real address). Besides, most KGs have user restricted access, through fees (e.g. Google's KG charges fees to a user who makes certain amounts or type of queries). For the reasons described above, KGs were not used in this master thesis despite their relevance as data sources in the SF task.

3

System Implementation

Contents

In order to develop this work, several Web Scraping and Slot Filling 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.

3.1 System Architecture

As can be seen in Figure 3.1, 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.

In Figure 3.1, 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 project the official web page links for all chosen entities. More concretely, the chosen entities are all Portuguese entities registered in

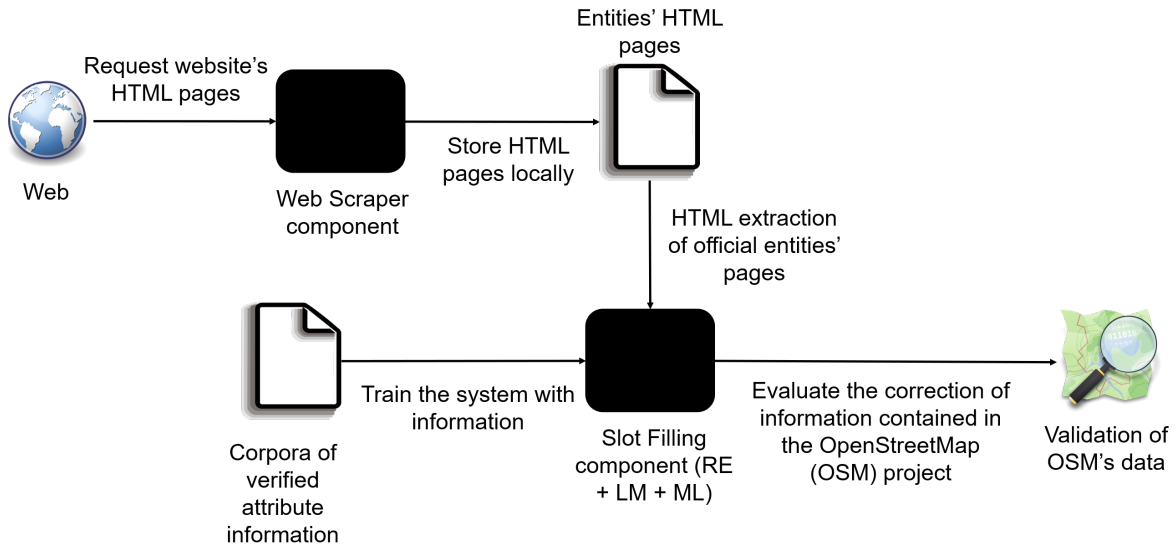

Figure 3.1: Architecture Representation

OpenStreetMap with the *Amenity*[1] 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 the OpenStreetMap project. This extraction step results in a list of official websites for all OSM entities with the 3 attributes in question.

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 Figure 3.1. The crawler was developed using Scrapy 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

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

entity. There can be multiple valid results and, if that is the case, the system returns them all.

3.2 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*[2] 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.

Sub Region	Quick Links		
	.osm.pbf	.shp.zip	.osm.bz2
Africa	[.osm.pbf] (4.2 GB)	✖	[.osm.bz2]
Antarctica	[.osm.pbf] (29.1 MB)	[.shp.zip]	[.osm.bz2]
Asia	[.osm.pbf] (8.9 GB)	✖	[.osm.bz2]
Australia and Oceania	[.osm.pbf] (829 MB)	✖	[.osm.bz2]
Central America	[.osm.pbf] (420 MB)	✖	[.osm.bz2]
Europe	[.osm.pbf] (22.5 GB)	✖	[.osm.bz2]
North America	[.osm.pbf] (9.9 GB)	✖	[.osm.bz2]
South America	[.osm.pbf] (2.3 GB)	✖	[.osm.bz2]

Montenegro	[.osm.pbf] (23.6 MB)	[.shp.zip]	[.osm.bz2]
Netherlands	[.osm.pbf] (1.0 GB)	[.shp.zip]	[.osm.bz2]
Norway	[.osm.pbf] (864 MB)	[.shp.zip]	[.osm.bz2]
Poland	[.osm.pbf] (1.2 GB)	✖	[.osm.bz2]
Portugal	[.osm.pbf] (221 MB)	[.shp.zip]	[.osm.bz2]
Romania	[.osm.pbf] (204 MB)	[.shp.zip]	[.osm.bz2]
Russian Federation	[.osm.pbf] (2.7 GB)	✖	[.osm.bz2]
Serbia	[.osm.pbf] (103 MB)	[.shp.zip]	[.osm.bz2]
Slovakia	[.osm.pbf] (200 MB)	[.shp.zip]	[.osm.bz2]
Slovenia	[.osm.pbf] (240 MB)	[.shp.zip]	[.osm.bz2]
Spain	[.osm.pbf] (878 MB)	[.shp.zip]	[.osm.bz2]

Figure 3.2: Screenshots of Geofabrik Download Server web page (by Continent and Country (Europe))

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*[3], 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 OpenStreetMap project, 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 Section 4.1.1.

3.3 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

[2] `https://download.geofabrik.de/`
[3] `https://wiki.openstreetmap.org/wiki/Osmosis`

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 in Chapter 2, 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 OpenStreetMap'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 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 [6] 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. [42]'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 regular expressionss (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 thesis' 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* [39] (with very specific queries), this thesis' 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. [31]'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 thesis' 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 [47] 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.

3.4 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.

3.4.1 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 [12]. 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) [40], 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 Slot Filling approach focused on the use of regular expressions 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 [47] and DeFacto [48]. Both system use regular expressions 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 this thesis case, REs will help when looking for attributes' values (e.g. addresses), which can have many different formats.

The use of Regular Expressions in this work is done with the *re*[4] 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 (used in both Penman et al. [39] and Vargiu and Urru [6]). 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 [47] and SiteScraper's [39] approach, as well as Vargiu and Urru's [6] work. On the latter approach, the authors are concerned with extracting image ads so they only focus on the HTML <a> tag, but since this thesis' 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. [42]'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 regular expressions 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]+|\bPRAÇA\b.*[0-9]+|\bP\. .*[0-9]+|\bP\b
```

[4] https://docs.python.org/3/library/re.html

```
.*[0-9]+|\bPRAÇETA\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 regular expressions 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|\bQUI\b|\bSEX\b|\bSÁB\b|\bSAB\b|\bDOM\b|\b
  S\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|\bDOMINGO\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 regular expressions 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 regular expression:

- 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' regular expressions 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 regular expression 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.

As we have just seen and according to Jurafsky and Martin [12], rules perform quite well in certain contexts, including this work's. 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 Chapter 4. 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.

3.4.2 Language Models

The Language Model implementation used in this master thesis was the NLTK implementation, more specifically the Language Model submodule, called *lm*[5]. 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_tokenize* from NLTK convert the continuous text collected from the datasets, Official Address Dataset and Yellow Pages Dataset, to tokens that are later converted to *n-grams*. Then, the function *padded_everygram_pipeline* from the NLTK's package *nltk.lm.preprocessing*[6] separates all tokenized

[5] https://www.nltk.org/api/nltk.lm.html
[6] https://www.nltk.org/_modules/nltk/lm/preprocessing.html

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 language models 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 language model 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. According to Salgueiro et al. [59], the Portuguese Vocabulário Toponímico (Toponymic Wordlist) (VT) 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 language model 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*[7]; *Witten-Bell Interpolated Smoothing*[8]; and *Kneser Ney Interpolated Smoothing*[9]. 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.

3.4.3 Machine Learning Techniques

In order to use Machine Learning approaches, such as SVM, LR, NB, DT and 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 Chapter 4. The vectorizers' implementations considered to translate the textual data into numeric feature vectors were the *Scikit-Learn*'s *CountVectorizer*[10], *HashingVectorizer*[11] and *TfidfVectorizer*[12]. The encoder's implementation considered to binarize data (set feature values to 0 or 1) according to a threshold for the NB classifier was the Scikit-Learn's *Binarizer*[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*[14]), 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*[15] 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 Chapter 4.

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

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

DecisionTreeClassifier[19] from the *sklearn.tree* module; and for RFs, the Scikit-Learn's *RandomForestClassifier*[20] from the *sklearn.ensemble* module.

[19] `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`
[20] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

4

Evaluation

Contents

Gathering useful data regarding geographical entities is a time-consuming task since these types of reliable and related information often come at a commercial price and are rarely gathered in one single platform. Thus, the importance of this work and its results. Some aspects regarding the experimental part of this project will be explained in this section like what datasets were used, what classifiers/models and their configurations and finally what metrics were used to evaluate them.

4.1 Experimental Setup

Before describing anything else, there is a need to characterize the textual datasets used in this work. Therefore, this section will begin by analyzing, for all datasets, the type of data, the average number of words per sentence, the minimum and maximum number of words per sentence, the number of sentences, if it is a balanced dataset or not, etc.

A *Balanced Dataset* is a dataset where the number of positive instances is approximately the same as the number of negative instances, whereas in an *Imbalanced Dataset* instances are divided into two distinct sets, majority instances which are the most frequent and minority instances which are the less frequent. Knowing this, there are two clear types of classification. The type that is applied to balanced datasets, called complete classification, and the type that deals with imbalanced ones, called partial classification [60]. This distinction will be applied to all experiments in this work.

