Automatic Collection of Citations in the LdoD Archive

Duarte Nunes Félix de Oliveira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Prof. António Manuel Ferreira Rito da Silva
Prof. Manuel José de Freitas Portela

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Prof. António Manuel Ferreira Rito da Silva
Member of the Committee: Prof. Tiago Guerreiro

October 2018
Acknowledgments

In first place, I would like to thank my parents for always providing me the opportunity to pursue my dreams, for being comprehensive with my decisions, for financially supporting all my studies and extra curricular activities throughout these years and most of all for giving me all the freedom and perspectives to grow into the man I am today. I would also like to thank my grandparents, aunts, uncles and cousins for their teachings, encouragement, friendship and caring over all these years.

I would also like to acknowledge my dissertation supervisors Prof. António Rito Silva and Prof. Manuel Portela for their insight, support, feedback and sharing of knowledge that has made this Thesis possible, as well as for placing their trust in my abilities.

Last but not least, a special thank you to my girlfriend, friends and colleagues that helped me when I needed the most, without questioning and without whom it would be possible to be where I am today. Thank you.
Abstract

In this thesis we describe a way to transform an existing collaborative digital archive for Fernando Pessoa’s *Book of Disquiet* (LdoD) into one that is sensitive to Fernando Pessoa’s citations in social networks. We propose a new model for the collaborative digital archive that uses an algorithm for automatic collection of citations to add sensitivity to the archive such that it can reflect on the awareness of the references to the *Book of Disquiet* on social networks.

With this research we intend to collect citations from social networks (namely from Twitter) and explore how the *LdoD Archive* can be extended to incorporate those citations such that they are stored in the Archive and it is possible to access meta information related to the citation through the cited fragments.

This new *Social Media Aware LdoD Archive* supports a new set of features, such as: the collection of citations from Twitter, their transformation into annotations of the fragments belonging to a virtual edition, the visualization of all the meta information associated with a citation in the context of the cited fragment and the customization of the Social Aware virtual editions.

Keywords

Resumo

Nesta tese é descrita uma abordagem para transformar um arquivo digital colaborativo do Livro do Desassossego de Fernando Pessoa (LdoD) num arquivo que seja sensível a citações de Fernando Pessoa nas redes sociais. É proposto um novo modelo para o arquivo digital colaborativo que utiliza um algoritmo para uma recolha automática de citações com o objetivo de adicionar sensibilidade ao arquivo de modo a que este se torne sensível às referências ao Livro do Desassossego nas redes sociais.

Com este trabalho o nosso objetivo é recolher citações das redes sociais (nomeadamente do Twitter) e explorar de que maneira o Arquivo LdoD pode ser estendido de maneira a incorporar essas citações e a guardá-las no Arquivo para que depois seja possível aceder à meta informação associada a essas mesmas citações através dos fragmentos citados.

Este novo Social Media Aware LdoD Archive apresenta um conjunto de novas funcionalidades, tais como: recolha de citações do Twitter, a transformação dessas citações em anotações nos fragmentos que pertencem a determinada edição virtual, a visualização de toda a meta informação associada a uma citação no contexto do fragmento citado e a customização de SAVE.

Palavras Chave

Humanidades Digitais, Edição Digital, Redes Sociais, Sensibilidade, Arquivo LdoD, Livro do Desassossego, Recolha de Citações, Edições Virtuais
## Contents

1 Introduction ........................................... 1
   1.1 Motivation ......................................... 3
   1.2 Goals ............................................... 3
   1.3 Problem ............................................ 4
   1.4 Contributions ...................................... 4
   1.5 Organization of the Document ....................... 4

2 Related Work ........................................... 7
   2.1 LdoD archive ......................................... 9
   2.2 Web Data Mining ..................................... 11
   2.3 Information Retrieval ................................ 12
   2.4 Related Tools ....................................... 13
      2.4.1 Social Bus ...................................... 13
      2.4.2 Spring Social ................................... 14
      2.4.3 Twitter4j ....................................... 15
      2.4.4 Apache Lucene .................................. 16
      2.4.5 Summary and tools comparison .................. 17
   2.5 Related Projects ..................................... 17
      2.5.1 Social Impact project ........................... 17
         2.5.1.A Social Impact: Back-end (Database) .......... 19
         2.5.1.B Social Impact: Back-end (Social Bus) ........ 20
         2.5.1.C Social Impact: Back-end (Detection of citations) 21
         2.5.1.D Details and specifications about Social Impact parameters 21
      2.5.2 Summary and projects comparison ................ 22

3 SAVE (Socially Aware Virtual Editions) ................... 23
   3.1 Requirements ....................................... 25
   3.2 SAVE: the new model ................................ 26
      3.2.1 UML ............................................ 26
List of Figures

2.1 Dynamic and static aspects in the LdoD archive [1] .......................... 10
2.3 Architecture of an Information Retrieval system ............................ 12
2.4 Social Bus architecture [3] ................................................. 14
2.5 Social Impact architecture [3] ............................................. 18

3.1 New UML for the Social Media Aware LdoD Archive ....................... 26
3.2 Overall process of our solution .............................................. 27
3.3 First step - Phase 1 (blue entities) [4] .................................... 29
3.4 Second step - Phase 1 (blue entities) [4] ................................ 31
3.5 Third step - Phase 1 (blue entities) [4] .................................... 32
3.6 Fragment interpretation differences (first example) ......................... 34
3.7 Fragment interpretation differences (second example) ...................... 34
3.8 Fourth step - Phase 1 (blue entities) [4] .................................. 37
3.9 Fifth step - Phase 2 (blue entities) [4] .................................... 39
3.10 Sixth step - Phase 2 (blue entities) [4] ................................... 40
3.11 Front end additions (red boxes) to the management of a Virtual Edition ......................................................... 41
3.12 New component-and-connector view of LdoD Archive (reused most of the components from [2]) .................................................. 42
3.13 Module view - Decomposition of modules required for SAVE implementation .............................. 43
3.14 Tweet text that did not quote the complete sentence ....................... 45
3.15 Example of a complete highlight of a sentence after detecting an incomplete quote ...................... 46
3.16 Example of an incomplete highlight ..................................... 46

4.1 Simple search of a sentence in the LdoD Archive ............................ 54
4.2 Example of a tweet that originated a True Positive citation ................ 59
4.3 Annotation created by a true positive citation ............................................. 59
4.4 Example of a tweet that originated a False Positive citation .......................... 60
4.5 Example of a tweet that should have originated a citation (False Negative) ........ 60
4.6 Annotation that should have been created (red box) ...................................... 60
List of Tables

3.1 Meta information obtained from a Tweet .................................................. 30
3.2 InfoRange entity ......................................................................................... 33

4.1 Total number of tweets collected by keyword (retweets included in all of them) ........ 51
4.2 Number of citations and info ranges created for Lucene score > 20 ...................... 51
4.3 Number of citations and info ranges created for Lucene score > 30 ...................... 52
4.4 Number of citations and info ranges created for Lucene score > 40 ...................... 52
4.5 Number of annotations for each authorial edition depending on the Lucene threshold . . 53
4.6 Twitter4j precision according to keywords used for fetching tweets ................... 53
4.7 Lucene performance for Lucene score > 20 ............................................... 55
4.8 Lucene performance for Lucene score > 30 ............................................... 56
4.9 Lucene performance for Lucene score > 40 ............................................... 56
4.10 String matching algorithm performance for Lucene score > 20 and word window > 10 . . . 57
4.11 String matching algorithm performance for Lucene score > 30 and word window > 10 . . . 57
4.12 String matching algorithm performance for Lucene score > 40 and word window > 10 . . . 57
4.13 String matching algorithm performance for Lucene score > 20 and word window > 5 . . . 58
4.14 String matching algorithm performance for Lucene score > 30 and word window > 5 . . . 58
4.15 String matching algorithm performance for Lucene score > 40 and word window > 5 . . . 58
4.16 Number of citations that originated info ranges (word window > 5) ..................... 61
4.17 Number of citations that originated info ranges (word window > 10) ..................... 61
Introduction

Contents

1.1 Motivation ................................................. 3
1.2 Goals ..................................................... 3
1.3 Problem ................................................... 4
1.4 Contributions ............................................. 4
1.5 Organization of the Document ......................... 4
The LdoD collaborative archive\(^1\) is composed of Fernando Pessoa’s *The Book of Disquiet* (LdoD). The book comprises a set of manuscript, typescript and printed fragments, which have been transcribed and organized by experts to form four different editions according to the editors’ interpretations.