4.1.1 Datasets

Throughout this project six datasets were used, which were crucial for this work's development and one of its biggest challenges, due to the lack of annotated data in this specific 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).

4.1.1.A OSM Dataset

The *OSM Dataset* was taken from the OpenStreetMap project through Osmosis, which is a command line Java application for processing OSM data. The extracted dataset consisted on a list of all official websites of entities associated with the key Amenity that had all three attributes registered in OSM: any kind of address, opening hours and phone number.

Amenity is a key used in OSM to describe useful and important facilities for visitors and residents. Facilities such as, toilets, restaurants, cafes, telephones, banks, pharmacies, prisons, schools, etc. The

OSM has several tags to select entities from. The reason behind the choice of the Amenity key was that it comprises entities of relevance to a common geographical app user. A user is highly likely to search for attributes like opening hours and phone number, of an Amenity entity, since these entities are public fundamental facilities.

This dataset was collected to allow for the extraction of the entities' HTML web pages and the entities contained in OSM with the Amenity tag were around 390, from which, later, 30 of those websites/entities were chosen for HTML web extraction and annotation. This dataset is clearly an imbalanced dataset as it only provided positive entities (entities that are a relevant website).

4.1.1.B Official Address Dataset

The *Official Address Dataset* was collected thanks to several Portuguese municipalities that each provided a table of addresses. These tables were later processed and aggregated into a single dataset in order to train the system to recognize Portuguese addresses. Taking into account the multiple sources where the information came from, it was expected that the received files had different formats, file encodings and representations, which proved to be true.

The received files had either an *UTF-8* or *ISO8859-1/Latin-1* (Portuguese) encoding. During the aggregation process each line was analyzed, if the encoding transformation was not possible that line was ignored, otherwise it was added to the unified dataset.

In total, the municipalities provided around 45 files with a varying number of addresses, between 8 and 1400. Aggregating them into one single file created a unified dataset with different formats and about 8500 address records in total with an average word length of 3.8, a minimum of 1 (e.g. "*EN233-3*") and a maximum of 11 (e.g. "*Rua Frei António Espírito Santo [1º Bispo de Angola]*") word length. The data type of this data is categorical and the dataset is imbalanced as it only has positive instances (sentences that are addresses). This is not a problem in this case since this dataset was mainly used to train the language models and LMs are unaffected by imbalanced datasets as they only *learn* from positive instances. This dataset was also used for an experiment involving classifiers, Section 4.2.4, and there the dataset was assumed as part of an unlabeled dataset.

4.1.1.C Yellow Pages Dataset

The *Yellow Pages Dataset* consisted of a list of addresses and was collected to train the LM to recognize valid addresses. The information was taken from the Portuguese Yellow Pages (YP) website[1], through the use of web scraping and regular expression techniques to extract the addresses of relevant

[1] `https://www.pai.pt/`

entities in it. This data was extracted with the help of Scrapy and contains around 25000 entities. The Yellow Pages website of a specific country usually has business listings, phone numbers, maps, email addresses and websites for local businesses. Therefore, the entities included in this dataset are all public services, similarly to those encountered in OSM with the Amenity key (e.g. cafes, bars, restaurants, discos, butchers, etc.). This dataset includes solely the attribute address, despite all relevant attributes being available in the YP's website.

In total, the Yellow Pages provided around 51390 sentences, each a part of or an entire address. These records have an average word length of 4.2, a minimum of 1 and a maximum of 31 word length. The data type of this information is categorical and the dataset is imbalanced as it only has positive instances (sentences are all addresses).

4.1.1.D Annotations Dataset

This dataset in particular 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, mainly KGs or independent datasets, provided by local municipalities or public companies like CTT. 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 the OSM Dataset. 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 [30]. 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[2], 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.

4.1.1.E OSM Attributes Dataset

[2] https://docs.python.org/3/library/random.html

The *OSM Attributes Dataset* resulted from an extraction of the OpenStreetMap project's Amenity set with the help of Osmosis. The dataset was captured simultaneously with OSM Dataset and consisted on a list of values for all three attributes (address, opening hours and phone number) belonging to the 30 selected entities. Its objective is to allow for the comparison between the information contained in OSM regarding the relevant entities and the information found by the system through the entities' official web pages. With this dataset the system will check how trustworthy and current is the data contained in already existing geographical tools with entity related information, such as the OpenStreetMap project.

It has to be taken into account in this comparison that the system's classifier can return no results which means that all input data (candidate fillers for a slot) were considered to be wrong for the attribute in question.

4.1.1.F Truth Base Dataset

The *Truth Base Dataset* was manually collected from the entities' official web pages and consists on a list of possible values for all three attributes (address, opening hours and phone number) by entity. This dataset was collected to allow for the comparison between the entities' real and current online information and the information found by the system through the entities' HTML web pages. This was done since our system must be evaluated in regards to its correctness, to conclude how accurately its results match the entities' real values.

Therefore, this dataset contains 30 entities' information, which means that for each entity one or more values are present for each attribute (phone numbers, addresses and opening hours). All values were transcribed directly from the original entity's website and their original format was preserved, so that whatever values are found by the system match the true values of the attributes registered in this dataset (e.g. if an entity contains in its website two addresses, where one is its official address and the other is another entity's address, both will be present in this dataset).

In order to evaluate the system in the many experiments explored in this work, measures were applied to deduce the system's accuracy and compare it with other available systems. In the section below, the metrics and model configurations used are further explained.

4.1.2 Models' Configurations

The language models used in the following experiments are from NLTK[3]. For all three language models with different smoothing techniques, both the bigram and the trigram models were considered. In order to account for unknown words, the language models used were smoothed with the *Laplace*, *Witten-Bell* and the *Kneser-Ney* smoothing techniques.

[3] `https://www.nltk.org/`

The classifiers used in the following experiments were the NB, SVM, LR, DT and RF classifiers. Before the classifiers could be applied, the data needed to be converted into vectors of numbers, since all available data was textual or categorical data which classifiers can not process. To do this, three Scikit-Learn classes that extract features from text were used: *CountVectorizer*, *HashingVectorizer* and *TfidfVectorizer*.

After converting the input text to numeric vectors that the classifiers can process, the classifiers were parameterized. In order to test several different parameters for each classifier, the GridSearchCV class from Scikit-Learn was used, since it does an exhaustive search for an estimator over some previously specified parameters.

For the *Support Vector Machine* classifier, the tested parameters were the *kernel* ("linear", "rbf", "sigmoid"), the *gamma* ("scale", "auto") and the *C* (1, 10, 100). For the data vectorized with the CountVectorizer the best parameters for the SVM were a *C* of 1, a scale *gamma* and a linear *kernel*. For the data vectorized with both HashingVectorizer and TfidfVectorizer, the best parameters were: a *C* of 10, a scale *gamma* and a rbf *kernel*.

For the *Logistic Regression* classifier, the tested parameters was the *C* (1, 10, 100). For all vectorizers, the best parameters were a *C* of 10.

For the *Decision Tree* classifier, the tested parameters were the *criterion* ("gini", "entropy"), the *splitter* ("best", "random") and *max_features* ("auto", "sqrt", "log2", "None"). For all vectorizers the best parameters for the Decision Tree were the gini criterion, the best splitter and no maximum number of features.

For the *Random Forest* classifier, the tested parameters were the *criterion* ("gini", "entropy") and the *n_estimators* (10, 100). For the data vectorized with the HashingVectorizer and TfidfVectorizer the best parameters for the Random Forest were the gini criterion and 100 estimators. For the data vectorized with CountVectorizer, the best parameters for the Random Forest were the entropy criterion with 100 estimators.

For the *Naive Bayes* classifier, no parameters were tested as there were no relevant parameters to be applied other than the default.

4.1.3 Evaluation Metrics

For a certain attribute (e.g. address):

- A True Positive (TP) is a textual line that contains an address value (which means it is annotated with the value 1) and is flagged by the system as an address;
- A False Positive (FP) is a textual line that does not contain an address and is, however, flagged by the system as an address;

- A True Negative (TN) is a textual line that does not contain an address and is not flagged by the system as an address;
- A False Negative (FN) is a textual line that does contain an address but is not flagged by the system as an address;

The metrics to be used are:

4.1.3.A Precision

Fraction of correctly extracted instances from the total of extracted instances. Informally speaking, precision is the number of correct values out of all extracted values for a certain attribute.

$$Precision = \frac{number\ of\ correctly\ extracted\ instances}{number\ of\ extracted\ instances} = \frac{TP}{TP + FP}$$

For example, for the attribute phone number:

Table 4.1: Precision Example

Entity	Phone Number Extracted	Official Phone Number
Fundação Calouste Gulbenkian	+351 217 823 000	+351 217 823 000
Casa das Histórias Paula Rego	9h - 16h	+351 214 826 970
Farol Museu de Santa Marta	2750-800 Cascais	+351 214 815 328

As can be seen in Table 4.1, for Casa das Histórias Paula Rego and Farol Museu de Santa Marta the phone number extracted was incorrect, but for Fundação Calouste Gulbenkian this attribute was correct. Therefore the precision, in this case, will be 1/3.

4.1.3.B Recall

Fraction of correctly extracted instances from the total of available instances. Informally speaking, recall is the number of correctly found values out of all existing values for a certain attribute.

$$Recall = \frac{number\ of\ correctly\ extracted\ instances}{number\ of\ instances} = \frac{TP}{TP + FN}$$

For example, for the attribute phone number:

Table 4.2: Recall Example

Entity	Phone Number Extracted	Official Phone Number
Fundação Calouste Gulbenkian		+351 217 823 000
Casa das Histórias Paula Rego	+351 214 826 970	+351 214 826 970
Farol Museu de Santa Marta	+351 214 815 328	+351 214 815 328

As can be seen in Table 4.2, for Fundação Calouste Gulbenkian no phone number could be found by the system despite there being at least one in the web page (for Fundação Calouste Gulbenkian, the system classified incorrectly *+351 217 823 000* as not being a phone number). On the other hand, for Farol Museu de Santa Marta and Casa das Histórias Paula Rego this attribute was correct. Therefore the recall, in this case, will be 2/3.

4.1.3.C F-measure

F-measure or F-score is the harmonic mean of both precision and recall. In one hand, a $\beta < 1$ lends more weight to precision, on the other hand, a $\beta > 1$ lends more weight to recall. The general formula for a non-negative β is:

$$F - measure_{\beta} = (\beta^2 + 1) * \frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$$

In order for the F-measure to be considered as acceptable there has to be a balance between both precision and recall, so a $\beta = 1$ provides a balance between both metrics. The formula for $\beta = 1$ is:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

4.1.3.D Macro Average

Macro Averaging computes the average of the precision, recall and f-score of the system on different sets. Informally speaking, macro averaging computes the performance of each class and then the average over the classes for a certain attribute. The macro precision average formula of the system, knowing there are *N* classes, would be:

$$Macro\ Precision\ Average = \frac{precision\ scores\ for\ all\ classes}{number\ of\ classes} = \sum_{n=0}^{N} \frac{(\frac{TP}{TP+FN})_n}{n}$$

4.1.3.E Micro Average

Micro Averaging collects the decisions for all classes and then computes precision, recall and f-score with those values. Informally speaking, micro averaging sums up the individual true positives, false positives, and false negatives of the system for different sets and then applies them to get statistics. The micro precision average formula of the system, knowing there are *N* classes, would be:

$$Micro\ Precision\ Average = \frac{number\ of\ correctly\ extracted\ instances\ for\ all\ classes}{number\ of\ extracted\ instances\ for\ all\ classes} =$$

$$\frac{\sum_{n=0}^{N} TP_n}{\sum_{n=0}^{N} (TP + FN)_n}$$

These metrics provide estimates of the accuracy of the system and were applied to the system's results. Later, the results were manually analyzed, since some fillers might be reported incorrect by the system, although they are correct. This can happen since some fillers can have the same meaning even if presented in different formats. The non-uniformity of the data being handled is the root of this thesis' problem and this final check aims to eliminate it. Fundamentally, the validation process for the system is composed of 2 types of validation, that provide a general overview of the correctness and validity of the system's score:

- Annotation Validation: This validation uses manual annotations to check the correctness of the system's results;
- Final Validations: OSM Validation - manual check to verify if the system values match the OSM values for each entity; Truth Validation - automatic validation to check if the system values match the real values for each entity contained in its website.

The first validation's objective is to analyze how well the system identifies possible attribute values (if they are valid or not). The second validation aims to compare the system's results with other available tools' results, more specifically OSM, to better understand how this system fares against its competitors. The third and final type of validation provides insight on the correctness of the obtained results relative to the current entity's online data.

4.2 Annotation Validation

The annotation process was explained both in Annotations Dataset and in Chapter 3 and it consists of assigning to each sentence a label 1 to addresses, a 2 to opening hours and a 3 to phone numbers, otherwise labeling it with a 0. After the system processes all data, the results are then compared with the annotations, to analyze the system's success at attributing labels to sentences.

4.2.1 Rules Evaluation

The first technique used by the Slot Filling component of the system was *Regular Expression Rules*, described in Section 3.4.1. In this phase, the system extracted values for three attributes: address, opening hours and phone number. Some experiments were conducted on each specific attribute's regular expressions. The initial RE considered were:

- Address: Record must contain an address starter marker (e.g. *Rua*, *R.*, *R*, *Praça*, *P.*, *P*, *Praçeta*, *Lugar*, *Largo*, *Sítio*, *Sitio*, *Avenida*, *Av.*, *Av*, *Quinta*, *Q.* and *Q*);

- Opening Hours: Record must contain both a *week day* reference (e.g. *2ª*, *3ª*, *segunda-feira*, *segunda*, *dias úteis*, etc.) and an hour reference (e.g. *12h*, *14h30*, *14:30 às 18:00*, etc.);

- Phone number: Record can optionally start by a country code like *+351* and then must have 9 digits with optional spaces in between (e.g. *911 111 111*, *91 111 11 11*, etc.).

Table 4.3: RE Rules Experiment 1 - Initial version

Attribute	Macro Precision	Micro Precision	Macro Recall	Micro Recall	Macro F-measure	Micro F-measure
Address	0,784	0,936	0,448	0,439	0,552	0,693
Opening hour	0,352	0,868	0,322	0,119	0,306	0,817
Phone number	0,977	0,981	0,999	0,998	0,987	0,988

From Table 4.3 we can see that, in this experiment, the results for the *phone number* were exceptionally good for all measures, thus this attribute's RE suffered no further changes. On the other hand, the results for the remaining attributes were below expectations. For the *address* attribute, although the precision values are quite high, the recall values are significantly low, which suggests a large presence of both FP and FN (the number of FN being more elevated than FP). While for the *opening hours* attribute, the macro average precision is quite low compared to the micro precision, which suggests a dataset imbalance, meaning that some entities have a really low precision in comparison with others. This could be due to the fact that, for all attributes, the number of true instances is very small in comparison to the number of negative instances, as mentioned in Annotations Dataset). Besides, since the macro average precision is quite low while the micro average precision is high, this suggests that some entities have a lot more FP than others. The opening hour's attribute recall scores are both very low, which means that there are too many instances being labelled with a 0 wrongly. With this, the second RE chosen for the attributes address and opening hours were:

- Address: Record must contain an address starter marker and be followed by a number (e.g. *23*, *4*, etc.) or it must contain a postal code reference (e.g. *1000-260*);

- Opening Hours: Record must contain both a *week day* and an hour reference and this RE assumes that an opening hours' instance can be divided in 2 or more sequential records, the current line being evaluated, its 2 previous lines and its next line, making a total of 4 sequential lines in evaluation at a time (e.g. *13h - 14h* on the current line and *Segunda-feira* on the second line above it, etc.) During this experiment, several variations of the opening hour's REs were experimented,

mainly to discover how many lines were required as context to significantly improve results. From the results it became clear that 3 lines of context besides the current one were the ideal number.

Table 4.4: RE Rules Experiment 2 - Second version

Attribute	Macro Precision	Micro Precision	Macro Recall	Micro Recall	Macro F-measure	Micro F-measure
Address	0,909	0,960	0,804	0,800	0,827	0,884
Opening hour	0,655	0,856	0,713	0,756	0,657	0,855
Phone number	0,977	0,981	0,999	0,998	0,987	0,988

From Table 4.4 we can see that, for the *address* attribute, the precision values improved while the recall values remained high, which means the new REs improved the labelling significantly, thus highlighting the importance of postal codes. For the *opening hours* attribute, both precision and recall values also improved. This last change to the attributes' REs indicates that providing context to the REs while evaluating the possible attribute's values (by using REs to check multiple sequential sentences) is desirable, since it will provide better results. A few more changes were applied to the address attribute but with no relevant improvements, thus its RE suffered no more changes. In this experiment, for the opening hours attribute, one particular problem was noticed after manually analyzing the data. Many address records were tagged as opening hours due to the postal codes or street numbers in them, which the opening hour's REs caught. This is why, for the final implementation, described in detail in Chapter 3, the opening hour's RE were updated to:

- Opening Hours: Record must contain both a *week day* and an hour reference and this RE assumes that an opening hours' instance can be divided in 2 or more records, both the 2 previous lines, the line being currently evaluated and the next line, making a total of 4 sequential lines in evaluation. Also, in order for the record to be identified, it can not contain an *address starter* marker;

Table 4.5: RE Rules Experiment 3 - Final Version

Attribute	Macro Precision	Micro Precision	Macro Recall	Micro Recall	Macro F-measure	Micro F-measure
Address	0,909	0,960	0,787	0,785	0,816	0,873
Opening hour	0,720	0,932	0,713	0,755	0,681	0,862
Phone number	0,977	0,981	0,999	0,998	0,987	0,988

This last change improved the precision while maintaining recall values, which confirmed the previous observations, that, in the data, many address records were being incorrectly labelled as opening hours by the rules.

Although, the final expressions find most cases, they cannot find all possible address values (e.g. *Beco* is also a possible start word in an address, however it is incredibly uncommon and when tested with, it resulted in more incorrect than correct results, reason why it was not considered). Thus, the system is unable to catch instances the REs do not cover, which is one of this method's shortcomings. Despite this and according to Table 4.5 the rules proved to be quite effective, specifically for the *phone number* attribute in all metrics. This is due to the limited number of variations phone numbers can take, which are all easily captured by a general enough RE. Since the average precision and recall for this attribute both neared *1,0*, the rules were considered to be a sufficient enough approach.