As described in [4], “the LdoD Archive explores the fragmentary and unfinished structure of Fernando Pessoa’s Book of Disquiet to foster a collaborative environment where the users can experiment with the creation of their own virtual editions [1, 2]. Besides representing four well-known experts’ editions of the book, the LdoD Archive enables the platform users to select fragments for inclusion in their virtual editions and, additionally, annotate and classify them according to a taxonomy. Both annotations and classifications can be applied to a fragment as a whole or to a part of it, for instance, a complete sentence or just a couple of words”.

### 1.1 Motivation

Currently, the LdoD collaborative archive contains two different types of editions: expert editions and virtual editions. The expert editions were proposed by scholarly authors (experts in Fernando Pessoa’s literature) such as, *Jacinto do Prado Coelho, Teresa Sobral Cunha, Richard Zenith* and *Jerónimo Pizarro*. On the other hand, virtual editions are created by non-experts who can provide a new perspective on the LdoD by deciding which fragments to include and which to exclude, their ordering, annotations and textual additions made by users.

However, the LdoD archive is not sensitive to external stimulus which makes it not able to reflect on the awareness of the references to the *Book of Disquiet* on social networks.

### 1.2 Goals

The main goal of this thesis is to transform the current LdoD archive into one that is sensitive to the *Book of Disquiet* citations in social media. To achieve this goal we intend to present a new algorithm for automatic collection of citations that supports a new sensitive model for the collaborative digital archive.

We intend to work on an existing algorithm, which collects citations from Fernando Pessoa’s work, and extend it with other aspects such as: time (weekly, monthly and annually citations) and logical relationships among the fragments that contain these citations. We propose a new model for the collaborative digital archive that uses an algorithm for automatic collection of citations to add sensitivity to the archive such that it can reflect on the awareness of the references to the *Book of Disquiet* on social networks. Therefore, the work to be done includes the integration of the information captured in the social networks with the LdoD model.

\(^1\)https://ldod.uc.pt/
As mentioned in [4], we intend “to explore how the LdoD Archive can be extended to incorporate those citations in social media such that they are stored in the Archive and that it is possible to access the sources of citation through the cited fragments. In some sense we intend to make the Archive sensitive to the social media”.

1.3 Problem

In the scope of LdoD archive, a virtual edition is made of fragments from other editions. Fragments selected for the virtual edition can also be enriched with annotations. Besides allowing its editor to choose which fragment should belong to the edition, it also allows them to define their order in the context of the virtual edition.

So far all editions were mainly human made. In this project we intend to define an automatic virtual edition which is sensitive to the external citations to the LdoD made in the social networks. Additionally, it should be able to reconfigure itself, whenever the set of citations change, and to keep track of the citations sources. We would like to introduce a new concept that would extend the existing virtual edition definition.

1.4 Contributions

The main contributions of this thesis are:

- Collecting citations from *The Book of Disquiet* in social networks over time
- Developing a substring pattern finding algorithm that finds a substring of a given pattern in a text
- Add sensitivity and awareness to LdoD Archive of external stimulus from social networks

1.5 Organization of the Document

This thesis is is organized as follows:

1. Introduction - in this chapter it is described the motivation for this thesis, the main goals, the problem statement and contributions of this thesis as well as the organization of the document.

2. Related Work - this chapter provides a context for this thesis and presents a literature review containing some of the work that was done in this field.

3. SAVE - this chapter describes the requirements and the solution for the problem. The solution is composed by two main phases (collecting information from social media and building social aware virtual editions) in addition to back-end and front-end implementation details.
4. Evaluation - in this chapter we present statistical results about the absolute number of citations and annotations created while varying different parameters and thresholds as well as performance levels of the different steps of our solution using precision and recall measures.

5. Conclusion - in this chapter we make an overview of the work done, the results obtained and contributions of this thesis.
Related Work

Contents

2.1 LdoD archive .................................................. 9
2.2 Web Data Mining ............................................. 11
2.3 Information Retrieval ....................................... 12
2.4 Related Tools .................................................. 13
2.5 Related Projects .............................................. 17
This chapter provides a context for this thesis starting by describing the current state of the LdoD archive: what it is, how it is organized, its functionalities and entities related. Afterwards, some tools, concepts and related projects are presented, completing, in this way, the literature review.

2.1 LdoD archive

As mentioned in section 1.1, LdoD contains two different types of editions (expert and virtual editions). Virtual editors are users that by interacting with LdoD archive can create their own interpretation of Pessoa's LdoD by selecting, ordering and annotating other authors' interpretations.

As described in [1], the archive also contains three related dimensions: "a genetic dimension that contains a narrative of authorial composition; a social dimension that contains a narrative of scholarly editing; and a virtual dimension that allows users to explore the possibilities of both writing and editing while interacting with The Book of Disquiet".

The expert editions are built on genetic dimension while virtual editions are built on genetic, social or virtual dimensions. This means that virtual editions are built on the original set of expert editions and on other virtual editions.

- Genetic dimension: genetic transcriptions of Pessoa
- Social dimension: the four expert editions
- Virtual dimension: customized virtual editions

These three dimensions can coexist because they are supported by a Web 2.0 environment which provides a more interactive and dynamic web experience where users can collaborate and share information on their virtual/social editions. In LdoD archive the Web 2.0 environment integrates an object-oriented database with a file repository for TEI-encoded files.

Figure 2.2 illustrates some of the main entities involved in the concept of "Virtual editions". In the scope of LdoD archive, a virtual edition is made of fragments from other editions and its fragment interpretations contain annotations in the text made by users. Each virtual edition has an associated taxonomy and a taxonomy aggregates several categories. As described in [2], "tagging represents the action of applying a category to a part of a fragment or to the whole fragment. Therefore, the same category can be applied to different fragments or to different parts of the same fragments". While interacting with a virtual fragment interpretation, a user can create an annotation with several tags or without specifying any tag, in which case it only contains a comment.
Figure 2.1: Dynamic and static aspects in the LdoD archive [1]

Figure 2.2: Current UML model for virtual editions in the LdoD archive [2]
2.2 Web Data Mining

The main goal of web data mining is to find and select the appropriate data and knowledge in the web (where most of the information is heterogeneous and usually not very well structured). It is important to take into account that this information can appear in different layouts, contexts and formats which makes it difficult to analyze and integrate with the existing knowledge in a given project or problem.

Web mining consists in discovering patterns and knowledge from web pages content, structured hyperlinks or usability data from web pages as described in [3]. Based on [3], [5], [6] and [7] web data mining is classified into three different categories depending on the data analyzed:

- Web content mining: consists in collecting data or knowledge from web pages content. This type of web data mining is the most similar to what happens in the traditional data mining where patterns and knowledge is found in a given data base or text.

- Web structure mining: consists in collecting data or knowledge from structured hyperlinks that represent the structure of the web. This type of web data mining is particularly useful for search engines where it is important to discover specific web pages by given links.

- Web-usage mining: consists in discovering user access patterns based on access logs that save each click made by a user.

Since the information in the web is sparse, dynamic and error prone it is important, specially for web content mining, the use of crawlers in a big number of web pages. In this project we intend to focus mainly in web content mining, since we are collecting information, patterns and knowledge from specific and well know social networks.

Web content mining consists in automatically collecting texts, images, audio, videos, metadata and hyperlinks from a given web page.

As described in [3] and [6] there are three different types of data present in a web page:

- Structured data (eg. entries in a table or in a relational database)

- Semi structured data (it is not present in a table or in a relational database, however it contains tags or markers that separate semantic elements and hierarchies, eg. HTML and XML documents)

- Unstructured data (it is the most common type of data in web pages and it is not present in a predefined data structure, eg. free text, facts or dates)
2.3 Information Retrieval

Information retrieval consists of identifying which documents contain relevant information in a given set of documents depending on what we are looking for. Then, a ranking is produced based on the relevance of each document depending on the search query. This search query contains the keywords (also designated as "terms") that we are interested in. An Information Retrieval System is split into two main phases: document indexing and search for the indexed documents (figure 2.3).

As described in [3], when a user makes a query to an Information Retrieval System the retrieval system module uses a document indexer to select the relevant documents from the document collection. Afterwards, the retrieval system module ranks these documents based on their relevance. It is also possible to improve the document ranking system based on the user's feedback. The four steps in an Information Retrieval System (Pre-process, Models, Indexation and Evaluation) are briefly described below.