For the *opening hours* and *address* attributes, the rules' results were a good indicator, as the best average precision and recall were all above *0,7*. Although the address attribute presented the best results out of the two attributes for both metrics, only the address was chosen to be processed by the system in an alternative evaluation phase, now based in LM and ML techniques. The reason behind this choice lies in the big focus this work has on the entities' geographical locations, which makes it more relevant to correctly classify an address than an opening hour schedule. Besides, during the development of this work, no opening hours datasets written in Portuguese could be found. On the other hand, thanks to the contribution of several Portuguese municipalities, a vast address dataset could be compiled, which is another reason why this attribute was chosen for further experiments.

4.2.2 Language Models Evaluation

LMs are based on word occurrence in documents and in the Markov property, which makes them highly susceptible to a string's length. This means, for example, that a long address will likely have a lower probability than a short sentence that is not an address. Even if some techniques were used to mitigate the string length's impact on the LM's probabilities, problems derived from *data sparsity* (e.g. the word "30" might never appear in training as a street number but might appear in the testing set and become an unknown word), *linguistic variability* (e.g. "avenida" and "av." mean the same) and bias to string length are to be expected in the results.

One of the methods to mitigate this issue is normalization. Numbers are particularly problematic in LMs as which number appears in a sentence is irrelevant, only knowing it is a number and after what word it appears is important. Therefore, the system first converts all postal code instances (e.g." 2300-503", etc.) and phone numbers (e.g. "213 457 111", etc.) into the fixed tags "_postal_code_" and "_phone_number_", respectively. Then all remaining numbers are converted into the fixed tag "_lone_number_" so the variation of number values does not affect the predictive ability of the LM. Finally, all instances of "av." and "r." were converted to "avenida" and "rua", respectively, so that these relevant words and abbreviations are considered the same by the LM. The application of this normalization im-

proved all LMs variations' results slightly, so its results are not shown but are applied in all following experiments.

The LMs used throughout this section learned both their bigrams and trigrams from the Official Address Dataset and Yellow Pages Dataset, since both were vast collections of valid address records. Both datasets suffered the normalization step as well as the testing set, Annotations Dataset. The threshold values - 10^{-30}, 10^{-50}, 10^{-80} and 10^{-100} - were selected after a manual observation of the scores obtained by both the bigram and trigram model. Of course, no threshold was a perfect division of the two types of sentences (address or not an address), since there were many unknown words and a bias to string length is always present in LMs. The LM score values were then combined with the threshold information and used to give a label to each sentence. These labels are finally compared with the true annotations from the Annotations Dataset and measures are calculated based on them, which can be seen in Table 4.6, Table 4.7 and Table 4.8. These experiments aim to find the best LM variation to classify a dataset based on the address attribute. The variations include the type of smoothing technique used, the type of *n*-gram used and the combination of RE rules with LM.

Table 4.6: LM Results Experiment 1 - Normalized Dataset with Bigrams

Language Model	Threshold	Macro Precision	Micro Precision	Macro Recall	Micro Recall	Macro F-measure	Micro F-measure
Laplace	10^{-30}	0,031	0,015	0,780	0,764	0,057	0,066
Laplace	10^{-50}	0,029	0,016	0,911	0,926	0,055	0,058
Laplace	10^{-80}	0,031	0,016	**0,975**	**0,966**	0,058	0,059
Laplace	10^{-100}	0,030	0,016	**0,975**	**0,966**	0,057	0,058
Witten-Bell	10^{-30}	**0,152**	**0,045**	0,605	0,695	0,218	**0,335**
Witten-Bell	10^{-50}	0,145	0,044	0,746	0,856	**0,219**	0,289
Witten-Bell	10^{-80}	0,145	0,043	0,750	0,864	**0,219**	0,287
Witten-Bell	10^{-100}	0,143	0,043	0,750	0,864	0,217	0,284
Kneser-Ney	10^{-30}	0,032	0,015	0,747	0,762	0,060	0,072
Kneser-Ney	10^{-50}	0,030	0,016	0,910	0,925	0,056	0,060
Kneser-Ney	10^{-80}	0,030	0,016	0,940	0,949	0,057	0,060
Kneser-Ney	10^{-100}	0,030	0,016	**0,975**	**0,966**	0,057	0,059

As can be seen in Table 4.6 and Table 4.7, bigrams outperform trigrams in general, specifically when the Witten-Bell smoothing technique is used. However, all precision results are incredibly low, while recall values are quite high, which leads to small f-measure scores as expected. These are all indicators that the LM is classifying the majority of instances as 1, wrongly. This conclusion can be achieved since the number of FP appears to be quite high and the number of FN quite low, from looking at the precision and recall scores, respectively. This can also be related with the fact that the majority of sentences in the Annotations Dataset are negative examples of the address attribute. In order to check the correctness

Table 4.7: LM Results Experiment 2 - Normalized Dataset with Trigrams

Language Model	Threshold	Macro Precision	Micro Precision	Macro Recall	Micro Recall	Macro F-measure	Micro F-measure
Laplace	10^{-30}	0,029	0,015	0,788	0,816	0,055	0,063
Laplace	10^{-50}	0,029	0,016	0,913	0,933	0,054	0,057
Laplace	10^{-80}	0,030	0,016	**0,975**	**0,966**	0,057	0,059
Laplace	10^{-100}	0,030	0,016	**0,975**	**0,966**	0,057	0,058
Witten-Bell	10^{-30}	0,037	0,019	0,574	0,670	0,068	0,094
Witten-Bell	10^{-50}	**0,047**	**0,023**	0,762	0,857	**0,086**	**0,104**
Witten-Bell	10^{-80}	**0,047**	**0,023**	0,768	0,869	0,085	0,102
Witten-Bell	10^{-100}	**0,047**	**0,023**	0,769	0,869	0,085	0,102
Kneser-Ney	10^{-30}	0,030	0,016	0,844	0,857	0,057	0,063
Kneser-Ney	10^{-50}	0,029	0,016	0,914	0,934	0,055	0,058
Kneser-Ney	10^{-80}	0,031	0,016	**0,975**	**0,966**	0,058	0,059
Kneser-Ney	10^{-100}	0,030	0,016	**0,975**	**0,966**	0,057	0,058

of this deduction, one more experiment was conducted.

Table 4.8: LM Results Experiment 3 - Normalized Dataset with Bigrams and Rules

Language Model	Threshold	Macro Precision	Micro Precision	Macro Recall	Micro Recall	Macro F-measure	Micro F-measure
Laplace	10^{-30}	0,798	0,997	0,662	0,694	0,702	0,904
Laplace	10^{-50}	0,927	0,996	0,874	0,929	0,893	0,964
Laplace	10^{-80}	**0,951**	0,981	**0,937**	**0,971**	**0,940**	**0,976**
Laplace	10^{-100}	**0,951**	0,981	**0,937**	**0,971**	**0,940**	**0,976**
Witten-Bell	10^{-30}	0,800	**1,000**	0,617	0,631	0,679	0,871
Witten-Bell	10^{-50}	0,867	**1,000**	0,718	0,786	0,772	0,906
Witten-Bell	10^{-80}	0,867	**1,000**	0,725	0,801	0,776	0,911
Witten-Bell	10^{-100}	0,867	**1,000**	0,725	0,801	0,776	0,911
Kneser-Ney	10^{-30}	0,798	0,997	0,679	0,713	0,722	0,917
Kneser-Ney	10^{-50}	0,894	0,995	0,840	0,925	0,859	0,962
Kneser-Ney	10^{-80}	0,919	0,987	0,882	0,943	0,894	0,964
Kneser-Ney	10^{-100}	**0,951**	0,981	**0,937**	**0,971**	**0,940**	**0,976**

In the last experiment performed, the LMs were combined with the address RE rules, described in detail in Section 3.4.1. In this situation, the rules were first applied to the Annotations Dataset, the sentences which matched the address rules were then processed by the LMs, thus obtaining the results seen in Table 4.8. These results showed a tremendous improvement, presenting both high results for precision, recall and f-measure, particularly for the Laplace and Kneser Ney smoothing techniques, but

because the LM was only used to label sentences that the RE rules matched. The similar success of both Laplace and Kneser Ney techniques lead us to believe that the attribution of some probability mass to unknown words does not necessarily result in a more accurate predictive model. This suggests that giving unknown words a very small probability still outputs decent results, although it is debatable if this would be equally valid for other datasets.

These preliminary LM results proved to be quite weak, since the good results of the final experiment in Table 4.8 were mostly due to the application of the RE patterns to filter the sentences the LM evaluated. Otherwise, the LM results would still be lacking and its scores would hardly be able to distinguish between correct and incorrect address values. In conclusion, this approach was not explored further due to poor results and since other approaches seemed more promising (ML techniques).

4.2.3 Classifiers and Supervised Learning

The third technique the system applies is based on machine learning algorithms such as, NB, SVM, LR, DT and RF. In this phase, the system performed on only one of the three attributes: address, as mentioned previously. In this section, the main objective was evaluating how good the developed classifiers were at recognizing valid address values.

The classifier starts by looking at the data with which it will be trained. Since the Annotations Dataset is mainly composed of negative instances, two experiments were performed, one for the original imbalanced dataset and another for a balanced version of the dataset, displayed respectively in Table 4.9 and Table 4.10. Both versions of the dataset were normalized with the same method applied in the LMs. The normalization procedure consisted in converting all postal code instances. phone numbers and remaining numbers into the fixed tags "_postal_code_", "_phone_number_" and "_lone_number_", respectively. Finally, it converted all instances of "av." and "r." to "avenida" and "rua", respectively.

Firstly, in Table 4.9 and Table 4.10 a summary of all metrics (precision, recall and f-measure) from the five used classifiers (Naive Bayes, Support Vector Machine, Logistic Regression, Decision Tree and Random Forest, respectfully represented in the following tables by NB, SVM, LR, DT and RF) regarding the differences between using a balanced and a non-balanced dataset can be analyzed.

From Table 4.9 and Table 4.10, it is clear that most classifiers perform more reasonably for a balanced dataset. Besides, using a not balanced dataset in this work often results in overly excessive predictions of negative instances, as we can see if we look at the same measures but for each specific class, represented in Table 4.11. Negative instances have clearly higher metric results, while the positive class has much worse results. For the balanced dataset, both classes have more similar metrics and not so much discrepant values between classes.

The classifiers with the highest performance both in f-measure, but with similarly high scores for

Table 4.9: ML Experiment 1 - Not Balanced Dataset

Classifier	Vectorizer	Macro Precision	Macro Recall	Macro F-measure
RF	Count	0,976	0,855	0,906
RF	Hash	0,960	0,709	0,787
RF	Tfidf	0,971	0,790	0,858
DT	Count	0,924	0,870	0,895
DT	Hash	0,949	**0,935**	**0,942**
DT	Tfidf	0,942	0,886	0,913
SVC	Count	0,921	0,934	0,928
SVC	Hash	0,962	0,903	0,930
SVC	Tfidf	0,945	0,903	0,923
LR	Count	0,947	0,919	0,932
LR	Hash	0,976	0,855	0,906
LR	Tfidf	**0,979**	0,903	0,938
NB	Count	0,904	0,645	0,712
NB	Hash	0,493	0,500	0,496
NB	Tfidf	0,904	0,645	0,712

Table 4.10: ML Experiment 2 - Balanced Dataset

Classifier	Vectorizer	Macro Precision	Macro Recall	Macro F-measure
RF	Count	0,929	0,928	0,928
RF	Hash	0,928	0,929	0,928
RF	Tfidf	**0,930**	**0,931**	**0,929**
DT	Count	0,929	0,928	0,928
DT	Hash	0,914	0,914	0,914
DT	Tfidf	0,929	0,928	0,928
SVC	Count	0,900	0,900	0,900
SVC	Hash	0,900	0,900	0,900
SVC	Tfidf	0,865	0,862	0,857
LR	Count	0,900	0,900	0,900
LR	Hash	0,914	0,916	0,914
LR	Tfidf	0,883	0,877	0,871
NB	Count	0,770	0,622	0,547
NB	Hash	0,750	0,554	0,431
NB	Tfidf	0,770	0,622	0,547

precision and recall, are the *Random Forests* for all vectorizers, but especially with a *Tf-Idf vectorizer*, with scores around *0,9*. Surprisingly enough *Decision Trees* are not to far behind in scores for all vectorizers, while *Support Vector Machines* and *Logistic Regressions* follow right behind with scores

Table 4.11: ML Experiment 1.A - Not Balanced Dataset By Class

Classifier	Vectorizer	0 Precision	0 Recall	0 F-measure	1 Precision	1 Recall	1 F-measure
RF	Count	0,996	**1,000**	**0,998**	0,957	0,710	0,815
RF	Hash	0,992	**1,000**	0,996	0,929	0,419	0,578
RF	Tfidf	0,994	**1,000**	0,997	0,947	0,581	0,720
DT	Count	0,996	0,998	0,997	0,852	0,742	0,793
DT	Hash	**0,998**	0,999	**0,998**	0,900	**0,871**	0,885
DT	Tfidf	0,997	0,999	**0,998**	0,889	0,774	0,828
SVC	Count	**0,998**	0,998	**0,998**	0,844	**0,871**	0,857
SVC	Hash	0,997	0,999	**0,998**	0,926	0,806	0,862
SVC	Tfidf	0,997	0,999	**0,998**	0,893	0,806	0,847
LR	Count	**0,998**	0,999	**0,998**	0,897	0,839	0,867
LR	Hash	0,996	**1,000**	**0,998**	0,957	0,710	0,815
LR	Tfidf	0,997	**1,000**	**0,998**	**0,962**	0,806	**0,877**
NB	Count	0,990	0,999	0,995	0,818	0,290	0,429
NB	Hash	0,986	**1,000**	0,993	0,000	0,000	0,000
NB	Tfidf	0,990	0,999	0,995	0,818	0,290	0,429

of *0,8*. Finally, *Naive Bayes* performs quite poorly very likely due to its extreme feature independence assumptions. The great results obtained by the RFs comply with Aggarwal's [28] statement that two of the most powerful classifiers in ML are random forests and kernel support vector machines. The results obtained seem to agree with this affirmation, since SVMs are not far behind RFs and present high f-scores (around *0,8*).

Since both the RE and ML obtained quite reasonable results, one final experiment was developed to evaluate if a combination of both could improve results significantly. Thus, the approach taken was to use the same methodology as previous experiments. First to separate the Annotations Dataset into train and test sets, fit the classifiers with the data and when testing, check for each prediction's confidence. If the confidence of the classifier according to the *predict_proba* function is below a certain threshold, then use the RE rules instead to decide if the sentence is an address value or not. This approach aims to mitigate less trustworthy classifier predictions, by using the rules to classify such instances.

Several thresholds were tested (e.g. *0,5*), the value which gave the best results was *0,6*. The classifier's metrics for a threshold of *0,6* in the Mixed Classifier are described in Table 4.12, and show better results for all classifiers than the results in the balanced experiment, displayed in Table 4.10. This indicates that since the rules are quite good at classifying correctly an address value they can even improve the classifier's results when its confidence in some predictions is lacking.

Table 4.12: ML Experiment 3 - Mixed Classifier (ML + RE) with a Balanced Dataset and a *0,6* Threshold

Classifier	Vectorizer	Macro Precision	Macro Recall	Macro F-measure
RF	Count	0,944	0,944	0,943
RF	Hash	0,957	**0,958**	**0,957**
RF	Tfidf	**0,959**	**0,958**	**0,957**
DT	Count	**0,959**	**0,958**	**0,957**
DT	Hash	0,878	0,873	0,871
DT	Tfidf	0,900	0,900	0,900
SVC	Count	0,929	0,929	0,929
SVC	Hash	0,947	0,944	0,943
SVC	Tfidf	0,859	0,856	0,857
LR	Count	0,929	0,929	0,929
LR	Hash	0,925	0,917	0,914
LR	Tfidf	0,859	0,856	0,857
NB	Count	0,858	0,840	0,840
NB	Hash	0,677	0,668	0,666
NB	Tfidf	0,858	0,840	0,840

Further considerations regarding the Results

The *Naive Bayes* classifier assumes independent features which contribute equally to the outcome result. This assumption is almost never true in real world problems, however, which sometimes explains this technique's poor results. Despite this, NB often works well in practice and it has the advantage of requiring a small amount of training data to perform decently well in some contexts. Lastly, they also perform extremely fast compared to some more sophisticated methods.

Support Vector Machines can only perform binary classification since they find a hyper-plane to separate the types of data with a boundary. SVMs are quite fast and usually perform well, as can be seen from this section's experiments, for both low and high dimensional data. Besides they can specify different kernel functions so that the boundary has different behaviors. Unfortunately, its efficiency decreases as the training size increases and the results are quite difficult to interpret.

The *Logistic Regression* classifier learns a linear relationship from a dataset, similarly to SVM, by use of the sigmoid function. LR are simple to implement, fast and efficient to train, and often result in a good accuracy for many simple datasets. Besides, it performs well if the data is linearly separable, even if in the real world linear separable data is rarely found.

Decision Trees can handle high dimensional data and often have a reasonable accuracy. This technique is mostly used to obtain understandable knowledge regarding the process of decision behind the classification process. Another advantage of DT is that it provides a clear indication of which features are the most important for the prediction. DT, however, are prone to errors in the presence of many

classes and relatively small number of training instances. Besides, training these models is computationally expensive since each candidate splitting field must be sorted before its best split is found. Even if pruning algorithms are used to reduce the number of calculations, they can also be expensive since many candidate sub-trees must be formed and compared.

Random Forest classifiers combine multiple decision trees, instead of relying on the predictions of a single tree which may present a high variance, thus obtaining a better predictive model with increased accuracy. This process of combining the output of multiple individual models (also known as weak learners) is called *Ensemble Learning*. Since a RF consists of an aggregation of trees it is difficult to interpret its predictions. However, RFs are able to deal with high dimensionality datasets. This large dataset capacity leads the algorithm to be computationally intensive and take a long time in the training phase.

A curious conclusion taken from all experiments was that NB is not an adequate solution for this specific problem, as it presented the worst results for all vectorizers. This could be explained due to the fact that Naive Bayes are not able of representing very complex behavior like DT and RF models, and assume a very strict feature independence condition, even if they do have some advantages like small training time and the impossibility of overfitting.