Pre-processing the documents consists of removing stopwords (e.g. prepositions, conjunctions, quantifiers and determiners). As described in [5], stopwords are words that occur frequently in a text but are not relevant for the query.

An information retrieval model is what dictates the importance of a given document in a collection. There are four main models: Boolean Model, Vector Space Model, Language Model and Probabilistic Model; but the most used are the first two.

---

1. https://www.slideshare.net/kaninoshiiu/tdm-information-retrieval
In the **Boolean Model** a document is considered to be relevant if the boolean logic of the query is *True*, i.e. if the combination of terms in the query satisfies the condition imposed. The terms are combined using logical terms (AND, OR and NOT). The main advantage of this model is its simplicity, however there is no partial correspondence of terms neither a ranking of documents.

In the **Vector Space Model** a document is represented as a vector of terms. It is used the TF-IDF formula to calculate the relevance of a term in a give collection of documents. TF (Term Frequency) refers to the frequency of a term in a given document. IDF (Inverse Term Frequency) is important because it privileges the weight of terms that occur rarely in a collection. TF-IDF of a term consists in multiplying these two values.

The big difference between these modules are that **Vector Space Models** is used to produce a ranking of documents instead of classifying a document as relevant or not relevant as in the **Boolean Model**. To avoid the problem of document size inherent to TF calculation it is used the *cosine-similarity*.

Indexation is an important step in order to produce efficient results about the terms present in a document. As described in [3] and [8], the most powerful method is the *inverted index* because it is possible to find the documents where a certain term is present in constant time. It assigns to each term of a given vocabulary a list with every document that contains that term.

To evaluate the quality of an Information Retrieval System it is often calculated the *precision* and the *recall*.

\[
Precision = \frac{\#(relevant \ documents \ obtained)}{\#(documents \ obtained)}
\]

\[
Recall = \frac{\#(relevant \ documents \ obtained)}{\#(relevant \ documents)}
\]

### 2.4 Related Tools

In this section we describe four different tools which had the goal of collecting tweets and posts from social media based on keywords. We described how Social Bus and TwitterEcho worked (Section 2.4.1) as well as Spring Social (Section 2.4.2) and Twitter4j (Section 2.4.3).

#### 2.4.1 Social Bus

Social Bus\(^2\) is a tool that is capable of collecting information from social networks by using *Twitter* and *Facebook’s APIs*\(^3\). It is used to monitor and consume tweets from *Twitter* and to search posts on

\(^2\)http://reaction.fe.up.pt/socialbus/
\(^3\)https://developer.twitter.com/
https://developers.facebook.com/
Facebook based on a given list of terms (keywords).

This tool is an extension of the TwitterEcho project described in [9] and [10]. TwitterEcho is a system that collects tweets from the portuguese community on Twitter: by giving it a keyword this tool returns all the citations that contain this keyword. Social Bus was an improvement to the TwitterEcho project because it is capable of collecting information from Facebook too. Another important improvement was the ability to collect this information over time by using crawlers (a thing that was difficult to achieve due to the fact that the number of requests to the APIs were limited over a certain period of time).

"When a crawler receives a tweet it sends the message to SocialBus Server which is responsible for processing, extracting metadata, indexing, tokenizing and other computations.”

![Diagram of Social Bus architecture](http://reaction.fe.up.pt/socialbus/)

**Figure 2.4: Social Bus architecture [3]**

### 2.4.2 Spring Social

Spring Social⁵ is an extension to the open source Spring Framework⁶ (which is an application framework for the Java platform). Spring Social is mainly composed by different projects such as: Spring Social Core, Spring Social Twitter, Spring Social Facebook and Spring Social Linkedin and allows users

---

⁴[http://reaction.fe.up.pt/socialbus/](http://reaction.fe.up.pt/socialbus/)
⁵[https://projects.spring.io/spring-social/](https://projects.spring.io/spring-social/)
⁶[https://spring.io/](https://spring.io/)

14
to connect their applications with Twitter, Facebook and Linkedin APIs, respectively.

**Spring Social** main features 7:

– A “Connect Framework” (service provider framework) that handles the core authorization and connection flow with service providers

– A “Connect Controller” that handles the authorization flow between a Java/Spring web application (service consumer), a service provider, and the users

– A “Signing Controller” that allows users to authenticate with the application by signing in with their provider accounts, such as their Twitter or Facebook accounts

**Spring Social Twitter** - relevant search operations 8:

– search(SearchParameters searchParameters) - searches Twitter, returning a set of results. Supports either user or application authorization

– search(String query) - searches Twitter, returning the first 50 matching Tweets. Supports either user or application authorization

– search(String query, int pageSize) - searches Twitter, returning a specific page out of the complete set of results

– search(String query, int pageSize, long sinceId, long maxId) - searches Twitter, returning a specific page out of the complete set of results

*SearchParameters* include an enumeration of Twitter search parameters that can be specified beforehand while searching for tweets. This includes: the language, geographic location, the query itself, the “until” date and the “max” and “since” IDs. Each tweet has a tweet ID which is a Java long. These two IDs correspond to the maximum and minimum ID we want to retrieve, respectively. Most recent IDs correspond to higher IDs.

### 2.4.3 Twitter4j

The original plan was to use Spring Social library to collect tweets based on keywords. Our main motivation to use this library was due to the fact that Spring Social was an extension of the Spring Framework (which was already being used in the LdoD Archive project). Another advantage of this library was the amount of support and documentation available as well as the Search API operations.

---

7https://docs.spring.io/spring-social/docs/1.0.x/reference/html/overview.html
8https://docs.spring.io/spring-social-twitter/docs/1.1.0.RELEASE/apidocs/
However, some weeks after working with this library we noticed that the Search API from Spring Social did not do everything we needed (at least directly), i.e. the mapping between the Spring Social Search API methods and the methods from Twitter Search API\(^9\) was not perfect and complete. For example, Spring Social Search API does not possess an individual method that corresponds to an individual request to the Twitter Search API. The mapping is not 1 to 1.

On the other hand, we found out that Twitter4j library\(^10\) had a more direct and complete mapping to the Twitter Search API, therefore facilitating the search for the several tweet fields. Nonetheless, the most important feature that made use change from Spring Social to Twitter4j was the fact that it was not possible (through the API methods available) to obtain complete tweet texts if a tweet had more than the original 140 characters. In 2017, Twitter doubled the maximum number of characters to 280\(^11\) and Spring Social only returns the original 140 characters, truncating the rest of the text. Fortunately, Twitter4j allows us to check if a tweet is truncated or not and if it is we can collect the complete text.

Additionally, Twitter’s search index only contains about 7 to 10 days worth of tweets\(^12\). “There are rate limits in the Twitter API. You cannot call a given Twitter API endpoint more than 180 times per 15 minutes.”\(^13\)

### 2.4.4 Apache Lucene

Apache Lucene is an information retrieval system that uses a document indexer API written in Java.\(^14\) This tool was used in the Social Impact project\(^3\) to build a mapping system between the information collected from social networks and all Fernando Pessoa’s work.

As described in\(^3\) and\(^11\), this tool was used based on its efficiency, reliability and implementation features, such as:

- Ranking based search (better results in the first place)
- Different types of search (it is possible to search for exact terms or combinations of these terms using operators AND, OR and NOT)
- Search for fields (it is possible to search for specific fields or group them as a whole)
- It allows for search and updates of information simultaneously

Apache Lucene allows a user to create his own queries through its API\(^15\). One interesting feature that Apache Lucene provides is Fuzzy Searches which consists in finding strings that match a pattern

---

10[http://twitter4j.org/javadoc/twitter4j/Status.html](http://twitter4j.org/javadoc/twitter4j/Status.html)
14[http://lucene.apache.org/core/](http://lucene.apache.org/core/)
approximately based on the Edit Distance algorithm (which uses the Levenshtein Distance as a metric for measuring the difference between two strings).

2.4.5 Summary and tools comparison

Social Bus (Section 2.4.1) would be an interesting tool to use since it was able to collect tweets and posts based on keywords, but unfortunately both Twitter and Facebook’s API had changed (because of privacy aspects). On the other hand, libraries such as Spring Social (Section 2.4.2) or Twitter4j (Section 2.4.3) are updated and present a full range of Java methods that can be directly invoked and easily integrated with the LdoD Archive.

Due to the fact that we are dealing with different versions of the Portuguese spelling agreement, when comparing the LdoD with posts made nowadays we have to be cautious about some words that were modified with this 1990 agreement.

The list of terms specified in Section 2.5.1.D is useful, however in our solution it would only make sense to include the terms “Fernando Pessoa” and “Bernardo Soares” from that list (since we are dealing with citations only from The Book of Disquiet).

Additionally we explain how a typical information retrieval system works and we give the example of Apache Lucene (Section 2.4.4). Lucene (Section 2.4.4) is the appropriate information retrieval tool for indexing and searching documents since the fragments in the LdoD Archive are already indexed by this library. Hence facilitating the search of fragments using queries.

2.5 Related Projects

In this section we describe three different projects which had the goal of collecting tweets and posts from social media based on keywords. We describe how Social Bus and TwitterEcho work as well as a more detailed description of the Social Impact project (2.5.1).

2.5.1 Social Impact project

The main goal of Social Impact [3] was to create an abstraction for a previous project called “O Mundo em Pessoa” which was initially built to collect citations from Social Networks and then map them with Pessoa’s original work. To complete those goals “Social Impact” used Apache Lucene and “Social Bus” tools. “Social Impact” is an abstraction because one of the goals was to create an extensible architecture to include several projects besides “O Mundo em Pessoa” such as, “Lusica”, “Onde há Bola” and “Missinks”.

17
“O Mundo em Pessoa” was already able to collect citations, map them with Pessoa’s work and show them in a proper interface (with the help of SAPO Labs in the construction of web applications). This project is a service oriented web system and it is composed by three fundamental layers: front-end, back-end and web services (described below).

There was a need to improve “O Mundo em Pessoa” project because it presented some problems concerning consistency (duplicate information of Pessoa’s work), the efficiency of queries to the database (because indexes and foreign keys were not used) and limitations about the string matching algorithm that produced some false positives and false negatives. Other than that, “Social Impact” architecture reused most of the components from “O Mundo em Pessoa” project.

Social Impact front-end interface was developed by designers and web-developers of SAPO Labs and it presents: the messages published in social networks, the information about the person who posted it and links to this person’s profile and to the corresponding work by Pessoa. The most interesting features are: the publications over certain periods of time (daily, weekly and monthly), the most cited works and their respective authors (heteronyms or ortonym).

Figure 2.5: Social Impact architecture [3]
Web services are used as an intermediary/broker that makes the integration between Front-end and Back-end. Their main motivation for using web services is due to the fact that it allows them to achieve interoperability between systems and to extend this architecture to other parallel projects such as "Lusica". The main goal is to query the database based on the parameters provided in Front-end. In "Social Impact" project these web services were developed in PHP and based on REST architecture. The response of these web services comes in JSON format.

Back-end (figure 2.5 and 2.6) is the layer responsible for: collecting data ("Social Bus" component and list of terms), processing it (detection of citations based on an information retrieval system) and storing it in a database together with the input initially provided. The database is a simple relational database in MySQL that stores: Pessoa's work, a list of terms to be searched in social networks and citations collected from social networks. The list of terms is a crucial aspect of the system because it contains the terms that are searched in social network posts: e.g. author's name, titles or quotes. After the back-end computations, an output is produced and saved in the database. This output can also be consulted using URL queries and the web service returns it in JSON format.

Crawlers and wrappers were used to search for Pessoa's work (one iteration only) and for citations in social networks (several times). Then all this information was stored in the database. They used Twitter and Facebook APIs to collect citations, more specifically the search method that given a term it returns the most recent citations where that term is present. However, these APIs have a limit for the number of requests per hour (Twitter: 150 and Facebook: 200), so in order to have this method running in a cycle it was required to do a sleep between requests.

2.5.1.A Social Impact: Back-end (Database)

"Social Impact" database is a relational database made in MySQL which contains five tables:

- Entity: it stores information about the author of a certain work
- Work: it stores information about all the works (provided by the administrator module)
- Post: it is the core table of this database. It stores all the information about the collected posts. It contains two important fields - "work_cod" (NULL by default) and "deleted" (true or false). They represent the correspondent work and if the post is a citation or not, respectively.
- User: it stores the information about the user who posted a certain message
- Matches: it stores statistics about the mapping between a certain post and the matched works
- Term: it stores the different terms present in a given post

16http://arquivopessoa.net
http://casafernandopessoa.cm-lisboa.pt
2.5.1.B *Social Impact*: Back-end (Social Bus)

The main goal of “Social Bus” is to collect the posts from social networks based on a list of terms provided. In the first place, “Social Bus” requires an access key to Twitter and Facebook APIs. This key contains several tokens that are necessary to each function of social networks. These tokens are stored in a CSV file.

Then, “Social Bus” starts by collecting posts that contain at least one term from the list of terms, which are then saved in a temporary file along with information about the date and the user. This file is in a folders tree with three levels: year, month and day. That file’s name is the hour. Each social network has its own consumer, therefore they work as two independent programs each with its own tree folders.
2.5.1.C  **Social Impact: Back-end (Detection of citations)**

This component is one of the main contributions of [3]. The goal of this component is to identify the correspondent work/text for a given post collected by Social Bus. The collected posts are saved in two different files, one for Twitter and other for Facebook. These two files are updated independently, therefore two threads were created, which were responsible for processing the posts from these two files and then saving their posts in the database. This component uses Apache Lucene tool and it is divided in two phases (indexing Pessoa’s works and mapping posts to their respective works).

In the first phase (indexing Pessoa’s work) it is made a SQL query to the database in order to retrieve every work and then add their contents to a Lucene’s class named Documents. The next step consists of splitting each work’s content in terms and finally filtering them by removing stopwords (present in a text file). After this it is possible to produce an inverted index where terms are the “keys” and the “values” are the works (documents) where they can be found. Each work is identified by a unique ID. This phase occurs only once (if works remain unchanged).

On the other hand, the second phase (mapping posts to works) is constantly being executed. The collected posts (present in Twitter and Facebook files) also have to go through a filtration process to remove stopwords. In addition, it is also required to filter these posts by removing badwords (words that are definitely not present in Pessoa’s work). Then, these filtered posts are split in their terms and queries are produced where these terms are separated by ANDs. The next step consists in providing a given query to the searcher method (from Apache Lucene) which is responsible for collecting the indexed works from Phase 1 and for each work is calculated a score based on their similarity to this query. Finally, a list with the rankings of each work is returned and the work that possesses the biggest rank is assigned as the correspondent work.

In the table “Post” from the database, the field “cod_obra” is the work with highest rank. The field “apagado” is True if the rank is lower than a previously defined threshold, otherwise it is False. Depending on the threshold the number of false positives and true negatives may vary, thus influencing precision and recall.

2.5.1.D  **Details and specifications about Social Impact parameters**

The list of terms is composed by the terms: “Fernando Pessoa”, “fernandopessoa”, “Ricardo Reis”, “ricardoreis”, “Alberto Caeiro”, “albertoecaiero”, “Álvaro de Campos”, “alvarodecampos”, “Bernardo Soares” e “bernardosoares”. This is based on a previous empirically analysis that was made during “O Mundo em Pessoa” project.

"Social Bus" collected 56212 posts from January 2014 until June 2014 (6 months). They defined the acceptable threshold to be 1.0 in order to satisfy SAPO Labs requirement about having rigorous treatment in the detection of citations because they were going to be presented to the public afterwards.
in their site. Only 4720 posts were considered in fact to be Pessoa’s citations.

2.5.2 Summary and projects comparison

After analyzing Social Impact project we conclude that it would be interesting to collect information from social media using keywords. However, as mentioned in Section 2.4.5 it would make sense to define our own set of keywords since we are dealing with *The Book of Disquiet* instead of all Pessoa’s work. Social Bus would be an interesting tool to use for this task, but we felt the need to develop our own module responsible for collecting tweets and posts over time since Twitter and Facebook’s API has changed.

In terms of citation detection, we conclude that Lucene would be the most appropriate information retrieval system to use because it originated precise results in the Social Impact project and it is already being used in the LdoD Archive for indexing and searching fragments.

Additionally, after analyzing Social Impact project we also felt the need to develop our own substring matching algorithm in order to increase recall values that were only 59%. This value was obtained while using Lucene as the main responsible for citation detection. Our main motivation is to reduce the number of false negatives therefore increasing the recall. For that we need to develop a more flexible substring matching algorithm to even find substrings of what people write that can be LdoD citations. This flexibility is important because users usually write additional information in the beginning or end of their tweets and posts. Besides that, users frequently misspell some words.
SAVE (Socially Aware Virtual Editions)

Contents

3.1 Requirements .................................................. 25
3.2 SAVE: the new model ........................................... 26
3.3 Phase 1: Get information from social media .................. 28
3.4 Phase 2: Build SAVE ............................................ 39
3.5 Architecture ..................................................... 41
3.6 Problems faced .................................................. 45
3.7 Summary ........................................................ 47
As mentioned in section 1.3, all the annotations in virtual editions are human made. The goal of this solution is to enrich the current LdoD Archive with virtual editions that are sensitive about *The Book of Disquiet* on social media.