On another hand, both SVMs and LRs performed quite well, having metric values around *0,8* on average. Besides, SVMs are known to be faster and less at risk of overfitting (meaning the classifier learns the training data too closely). One fundamental difference, however, between SVMs and LRs is that the first tries to find the best geometrical margin that separates all the classes, thus reducing the risk of error, and the second also tries to find the best margin but it can have different decision boundaries with each different weight that is near the optimal path. In this work, both techniques presented similar results above a *0,8* threshold, making them both good options for the system's solution.

Yet, both DTs and RFs surpassed the results obtained by the SVMs and LRs. Typically, DTs are a preferred classification method as their decision process can be easily interpreted and does not require a large dataset. Besides LR has a single decision boundary and as such a decision threshold has to be set, while DT automatically handles decision making, by bisecting the space into smaller spaces at each split until a final decision is reached. However, DTs are prone to overfitting as they are greatly affected by noise, which could be a reason to opt for other options like SVM or LR. In this work's experiments though, DT produced better results for all measures and is the most interpretable approach, so in this case it would be the best option.

When comparing the feature importance of both DTs and RFs, often the decision tree model gives high importance to a particular set of features, while a random forest chooses features randomly during training. Therefore, RFs do not depend highly on any specific set of features, which allows them to generalize data in a better way and be much more accurate than a decision tree. This is proved in

Table 4.10 where RFs outperform DTs, which leads us to choose RFs as the best approach to solve this work's problem. However, RFs have a higher training time than a single DT and are not as interpretable. An increase in the number of trees in a random forest, results in an increase in the time taken to train each of the trees. Therefore, if training time and interpretability are more important, DTs are the best technique, if predictive accuracy is more relevant then RFs would be a better choice.

4.2.4 Classifiers and Semi-Supervised Learning

Semi-supervised learning falls at a category between unsupervised learning, which means there is no labeled training and testing data available, and supervised learning methods, which have labeled training data available. Semi-supervision learning is a category of techniques used to improve model performance, particularly for smaller samples of labeled data, that uses both labeled and unlabeled data to perform predictions [60]. Before a large dataset of unlabeled data which can not be used in a classification problem, semi-supervised learning proposes the use of those large datasets by attributing the same labels to items similar to the labeled training dataset. Semi-supervised learning approaches often use the model itself to decide which instances it believes to belong to certain classes with more certainty. This serves to reinforce the beliefs of the model by iterating through the various possible unlabeled samples.

The amount of available data in the Annotations Dataset was not very large as it was manually annotated, which might have impacted the results obtained in previous experiments of this section. In situations like this where annotated data is lacking, techniques like semi-supervised classification provide some help, since they allow for the use of non-annotated data to train classifiers. Using semi-supervised approaches allows for the use of large datasets of unlabeled data available online, which is very useful in this thesis' context as there are many available unlabeled geographical and entity related datasets, but very few labeled and free. Mainly in this work, there are two large unlabeled geographical datasets available, Official Address Dataset and Yellow Pages Dataset, which had limited use in this thesis as they were both unlabeled. The use of both these datasets in the classification experiments is expected to provide better results as it reinforces the beliefs the model gained in the training phase and adds more address word variations.

Two experiments are run for semi-supervised classification. The first - Experiment 4.1 - considers only an annotated dataset, while the second - Experiment 4.2 - uses several different unlabeled datasets. Experiment 4.1 uses the Annotations Dataset, normalizes the dataset (with the same method applied in the LMs), balances the dataset (by using the Python's function *random.sample* to randomly select the same number of negative as the number of positive instances) and then splits it two times with the Scikit-Learn's function *train_test_split*. The first split saves 30% for the test set and 70% for the training

set. The second split produces the unlabeled set by separating the previous training set in half, 50% for the training set and 50% for the unlabeled set.

On the other hand, experiment 4.2 uses the Annotations Dataset, normalizes the dataset (with the same method applied in the LMs), balances the dataset (by using the Python's function *random.sample* to randomly select the same number of negative as the number of positive instances) and then splits it with the Scikit-Learn's function *train_test_split*, 30% for the test set and 70% for the training set. The unlabeled sets considered in this experiment are, for the positive instances, the datasets Official Address Dataset and Yellow Pages Dataset and, for the negative instances, all sentences with a 0 label from the dataset Annotations Dataset.

For both experiments, the classifiers are trained with the selected training data and then the unlabeled datasets are evaluated with the Scikit-Learn's function *predict_proba*, which returns the probability of the evaluated string belonging to a certain class, thus translating the classifier's label belief for that instance. Then, the strings with the highest probabilities of belonging to the positive and negative class are both added to the training dataset of the classifier and removed from the unlabeled datasets. Finally, the classifier is re-trained with the new training set. This process is repeated for 10 rounds and to obtain results, that are later translated into the metrics displayed in the Table 4.13 and Table 4.14, the classifier processes the test set which remains untouched throughout the whole process and tries to label it, thus obtaining predictions with which to calculate the needed metrics to evaluate the semi-supervised learning technique's success.

For these experiments, a summary of all metrics (precision, recall and f-measure) for the Support Vector Machine classifier with a Count Vectorizer can be analyzed in Table 4.13 and Table 4.14. The classifier select was SVM as it is not sensible to disproportions among classes and the CountVectorizer was chosen since it presented better results compared to other vectorization techniques. The datasets considered in this experiment are Annotations Dataset for Experiment 4.1 and both Annotations Dataset (negative examples), Official Address Dataset and Yellow Pages Dataset (positive examples) for Experiment 4.2.

It becomes clear quite quickly that enlarging the training dataset in this context with the classifier's beliefs of each sample of unlabeled data does not necessarily produce better results. Both experiments do present some improvements (depending on the number of rounds of enlargement of the dataset), but they are very erratic and follow no pattern, thus are not very reliable. This could be explained, in Experiment 4.2, by the fact that the instances added at each round were not necessarily correct addresses, since the Official Address Dataset was collected through municipalities and could possess some errors, and, in Experiment 4.1, it could be explained by manual errors when annotating. Both are unlikely but possible causes. However, all instances picked by the *predict_proba* function were considered to be an address with higher probability according to what the classifier had learned so

Table 4.13: ML Experiment 4.1 - SVM with CountVectorizer and Annotations Dataset

Attribute	Round	Macro Precision	Macro Recall	Macro F1-Measure
0	0	0,881	**1,000**	0,937
0	1	0,881	**1,000**	0,937
0	2	0,841	**1,000**	0,914
0	3	0,881	**1,000**	0,937
0	4	0,881	**1,000**	0,937
0	5	0,860	**1,000**	0,925
0	6	0,881	**1,000**	0,937
0	7	0,881	**1,000**	0,937
0	8	**0,949**	**1,000**	**0,974**
0	9	0,902	**1,000**	0,949
0	10	0,878	0,973	0,923
1	0	**1,000**	0,848	0,918
1	1	**1,000**	0,848	0,918
1	2	**1,000**	0,788	0,881
1	3	**1,000**	0,848	0,918
1	4	**1,000**	0,848	0,918
1	5	**1,000**	0,818	0,900
1	6	**1,000**	0,848	0,918
1	7	**1,000**	0,848	0,918
1	8	**1,000**	**0,939**	**0,969**
1	9	**1,000**	0,879	0,935
1	10	0,966	0,848	0,903
macro	0	0,940	0,924	0,927
macro	1	0,940	0,924	0,927
macro	2	0,920	0,894	0,897
macro	3	0,940	0,924	0,927
macro	4	0,940	0,924	0,927
macro	5	0,930	0,909	0,913
macro	6	0,940	0,924	0,927
macro	7	0,940	0,924	0,927
macro	8	**0,974**	**0,970**	**0,971**
macro	9	0,951	0,939	0,942
macro	10	0,922	0,911	0,913

far. Therefore, the problem could also lie in the small size of the dataset, which leads the classifier to learn insufficient data to correctly classify new instances, even after increasing its training dataset with unlabeled data.

Table 4.14: ML Experiment 4.2 - SVM with CountVectorizer and Official Address Dataset

Attribute	Round	Macro Precision	Macro Recall	Macro F1-Measure
0	0	0,927	0,927	0,927
0	1	0,909	0,976	0,941
0	2	0,927	0,927	0,927
0	3	0,950	0,927	0,938
0	4	0,952	0,976	0,964
0	5	0,932	**1,000**	0,965
0	6	0,911	**1,000**	0,953
0	7	0,909	0,976	0,941
0	8	**0,976**	0,976	**0,976**
0	9	0,930	0,976	0,952
0	10	0,953	**1,000**	**0,976**
1	0	0,897	0,897	0,897
1	1	0,962	0,862	0,909
1	2	0,897	0,897	0,897
1	3	0,900	0,931	0,915
1	4	0,964	0,931	0,947
1	5	**1,000**	0,897	0,945
1	6	**1,000**	0,862	0,926
1	7	0,962	0,862	0,909
1	8	0,966	**0,966**	**0,966**
1	9	0,963	0,897	0,929
1	10	**1,000**	0,931	0,964
macro	0	0,912	0,912	0,912
macro	1	0,935	0,919	0,925
macro	2	0,912	0,912	0,912
macro	3	0,925	0,929	0,927
macro	4	0,958	0,953	0,956
macro	5	0,966	0,948	0,955
macro	6	0,956	0,931	0,940
macro	7	0,935	0,919	0,925
macro	8	0,971	**0,971**	**0,971**
macro	9	0,947	0,936	0,940
macro	10	**0,977**	0,966	0,970

4.3 Final Validations

OSM Validation

In order to execute the first comparison (with OSM information), a test dataset was needed to compare the results obtained by the system with the OSM entities' values. For this, as mentioned before, the OSM Attributes Dataset was used. This comparison was done manually as the Portuguese language has many equivalent variations for the same address or opening hours (e.g. OSM will contain for the entity *Fundação Calouste Gulbenkian*, "Av. de Berna, 45A, 1067-001 Lisboa", but the system could return "Avenida de Berna 45A 1067-001" as the address, which although are very similar formats are not identical). Besides, the address format contained on an entity's website and the address format of the entity extracted from the OSM platform might be extremely different as there are no universal standards on address written formulation.