It means that a new enriched type of virtual edition will be introduced, which is going to extend the current model. This new virtual edition is called *Socially Aware Virtual Edition*, seeing that this new virtual edition is aware of the stimulus from social media based on LdoD citations.

For that, it was important to define the functional requirements in first place (section 3.1).

Then we describe two main phases that constitute this solution. Phase 1 describes how the information was collected from social media whereas phase 2 describes the process of creating socially aware virtual editions with aware–annotations based on the citations collected in the previous phase.

Afterwards, we present two architectural perspectives of the system highlighting the main additions and changes done to the LdoD Archive model.

In the end there is a brief description about the problems faced explaining some of the different paths that were taken compared to the initially proposed solution and as well as a summary.

### 3.1 Requirements

Functional Requirements describe the functionalities of the system, i.e. what the system in fact does. These are the main functionalities of LdoD enrichment we intend to do:

- Collect citations from social networks: automatic collection over time from tweets or posts that cite LdoD
- Mapping the collected citations with LdoD fragments
- Maintain a set of meta information associated with each one of the citations
- Dynamically define virtual editions which reconfigure themselves whenever the set of citations change
- Construct a way for users to interact with this new concept of socially aware virtual edition (SAVE) based on a set of criteria

The way a user interacts with his own "personalized" SAVE versions is by choosing the set of criteria to be considered at the moment of creation or even latter.

After applying these criteria the fragments present in a personalized SAVE are modified in terms of annotations, which are added or removed automatically.
3.2 SAVE: the new model

3.2.1 UML

In Section 2.1 it was presented a figure that illustrated how the UML of the LdoD Archive was when we first started. In this section we present the new UML for the Social Media Aware LdoD Archive (Figure 3.1). This domain model represents how the actual LdoD Archive model was extended to support the new set of features. In blue are the entities and relationships added to the previous model [2].

The Citation class is the core entity of this new module. As described in [4], this class “represents the citations from LdoD that were identified in social media. Currently, the Archive only collects citations from Twitter, therefore, there is a single subclass of Citation, TwitterCitation. This class is associated to a Twitter class, which contains all the data collected with the tweet: source link (URL), date, tweet text, tweet ID, location, country, username, user profile URL, user image URL and the original tweet ID (in case we are dealing with a retweet instead of the original tweet”).

It is important to refer that some instances of Tweet may not be associated to a TwitterCitation instance (represented by 0..1 multiplicity in the diagram). This happens when a tweet that was collected is not a citation of The Book of Disquiet. On the other hand, when a tweet that was collected is in fact a citation of The Book of Disquiet, a twitter citation instance is created and associated to that tweet (the multiplicity is represented by * because the tweet that originated the twitter citation might have been retweeted).
Afterwards, citations are associated to the correspondent fragment and later, (indirectly) associated to a particular transcription of a fragment, the one that is used in the virtual edition.

“This was achieved by extending the former Annotation class into two subclasses, HumanAnnotation and AwareAnnotation, where the former represents the previous annotation made manually by the users, while the latter is the automatically generated social aware annotation. Since citations correspond to parts of a fragment, for instance a sentence, the aware annotations also applies to the part of the fragment that is cited, class Range, and the system is able to automatically identify it, class InfoRange” [4].

Class HumanAnnotation embodies the behaviour of the original class Annotation (see Figure 2.2). Both these two new sub classes of Annotation are associated to a class Range. The class Range contains the start and end positions of the quote in the fragment text. Additionally, we created a new auxiliary class InfoRange that contains all the necessary information in order to create an aware annotation (see Table 3.2). This information includes: start and end positions of the quote in the fragment text, the quote itself and the meta information from the tweet that we want to present to the user in the aware annotation. The main motivation for creating this auxiliary entity was due to performance reasons (explained in detail in Section 3.3.4).

Finally, as mentioned in [4]: “in order to enrich the virtual editions with the new set of features, a set of social media criteria, represented by the class SocialMediaCriteria, can be added to a virtual edition”. These criteria are interpreted by the new Social Aware Archive and then it generates aware annotations according to the conditions specified by the criteria. For example, if the GeographicLocation of a virtual edition specifies a certain country or set of countries, only citations done in that country or set of countries are considered for the creation of aware annotations in that virtual edition.

3.2.2 The process

![Figure 3.2: Overall process of our solution](image-url)
In the previous section we described the new entities that were created. In this section we briefly describe the overall process of our solution that collects citations from Twitter, classifies them and, finally, generates social aware annotations based on social media criteria (Figure 3.2) [4].

This process is divided into six fundamental steps described in detail in Sections 3.3 and 3.4. Phase 1 includes the first four steps and Phase 2 includes the fifth and sixth steps. Phase 1 consists in getting information from social media (in particular, from Twitter) and classifying every tweet as being a citation of the book or not. Phase 2 consists in generating socially aware annotations based on the citations identified on Phase 1 and presenting them to the user according to the set of social media criteria defined.

3.3 Phase 1: Get information from social media

Since we already have the book fragments available, this phase consists in collecting tweets and posts based on specific keywords related to specific citations from texts of *The Book of Disquiet* and then map them to the correspondent fragments using indexes from Lucene. The specific set of keywords that are used for collecting tweets are: "Livro do Desassossego", "Fernando Pessoa", "Bernardo Soares" and "Vicente Guedes".

This set is based on the list of terms used in Section 2.5.1 but in our project terms are oriented to *The Book of Disquiet*. This book was written by *Fernando Pessoa* but the publication was credit to the heteronyms *Bernardo Soares* and *Vicente Guedes*. It would also be interesting to collect tweets that mention "The Book of Disquiet", but since the fragments are written in Portuguese it would be complicated to find matches between what was written in the tweets and what was written in the fragments present in the LdD Archive.

Initially, the goal was to use *Social Bus* (Section 2.4.1) tool to collect either tweets and posts from Twitter and Facebook, respectively. But later we realized that both Twitter and Facebook’s API had changed, mainly in what concerns privacy aspects. Then, the following option was to use *Spring Social* library (Section 2.4.2). But in the end we used *Twitter4j* library because of the problems explained in Section 2.4.3.

3.3.1 Twitter - Twitter4j

Figure 3.3 illustrates, in blue, the first step (1) of Phase 1. In first place we are using the *Twitter4j* library to collect *The Book of Disquiet* citations from Twitter during certain periods of time using the *Search API* operations described in Section 2.4.3. We are collecting tweets which contain at least one term from the list of terms specified in Section 3.3. The module responsible for collecting tweets was based on the
Every time the module responsible for collecting tweets runs, the maximum number of collected tweets per page is 100 and the maximum number of pages that can be obtained are 180. Each query corresponds to one retrieved page.

Each tweet has a tweet ID which is a Java long. The most recent tweet has the higher tweet ID. As described in SocialSeer.com, the process consists in collecting 100 tweets at the time while maintaining a reference to the lowest ID retrieved until then. Since we still have 179 calls to make we need to know (before each call) the lowest ID retrieved in order to decide whether collect another 100 tweets.

This process is repeated until all the 180 calls are made or (most likely) until there are no tweets in the result set. It is important to remember that Twitter’s search index only contains about 7 to 10 days worth of tweets.

As explained in Section 2.4.3, you cannot call a given Twitter API endpoint more than 180 times during 15 minutes. These are the rate limits that control how fast one makes the calls. In order to get around this limitation it is done a `Thread.sleep()` to delay this process for some seconds. The seconds we need to sleep are given by the method `getSecondsUntilReset()` from class `twitter4j.RateLimitStatus`.

Another important aspect is that we only collect tweets written in Portuguese because the fragments present in LdDoD Archive are also in Portuguese. However, this does not exclude the possibility of collecting tweets from all over the world (which is an interesting feature discussed in section 3.4).

Tweets were collected (approximately) every each week and saved into JSON files. The module responsible for collecting tweets stores them into into four different JSON files for each of the four keywords described in Section 3.3 with the current date appended. For example, on March 16th were created four different files: `twitter-bernardo-2018-03-16.json`, `twitter-fp-2018-03-16.json`, `twitter-livro-2018-03-16.json`.  

---

1[^1]


2[^2]

[^2]: https://www.socialseer.com/aboutcontact/
Each line in a JSON file corresponds to a tweet collected. The meta information that we collect for each tweet is exhibited on Table 3.1.

<table>
<thead>
<tr>
<th>Tweet String sourceLink</th>
<th>String date</th>
<th>String tweetText</th>
</tr>
</thead>
<tbody>
<tr>
<td>long tweetID</td>
<td>String location</td>
<td>String country</td>
</tr>
<tr>
<td>String username</td>
<td>String userProfileURL</td>
<td>String userImageURL</td>
</tr>
<tr>
<td>long originalTweetID</td>
<td>boolean isRetweet</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Meta information obtained from a Tweet

The difference between location and country is that location represents the current location (e.g. the city or neighbourhood) of the user, if available. This means that this value can be most of the times null if not provided by the user. On the other hand, country represents the country where the tweet was written, if available. This also means that this value can be most of the times null. These two fields are represented by Strings. When the values returned by the API are null we decided to assign the value “unknown” to these fields. Unfortunately, most of the times users do not provide these information in their tweets. Besides that, although country is a thrust worthy value we cannot say the same thing about location because it can be inserted by the user, including wrong locations or even made-up words. Hence, we decided to use only the information given by the country.

The original tweet ID field only applies to retweets. This field returns the tweet ID of the original tweet that was retweeted.

We started collecting tweets on March 16th, but only after May 14th we started collecting tweets that contained meta information about being retweets or not, concretely their originalTweetID and isRetweet information. This means that during the first two months we did not make any distinctions between tweets and retweets. After that period, these two concepts were treated differently as explained below on Section 3.3.3.

3.3.2 Facebook - Facebook4j

As mentioned in Section 2.5.1, Social Impact project used to invoke Facebook’s API using Social Bus in order to search for public posts based on keywords (similar to the process used for Twitter).

Similar to what we did for Twitter (using Twitter4j library) we tried to do the same for Facebook (using Facebook4j library). However it was not possible to search for public posts based on keywords on Facebook anymore. Since Facebook’s API changed from version v1.0 to v2 permanently in 2015 it is not possible to query public posts anymore. Nowadays, Facebook has a restrict access to his Public Feed API which is limited to a set of media publishers and its usage requires prior approval. It is not possible to apply to use this API at this time 3.

3https://developers.facebook.com/docs/public_feed
However, there is here a potential future work opportunity (see Section 5.2). In spite of not being possible to search for public posts using Facebook’s API, it is possible, for example, to search for posts in a given Facebook page. I.e. using Facebook4j we can retrieve posts from facebook pages such as: Livro do Desassossego and Livro do Desassossego - Fernando Pessoa.

3.3.3 Citation Detection - Apache Lucene

Currently, in LdoD Archive it is implemented a Lucene Index with fragments present in LdoD Archive. Similarly to the process described in Section 2.5.1 every fragment content is in a Lucene’s class named Document. This class is the basic unit and its content is represented by a group of fields. A Document represents a fragment interpretation.

As mentioned in [4]: “in the second step (figure 3.4), twitter citations are created from the tweets previously collected. To do so, the content of the tweets in the JSON files is compared to the inverted index of fragments produced by Lucene and, depending on a threshold, a twitter citation is created if the content is similar enough (using the Boolean Model and Vector Space Model together with normalization methods [5])”.

In the beginning, we set this threshold to 1.0 based on what was done on Social Impact project, specified in Section 2.5.1.D. Afterwards, we empirically noticed that for several collected tweets this threshold was to low, i.e. there was a lot of false positives (tweets that Lucene considered as being citations but in fact they were not). The solution was simple: increase the threshold.

After empirically observing Lucene’s behaviour after this change we came to the conclusion that a value of 30.0 was the ideal for the set of tweets that we tested. Although this value was not high enough to prevent occasional false positives, we noticed that if it was higher the number of false negatives would increase, resulting in a loss of potential LdoD citations. False negatives are tweets that Lucene considered as not being citations but are in fact LdoD citations.
In Section 3.3.4 we explain how the false positives were basically eliminated, remaining only true positives (tweets that Lucene classifies as citations and are indeed LdoD citations). In Chapter 4 we describe how Lucene performance is tested for different thresholds.

Concretely, we initiate the mapping step by providing a query (i.e. the tweet text of a given tweet present in a JSON file) to the IndexSearcher which returns a list of ranked candidate documents/fragments. The document with the biggest rank shall be considered as the cited fragment. Therefore, creating a twitter citation instance which is associated to its correspondent fragment.

We only analyze original tweets. This means that in this step we skip retweets when creating twitter citations. Only ahead we take into account the number of retweets for counting purposes.

The information present in class TwitterCitation is also illustrated in Table 3.1. As described in [4], “the Citation class represents the citations from LdoD that were identified in social media. Currently, the Archive only collects citations from Twitter, therefore, there is a single subclass of Citation, TwitterCitation. This class is associated with a Twitter class, which contains all the data collected with the tweet: source link (URL), date, tweet text, tweet ID, location, country, username, user profile URL, user image URL and the original tweet ID (in case we are dealing with a retweet instead of the original tweet)”.

### 3.3.4 Identify ranges - Jaro Winkler

![Diagram of the mapping process](image)

Figure 3.5: Third step - Phase 1 (blue entities) [4]

After using Lucene to find the correspondent fragments to the given tweets we noticed that some of the tweets were incorrectly associated to LdoD fragments (false positives). Lucene classified some of the tweets as being LdoD citations but in fact they were not. This happened because for Lucene some of these tweets were similar enough to the texts compared. To reduce and even eliminate these false positives we run a string matching algorithm in this third step in order to find an exact match between tweet texts and the their associated fragments. Aside from that, Lucene does not also provide start and end positions for a quote found in a text. This means that the goal of the previous step (Section 3.3.3)
was to find a potential candidate to be the fragment where a certain tweet text occurs. The goal of this step is to find precisely where this quote occurs if it even occurs.

In the beginning we thought about using a string matching algorithm such as Boyer-Moore string-search algorithm [12]. But later we realized that we did not want to find exact matches between what people write in their tweets and what is written in LdoD. We want to be rigorous but also tolerant enough to even find substrings of what people write that can be LdoD citations. This flexibility is crucial because Twitter users usually write additional information in the beginning or end of their tweets. Besides that, Twitter users frequently misspell some words. Last but not least, we are dealing with tweets potentially written with different versions of the Portuguese spelling agreement. Consequently, this flexibility would not be achievable using exact string matching algorithms.

As described in [4], in this third step (figure 3.5): “we identify the citation inside the cited fragment. To do so, the content of a twitter citation is compared to its fragment text and a match is considered when a substring of the tweet text is found. A valid substring has a predefined number of minimum words to be considered as a valid quote, e.g. 10 words. Most of the tweet texts are not exact quotes because users often write additional information in their texts, therefore it was necessary to implement an algorithm that searches for a substring of the tweet text in its fragment text. As a result, an InfoRange instance, containing the correct location of the citation in the text, is created for each interpretation of the fragment”.

InfoRange is an auxiliary entity that contains the necessary information in order to create aware annotations in the fifth step (Section 3.4.1). This information includes the quote itself, the fragment interpretation text where the quote was found as well as the start and end positions of the quote in the text (see Table 3.2).

<table>
<thead>
<tr>
<th>InfoRange</th>
</tr>
</thead>
<tbody>
<tr>
<td>String start</td>
</tr>
</tbody>
</table>

Table 3.2: InfoRange entity

The main motivation for creating this auxiliary entity was to reduce the time of generating aware annotations. It separates the hard computational work of string matching from the generation of aware annotations. The string matching algorithm runs previously in this third step providing then enough information to create efficiently aware annotations in the fifth step. This efficiency demand was taken into account because users need to see the annotations promptly right after defining/updating the criteria.

The algorithm that identifies the substring inside the fragment text uses the Jaro-Winkler distance [13] to measure the similarity score between two strings by counting the minimum number of operations required to transform one string another. The score obtained by using Jaro-Winkler distance between two words is normalized such that 0 equates to no similarity and 1 is an exact match. We set the threshold to 0.9 after empirically observing that with this value we basically did not obtain any false
positives and we were also able to tolerate different versions of the Portuguese spelling agreement on people’s writings. We decided to use Jaro-Winkler distance because in this algorithm we are comparing small strings, like words and names. This distance prioritizes differences near the start of the string than differences near the end of the string. To apply Jaro-Winkler distance to our string matching algorithm we used the code from Apache.org.