For example, for the entity "ShariSushi Bar", the address information provided in OSM is "Avenida Mendes Silva" as a street, "279" as house number, "3030-193" as postal code, "Coimbra" as city, while the information returned by our system was "Av. Mendes Silva Nº 279/281 1ºAndar", "3030–193 Coimbra" and "Av. Mendes Silva Nº 279/281 1ºAndar". Except for "3030–193 Coimbra", the format of the remainder parts of the address information is different, mainly due to variations like "Avenida" and "Av.", "279" and "Nº 279/281". In particular, our system even adds one new detail "1º andar", which OSM did not make available. There are other cases where the information contained in OSM was plain incorrect (e.g. for the entity "Restaurante do Clube Naval de Lisboa", the postal code in OSM was "1400-032" and the one found both by our system and manually in the website was "1400–038").

This validation phase was performed manually and the variation of the system used was the REs rules (Section 3.4.1) as it was the only approach that found values for all attributes. For the *address* attribute, in one hand, both our system and OSM found equivalent attribute values for around 50% of the selected entities. On another hand, our system found more detailed information for 15% of entities and found less data than OSM in another 15%. This leads us to believe that, in terms of addresses, OSM has a reasonable accuracy and our approach does not improve it significantly. For the *opening hours* attribute, our system mostly fixed wrong opening hour values in OSM, which was to be expected as in volatile entities, like the ones considered in this work, the working schedule often changes, thus easily becoming outdated. For the *phone number* attribute, our results were either equivalent or we added new information to the OSM platform, since usually it contains only one value for the phone number, while the actual entities typically have more than one phone number. When assuming attribute values as equivalent, we ignored variations in format as there were many and they did not change the meaning of the values. This validation was done manually since an automated tool would have trouble identifying such variations as the same value, due to the use of abbreviations, etc. An automatic alternative to this process would be to use a tool like *libpostal*[4], to convert all address values to the same format, for

[4] `https://github.com/openvenues/libpostal`

example. Due to time constraints and the necessity of performing a qualitative analysis (if our system performed better or worse than OSM), this comparison was done manually, which for 30 entities was simple enough. However, if more entities were added, it would quickly become a strenuous task, and other approaches would have to be evaluated.

Truth Validation

On another hand, to execute the second comparison with the actual true entities' values, the Truth Base Dataset was used. The system's results were compared against the manually collected information taken from the entities' websites. The comparison process consisted on a manual check to ensure the correctness of the results, since the information contained in the Truth Base Dataset was extracted exactly as it was written on the website but the copied information might not be exactly the same (e.g. the process of extraction of the system can sometimes break sentences in two).

After a manual verification, by using the RE technique of the system, for the phone number, the system found all valid instances which makes sense after looking at the results in Section 4.2.1, while for the address and opening hour attributes, the system found only around 66% of all correct values for that attribute. This was to be expected due to the lower metric results obtained previously for these two attributes.

5

Conclusion

Contents

5.1 Conclusions

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.

For the RE method variation, Section 4.2.1, the system obtained f-measure scores of *0,816* for the address, *0,681* for the opening hours and *0,987* for the phone number attribute. For the LM variation, Section 4.2.2, without using a filter, the system obtained f-measures of *0,086* and, with the filter, a maximum score of *0,940*, for the address. Finally, for the ML variation, Section 4.2.3, 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 hour* 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 hour* attributes.

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 OpenStreetMap 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.

5.2 Limitations and Future Work

In the web extraction phase of the system, the discovery of the official web pages is done via the OSM platform. However, the websites contained in it are often inaccurate or no longer valid. This makes it so that the *Scrapy*'s extraction process may fail if the URL is invalid or if it no longer exists. Thus, the system should use a web crawling method, as described in Srinivasa's work [38], to automatically collect the relevant websites' URLs. One possible approach could be to query a search engine for an entity's name [48] and use heuristics or other type of classification technique (which will require a labelled dataset of non-official and official websites) to recognize the entity's website.

Another clear limitation of this work is the crawling depth threshold (e.g. for the entity *AudiçãoActiva*, the crawler goes to the tab *Lojas* of the entity's website and there it finds a map with a list of Portuguese municipalities where there are *AudiçãoActiva*'s shops. However, the system fails to follow those links, which have the shop's address, because no text in that tab contains a keyword). This limit was an assumption made at the beginning of this thesis to avoid an exponential scraping endeavour and also to keep the search in reasonable time and memory consumption levels. Besides, the bigger the crawling limit, the less relevant content will be found at each new depth. In case computational power is not an issue, this limit could be increased as it can slightly improve results.

Another phase of the system which could be improved is the language models' approach. In this work, language models were not investigated thoroughly as the preliminary results were quite poor, as mentioned in Chapter 4. In future iterations, however, to improve results, other smoothing techniques and *n*-gram combinations, like 4-grams or 5-grams, could be explored. Additionally, the measures chosen to evaluate the LMs were precision, recall and f-measure, to allow for a direct comparison to other approaches, but others could be tested, such as perplexity and entropy (both mentioned previously in Chapter 2).

Also, instead of applying the LMs to whole sentences, which could lead to low probabilities due to length bias, the system could classify small partitions of each sentence (e.g. in "O restaurante localiza-se na Rua do Carmo.", *Rua do Carmo* is a correct address, but since it is part of a longer sentence, the LM gives the sentence a lower probability than if it was only "Rua do Carmo"). The size selected for the partitions (e.g. "Rua do Carmo", "localiza-se na Rua do Carmo", etc.) should take into account the average word length of an address, otherwise they might include too many irrelevant words or even cut important parts of the address. To account for the high variation in address length, the system should analyze each sentence with a few different partition sizes so as to return the portion of the sentence with the highest score. Since LMs learn only positive instances (in this case, addresses) and the dataset where they are applied, Annotations Dataset has sentences that only partially contain addresses, this

change could improve this technique's results by mitigating the effects of sentence length bias. This experiment was not explored further due to time constraints and the lacking results of the LM preliminary tests. The use of partitions could also be applied to the ML classifiers, to reduce the impact of irrelevant context in their labelling decision process.

There are several NLP tools with interesting characteristics to this work, such as *STRING* which, according to Mamede et al. [61], is a Hybrid Statistical and Rule-Based Natural Language Processing Chain for Portuguese that performs useful NLP tasks, such as Named Entity Recognition, Information Retrieval, Anaphora Resolution, etc. *STRING* is capable of identifying certain types of information, like people's names, locations, streets or even quantities, it could be an interesting solution to this work's problem. Unfortunately, due to time constraints it could not be explored further.

Bibliography

[1] E. Cambria and B. White, “Jumping NLP curves: A review of natural language processing research,” *IEEE Computational Intelligence Magazine*, vol. 9, no. 2, pp. 48–57, 2014.

[2] V. B. Kadam and G. K. Pakle, “A Survey on HTML Structure Aware and Tree Based Web Data Scraping Technique,” *International Journal of Computer Science and Information Technologies*, vol. 5, no. 2, pp. 1655–1658, 2014.

[3] K.-T. Chang, *Introduction to Geographic Information Systems 9th edition*. McGraw-Hill Education, 2019.

[4] T. Bahaire and M. Elliott-White, “The application of geographical information systems (GIS) in sustainable tourism planning: A review,” *Journal of Sustainable Tourism*, vol. 7, no. 2, pp. 159–174, 1999.

[5] A. D. Jhingran, N. Mattos, and H. Pirahesh, “Information integration: A research agenda,” *IBM Systems Journal*, vol. 41, no. 4, pp. 555–562, 2002.

[6] E. Vargiu and M. Urru, “Exploiting web scraping in a collaborative filtering- based approach to web advertising,” *Artificial Intelligence Research*, vol. 2, no. 1, pp. 44–54, 2012.

[7] D. K. Mahto and L. Singh, “A dive into Web Scraper world,” *Proceedings of the 10th INDIACom; 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016*, pp. 689–693, 2016.

[8] B. Ghosh Dastidar, D. Banerjee, and S. Sengupta, “An Intelligent Survey of Personalized Information Retrieval using Web Scraper,” *I. J. Education and Management Engineering*, vol. 5, no. September, pp. 24–31, 2016.

[9] N. Marres and E. Weltevrede, “SCRAPING THE SOCIAL?: Issues in live social research,” *Journal of Cultural Economy*, vol. 6, no. 3, pp. 313–335, 2013.

[10] J. Gracia and E. Mena, “Semantic heterogeneity issues on the web,” *IEEE Internet Computing*, vol. 16, no. 5, pp. 60–67, 2012.

[11] R. Dale, H. Moisl, and H. L. Somers, *Handbook of natural language processing*. Marcel Dekker, 2000.

[12] D. Jurafsky and J. H. Martin, *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1999.

[13] Y. Kobayashi, T. Yoshida, K. Iwata, and H. Fujimura, "Slot filling with weighted multi-encoders for out-of-domain values," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2019-Septe, pp. 854–858, 2019.

[14] S. Chen and S. Yu, "WAIS: Word Attention for Joint Intent Detection and Slot Filling," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 1, pp. 9927–9928, 2019.

[15] L. Pan, Y. Zhang, F. Ren, Y. Hou, Y. Li, X. Liang, and Y. Liu, "A Multiple Utterances based Neural Network Model for Joint Intent Detection and Slot Filling," *CEUR Workshop Proceedings*, vol. 2242, pp. 25–33, 2018.

[16] C. Zhang, Y. Li, N. Du, W. Fan, and P. S. Yu, "Joint Slot Filling and Intent Detection via Capsule Neural Networks," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 5259–5267.

[17] H. Chen, X. Liu, D. Yin, and J. Tang, "A Survey on Dialogue Systems: Recent Advances and New Frontiers," *SIGKDD Explorations Newsletter*, vol. 19, pp. 25–35, 2017.

[18] D. Serdyuk, Y. Wang, C. Fuegen, A. Kumar, B. Liu, and Y. Bengio, "Towards End-to-end Spoken Language Understanding," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5754–5758.

[19] G. Tur, D. Hakkani-Tür, D. Hillard, and A. Celikyilmaz, "Towards unsupervised spoken language understanding: Exploiting query click logs for slot filling," in *Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech*, no. August, 2011, pp. 1293–1296.

[20] M. Surdeanu and H. Ji, "Overview of the English Slot Filling Track at the TAC2014 Knowledge Base Population Evaluation," *Proceedings of the Text Analysis Conference - Knowledge Base Population 2014*, 2014.

[21] H. Ji and J. Nothman, "Overview of TAC-KBP2016 Tri-lingual EDL and Its Impact on End-to-End Cold-Start KBP," in *Proceedings of the Text Analysis Conference - Knowledge Base Population 2016*, 2016.

[22] H. Ji, J. Nothman, B. Hachey, and R. Florian, “Overview of TAC-KBP2015 Tri-lingual Entity Discovery and Linking,” in *Proceedings of the Text Analysis Conference - Knowledge Base Population 2015*, 2015.

[23] M. Haklay, “How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets,” *Environment and Planning B: Planning and Design*, vol. 37, no. 4, pp. 682–703, 2010.

[24] M. Haklay and P. Weber, “OpenStreet map: User-generated street maps,” *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[25] D. H. Dalip, M. A. Gonçalves, M. Cristo, and P. Calado, “A General Multiview Framework for Assessing the Quality of Collaboratively Created Content on Web 2.0 Daniel,” *Journal of the Association for Information Science and Technology*, vol. 68, no. 2, pp. 286–308, 2017.

[26] D. Fensel, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2005.

[27] M. Graham and T. Shelton, “Geography and the future of big data, big data and the future of geography,” *Dialogues in Human Geography*, vol. 3, no. 3, pp. 255–261, 2013.

[28] C. C. Aggarwal, *Machine Learning for Text*. Springer International Publishing, 2018.

[29] D. Lewandowski, “Web searching, search engines and Information Retrieval,” *Information Services and Use*, vol. 25, no. 3-4, pp. 137–147, 2005.

[30] Y. Zhai and B. Liu, “Web data extraction based on partial tree alignment,” in *Proceedings of the 14th International Conference on World Wide Web*, 2005, pp. 76–85.

[31] M. Kowalkiewicz, M. E. Orlowska, and T. Kaczmarek, “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*, 2006, vol. 3841, pp. 668–679.

[32] F. Johnson and S. Kumar Gupta, “Web Content Mining Techniques: A Survey,” *International Journal of Computer Applications*, vol. 47, no. 11, pp. 44–50, 2012.

[33] L. Huang, A. Sil, H. Ji, and R. Florian, “Improving Slot Filling Performance with Attentive Neural Networks on Dependency Structures,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2588–2597.

[34] R. Baumgartner, W. Gatterbauer, and G. Gottlob, “Web Data Extraction System,” in *Encyclopedia of Database Systems*, 2009, pp. 3465–3471.

[35] E. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner, “Web Data Extraction, Applications and Techniques,” *Knowledge-Based Systems*, vol. 70, no. November, pp. 301–323, 2014.

[36] D. Glez-Peña, A. Lourenço, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola, “Web scraping technologies in an API world,” *Briefings in Bioinformatics*, vol. 15, no. 5, pp. 788–797, 2013.

[37] S.C.M. de S Sirisuriya, “A Comparative Study on Web Scraping,” *8th International Research Conference KDU*, pp. 135–140, 2015.

[38] S. Srinivasa and P. C. P. Bhatt, “Introduction to web information retrieval: A user perspective,” *Resonance*, vol. 7, no. 6, pp. 27–38, 2002.

[39] R. Penman, T. Baldwin, and D. Martinez, “Web Scraping Made Simple with SiteScraper,” pp. 1–10, 2009.

[40] G. Kaur, “Usage of Regular Expressions in NLP,” *International Journal of Research in Engineering and Technology*, vol. 03, no. 01, pp. 168–174, 2014.

[41] S. K. Malik and S. Rizvi, “Information Extraction Using Web Usage Mining, Web Scrapping and Semantic Annotation,” in *Proceedings of the 2011 International Conference on Computational Intelligence and Communication Networks*, 2011, pp. 465–469.

[42] R. Gunawan, A. Rahmatulloh, I. Darmawan, and F. Firdaus, “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)*, 2019, pp. 283–287.

[43] J. Wang and F. H. Lochovsky, “Data extraction and label assignment for web databases,” *Proceedings of the 12th International Conference on World Wide Web, WWW 2003*, pp. 187–196, 2003.

[44] A. Sahuguet and F. Azavant, “Building light-weight wrappers for legacy web data-sources using W4F,” in *Proceedings of the 25th International Conference on Very Large Data Bases*, 1999, pp. 738–741.

[45] G. L. Hajba, *Website Scraping with Python using BeautifulSoup and Scrapy*, 2018.

[46] D. W. Otter, J. R. Medina, and J. K. Kalita, “A Survey of the Usages of Deep Learning for Natural Language Processing,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[47] R. Speck and A. C. Ngonga Ngomo, “Leopard — A baseline approach to attribute prediction and validation for knowledge graph population,” *Journal of Web Semantics*, vol. 55, pp. 102–107, 2019.

[48] D. Gerber, D. Esteves, J. Lehmann, L. Bühmann, R. Usbeck, A. C. Ngonga Ngomo, and R. Speck, "DeFacto - Temporal and multilingual deep fact validation," *Journal of Web Semantics*, vol. 35, pp. 85–101, 2015.

[49] D. Gerber and A.-C. Ngonga Ngomo, "Bootstrapping the Linked Data Web," *1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011*, 2011.

[50] K. Jing and J. Xu, "A Survey on Neural Network Language Models," 2019.

[51] S. F. Chen and J. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," in *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, 1996, pp. 310–318.

[52] Z. B. Wu, L. S. Hsu, and C. L. Tan, "A Survey on Statistical Approaches to Natural Language Processing," Tech. Rep., 1992.

[53] W. De Mulder, S. Bethard, and M. F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech and Language*, vol. 30, no. 1, pp. 61–98, 2015.

[54] F. Almeida and G. Xexéo, "Word Embeddings: A Survey," 2019.

[55] H. Adel, B. Roth, and H. Schütze, "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*, 2016, pp. 828–838.

[56] T. Segaran, *Programming Collective Intelligence*. O'Reilly Media, Inc., 2007.

[57] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning Data: Data Mining, Inference, and Prediction*. Springer, 2016.

[58] Y. Zuo, Q. Fang, S. Qian, X. Zhang, and C. Xu, "Representation Learning of Knowledge Graphs with Entity Attributes and Multimedia Descriptions," *2018 IEEE 4th International Conference on Multimedia Big Data, BigMM 2018*, pp. 2659–2665, 2018.

[59] A. M. Salgueiro, *Topónimos no espaço da CPLP: o Vocabulário Toponímico (MSc thesis)*, 2016.

[60] A. Mahani and A. Riad Baba Ali, "Classification Problem in Imbalanced Datasets," in *Recent Trends in Computational Intelligence*, A. Sadollah and T. S. Sinha, Eds. IntechOpen, 2020, ch. 4.

[61] N. Mamede, J. Baptista, C. Diniz, and V. Cabarrão, "STRING: An Hybrid Statistical and Rule-Based Natural Language Processing Chain for Portuguese," *PROPOR 2012 - 10th International Conference on Computational Processing of Portuguese, series Demo Session*, 2012.