It is important to remember that the main goal of this string matching algorithm is not only to find a match between the text of a tweet in its correspondent fragment, it is more precise than that. This algorithm finds a substring of a tweet text in its correspondent fragment. Each Twitter Citation contains information about the tweet text as well as the text from its correspondent Fragment.

![Figure 3.6: Fragment interpretation differences (first example)](image)

![Figure 3.7: Fragment interpretation differences (second example)](image)

A Citation is associated to 0 or more Info Ranges because we pretend to create an info range for each FragInter where the citation appears (see Figure 3.1). The fragment text may be different or rearranged.
according to different fragment interpretations. For example, figures 3.6 and 3.7 illustrate, in blue, the differences between two authorial editions that are being compared. In Figure 3.6, we show a fragment named “Adoramos a perfeição, porque a não podemos ter” and the differences between the Teresa Sobral Cunha’s edition and Jerónimo Pizarro’s edition. It is possible to observe, in the right column, that this fragment is part of the four authorial editions, although having slightly differences in each one of them. In Figure 3.7, we show a fragment named “[Carta a João Gaspar Simões, 28 de Julho de 1932]” and the differences between Jacinto do Prado Coelho’s edition and Richard Zenith’s edition. It is possible to observe, in the right column, that this fragment is only part of two of the four authorial editions, showing this way an example of a fragment that only exists in some authorial editions.

In first place, for each Twitter Citation instance it is done a preprocessing for both texts (the tweet text and the fragment text). This preprocessing consists in converting both texts to lower case and cleaning some special characters from the original tweet text. To remove and clean the original tweet text we created an auxiliary Java method String cleanTweetText(String originalTweetText) that receives as an argument the original tweet text and returns as a result the tweet text ready to be compared against the fragment text. While empirically using Jaro-Winkler distance, we realized that it would make things more flexible and precise if both texts were converted to lower case. We noticed that some people write their tweets with full capital letters and others without any capital letters so to normalize these extreme situations we decided to convert everything to lower case. If we had not done that, Jaro-Winkler distance would return lower scores even if we were comparing the same words. For example, the Jaro-Winkler score between the words “Gato” and “gato” is 83. On the other hand, if the words are “gato” and “gato” the score is 100. That is why we decided to convert everything to lower case. Consequently, this allowed us to use a bigger and more precise threshold when comparing two words between the tweet text and fragment text. More concretely a threshold of 90. The bigger the threshold the more precise is the algorithm.

The special characters that we decided to clean are given by the regular expression:

```
First set: \["*;<>\"
Second set: -.,;?!\\n
We identified the characters from the first set as being frequently written by users in their tweets and it was important to remove them because they lowered Jaro-Winkler score in the same way capital letters did. For example, the Jaro-Winkler score between the words “=gato” and “gato” is 93 instead of 100. We removed them by replacing each of these characters by a blank space. Blank spaces also affect Jaro-Winkler performance but since we are splitting the words in the tweet text by blank spaces this will not affect Jaro-Winkler values while comparing tweet text words with fragment text words.

Regarding the second set, we decided to remove hyphens, full stops, commas, semicolons, question
marks and exclamation mark if there were blank spaces to the left and to the right. We noticed that some people write these characters incorrectly with blank spaces to the left and to the right and since we are splitting tweet text words by blank spaces these characters with blank spaces around them would affect the performance of the string matching algorithm. Last but not least, the letter ‘q’ has a special treatment. When found between two blank spaces we replace it by the word “que”. This assumption only works for the Portuguese language, since many Portuguese writers on Twitter simply use the letter ‘q’ instead of “que” to abbreviate their tweet texts.

It is important to refer that these treatments and assumptions were taken into account due to empirical observations from the first tweets collected in March and April. This preprocessing of the tweet text improved greatly the latter performance of the string matching algorithm, although there is probably more room for improvements for the regular expressions treatment in the Portuguese language. These two sets of characters were the main disturbances we found.

Finished the preprocessing step, the following step consists in specifying the parameters that are going to be used by the pattern finding algorithm. There are two fundamental parameters: the word window and Jaro-Winkler threshold. The word window parameter defines a predefined number of minimum words for a substring to be considered a valid quote (this parameter is currently set to 10). However, the common guideline relies on 15-20 words in a sentence (in the English language). The Jaro-Winkler threshold parameter defines the minimum acceptable Jaro-Winkler similarity score between two words that is used by the pattern finding algorithm (this parameter is currently set to 0.9, the maximum would be 1).

Afterwards, the pattern finding algorithm begins. It consists in a for loop that iterates over the cleaned tweet text split by blank spaces. In each iteration we try to find a match between the tweet word we are iterating and the most similar word in the associated fragment text. There are two possible outcomes for each iteration: we find a fragment word with a score above 0.9 or we don’t.

If we do not find a match above 0.9 we assume that the tweet word does not exist in the fragment text so we continue straight to the next iteration and we search for the following tweet word.

If we find a match above 0.9 we add that word to a variable called String patternFound, we record the position it was found in the fragment text and we continue to the following tweet word until the word window is achieved. If during this process we do not find a match between the current tweet word and a word in the fragment text before reaching the limit of word window, we then reset the value of the variable patternFound and we start searching for another substring inside the tweet text until we reach the word window.

The pattern finding algorithm ends when we reach the minimum word window and, consequently, a substring of the tweet text was successfully found or when we finished iterating over the tweet text and

---

no substring was long enough to meet the minimum word window.

While the variable \textit{patternFound} is being update during this process, the start and end positions of the \textit{patternFound} are also updated in order to know where the substring from the tweet text appears in the fragment text. Finished this algorithm we get as a result the substring that was found as well as the start and end positions. This is the necessary information that is present in class \textit{InfoRange} (see Table 3.2).

Equally to Lucene, our string matching algorithm can also be evaluated in terms of precision and recall (see Chapter 4).

\subsection{3.3.5 Create tweet objects}

Each entry in a JSON file correspond to a tweet collected on step (1) Section 3.3.1. As illustrated in Figure 3.8 and described in [4], in the fourth step: “Tweet instances are created to all entries in the JSON files, and are associated with the respective TwitterCitation instances. After this step there are two types of false positives: tweets that contain one of the keywords but are below the Lucene threshold, which are represented by Tweet instances without an associated TwitterCitation; and tweets above the threshold that do not contain any substring of the fragment, which are represented as TwitterCitation instances but do not have any InfoRange associated. Both types of information is not discarded to allow the tuning of steps (2) and (3)”.

After this step a TwitterCitation is associated to at least one Tweet instance that corresponds to the original tweet that originated that twitter citation. However, a TwitterCitation can be associated to one or more Tweets if we take into account the retweets of the original tweet. On the other hand, a Tweet can only be associated to a TwitterCitation because we assume that one tweet can not be associated to multiple quotes. Additionally, some instances of Tweet may not be associated to a TwitterCitation instance, this situation corresponds to the first type of false positives.
The first type of false positive is due to the Twitter’s Search API performance. By default, Twitter search queries return tweets that include all the terms specified in the query. This means that if we want to search, for example, for “Fernando Pessoa” we do not need to specify an “AND” or ‘+’ operator. By default it will return tweets containing both “Fernando” and “Pessoa” words. The same goes for “Livro do Desassossego”, “Bernardo Soares” and “Vicente Guedes”. The problem is that while “Fernando Pessoa” and “Livro do Desassossego” are relatively known and used combinations of words on Twitter, we can not say the same thing about “Bernardo Soares” and “Vicente Guedes” which are not so frequently written in tweets. This means that while searching for “Bernardo Soares” and “Vicente Guedes” we will obtain several tweets containing only the word “Bernardo” or “Soares”, or only the word “Vicente” or “Guedes”. Therefore, tweets containing only one keyword or tweets that are not in fact LdoD citations are the first type of false positives.

The second type of false positive was also mentioned in Section 3.3.3. These false positives are obtained due to Lucene’s performance. Some tweets are above the established threshold so Lucene classifies them as citations but in fact they are not LdoD citations.

Although it was not mentioned in [4], there is a third type of false positives that can also be evaluated. This type is due to the string matching algorithm performance in third step (Section 3.3.4). These false positives correspond to tweets that the string matching algorithm classifies as citations but in fact they are not LdoD citations. This type is rarer because the string matching algorithm verifies word by word the tweet text while comparing with the potential correspondent fragment text until completing a predefined minimum number of words to be considered as a valid quote, e.g. 10 words.

The evaluation of these three steps is explained in Chapter 4 using Precision and Recall measures.

The main motivation for creating the “Tweet” entity was to keep a record of all the tweets collected as well as how they relate to the Twitter Citation entity in terms of the original tweet or retweets. The meta information contained in this entity is also illustrated in Table 3.1.

Besides being mentioned in Section 2.4.3, it is important to refer that we only started distinguishing between tweets and retweets after May 14th. Consequently, this means that during, approximately, the first two months we treated retweets as original tweets which resulted in an increase of twitter citations detected. This happened because we only noticed later on that it would be useful and interesting to make this distinction, in terms of counting. By treating these two concepts differently, the number of twitter citations created is more faithful because we only create twitter citations for original tweets. Their retweets are only used for counting purposes, for example, to know how many times a fragment has been cited.

We took this approach but this does not mean that it would be incorrect to create twitter citation instances for each tweet retrieved instead (as we were doing during the first two months). Some may

5https://unionmetrics.com/resources/how-to-use-advanced-twitter-search-queries/
consider that a retweet is also a form of citation.

3.4 Phase 2: Build SAVE

3.4.1 Create aware annotations

This step (Figure 3.9) takes into account the concepts of Social Media Criteria and Aware Annotations which are some of the most important and relevant concepts in this thesis.

A social media criteria is represented by the class “SocialMediaCriteria”. A set of social media criteria can be added to a virtual edition in order to enrich it with a new set of features. This set includes: Media Source, Time Window, Geographic Location and Frequency and can be considered at the moment of creation and even later. After applying these criteria the fragments present in a SAVE are modified in terms of annotations, which are automatically added and removed. Therefore, we can say that SAVE is a specific instance of a Virtual Edition which brings new functionalities while using a set of social media criteria. However, in terms of code, there is not a specific entity to represent a SAVE. What distinguishes a SAVE from a common Virtual Edition is the fact that the former has a set of social media criteria associated.

On the other hand, an aware annotation is represented by the class “AwareAnnotation”, an extension of the original class “Annotation”. To support SAVE, one of the steps in this enrichment process consists of extending the class Annotation by creating two new sub classes: “HumanAnnotation” and “AwareAnnotation”. A human annotation embodies the current behaviour of an annotation in LdoD Archive and it is made by users while interacting with fragment interpretations. An aware annotation is a new concept which represents an annotation automatically generated when certain fragment is cited on social media. Each aware annotation has only one citation associated. But a citation can be associated to several
aware annotations (each of these aware annotations is belongs to a different Virtual Edition).

Social media criteria can be defined when a user is editing his virtual edition. When a user creates a virtual edition he is able to define a set of criteria in order to make his virtual edition socially aware. These criteria define which aware annotations are automatically created. By default none of these criteria is applied (which would produce a regular virtual edition), but if a user wants to make it socially aware he needs to define his set of social media criteria: social media source (Twitter, Facebook, or combinations of them), a time window (a start date and end date for the citations), a geographic location (countries where the citation source belongs to) and a frequency (fragments with a number of citations greater or equal to a certain number). The frequency criteria takes into account the number of retweets of each citation.

Note that currently we are only collecting citations from Twitter. Facebook posts are a possible future extension as mentioned in Section 3.3.2.

As mentioned in [4]: “whenever a user creates or edits a social media criterion, the fifth step (5) occurs where all the citations associated with the fragments contained in the virtual edition are filtered according to the set of virtual edition social media criteria, and AwareAnnotation instances are automatically created or deleted, depending on the changes on the social media criteria”.

### 3.4.2 Presentation

![Diagram](image)

**Figure 3.10: Sixth step - Phase 2 (blue entities) [4]**

The fragment interpretations of a SAVE have awareness properties based on the set of criteria applied. Once you click on a fragment, in order to read it, the words quoted are highlighted and there is an annotation associated to the quote displaying meta information about the citation (see Table 3.1).

After creating a SAVE a user can visualize cited fragments according to different criteria combinations. For example, it is possible for a user to visualize cited fragments from a different media source or
time window compared to what was previously chosen at the creation moment. The new set of criteria is then saved but can be modified again at anytime.

As described in [4], “the Social Aware Archive interprets these criteria and generates the AwareAnnotations that fulfill the conditions, for instance, if the TimeWindow of a virtual edition specifies the last two months, only citations done in the last two months are considered for the creation the of AwareAnnotations for that virtual edition. Note that this customization mechanism is dynamic, which means that whenever a criterion is changed the set of AwareAnnotations is automatically regenerated.”

As mentioned in [4], “when a user selects a fragment of a virtual edition which contains social aware annotations he can visualize the cited part of the text and the link for the citing tweet” as well as filter the fragments present in a virtual edition based on the defined set of criteria.

![Figure 3.11: Front end additions (red boxes) to the management of a Virtual Edition](image)

Figure 3.11 represents the management panel where a user can define his set of social media criteria.

### 3.5 Architecture

In this section we present the architecture of our solution. To illustrate the architectural modifications and additions that were done to the LdoD Archive we show two different architectural views of the system. A Component-and-Connector view (Section 3.5.1) and a Module view (Section 3.5.2).

An architectural view may mix more than one architectural style. A view describes the architecture software elements and the relations between them. An element can be either a component or a module [14].

As mentioned in [14], a component is always a runtime entity. The Component-and-Connector view
type illustrates “how the system is structured as a set of elements that have runtime behaviour (components) and interactions (connectors)”.

A module is an implementation unit that provides a coherent set of responsibilities [14], it contains the code that can execute in several components and a component can execute the code of several modules.

In the description of an architecture there should be at least one view of the Decomposition architectural style of the module viewtype. That is why we present a decomposition of modules required for this thesis implementation in Section 3.5.2.

3.5.1 C&C view

![Component-and-Connector Diagram](image)

**Figure 3.12:** New component-and-connector view of LdoD Archive (reused most of the components from [2]).

The architectural structure is inspired by the Social Impact architecture (see Figure 2.5) and it reuses most of the components in LdoD Archive [2] but with some modifications and additions.

Figure 3.12 represents a Component-and-Connector view of the system where the elements are represented by components and connectors. “Component-and-connector structures show a runtime view of the system. The relation in all component-and-connector structures is attachment, showing how the components and the connectors are hooked together” [14].

The main similarity with Social Impact architecture is that both systems have a module responsible for collecting information from social media and a module responsible for extracting citations.

The main architectural differences between Social Impact project is that instead of Social Bus (Sec-
tion 2.4.1) we are using Twitter4j API (Section 2.4.3) for collecting tweets; and we are working with Fernando Pessoa’s *The Book of Disquiet* instead of all his work present in *Arquivo Pessoa*.

All the new modules developed in this thesis (Section 3.5.2) run in the *LdoD Application Server* component illustrated in Figure 3.12. This figure reuses most of its original components from [2]. The main contribute of this thesis is adding functionality to the Archive by developing new modules inside the core component of the LdoD Archive which is *LdoD Application Server*. By observing this figure it is possible to observe, in blue, a new connector between *LdoD Application Server* and Twitter which is responsible for fetching tweets. In a Component-and-Connector view type a connector embodies a communication protocol.

The major executing components of this thesis are the *LdoD Application Server* (the server) and the *LdoD Object-Oriented Repository* and *LdoD TEI File Repository* (both repositories).

In the first step (Phase 1 - Section 2.4.3), the collected tweets are stored in JSON files. These files are in a specific folder called "/social/aware" inside the *LdoD Application Server* component.

During the following steps of Phase 1 and Phase 2 (Sections 3.3 and 3.4), the Application Server interacts at runtime with the Object-Oriented Repository. During Phase 1 it creates Twitter Citation instances, Info Range instances and Tweet instances; during Phase 2 it creates Aware Annotation and Social Media Criteria instances; then these objects are stored in the Object-Oriented Repository. Note that there are no duplicate tweets or citations in our database.

The interaction between *LdoD Application Server* and the *LdoD TEI File Repository* is present during Phase 1, more precisely during the second step (*Citation Detection* - Section 3.3.3). The module responsible for citation extraction uses Lucene library in order to find out the most suited fragment to be associated to each tweet. For that, the *LdoD Application Server* needs to access the *LdoD TEI File Repository* where the fragments are located.

### 3.5.2 Module view

![Figure 3.13](image)

"Module views show the units of implementation that have to be built in order to create that system"
In what concerns the construction of SAVE, the decomposition module view type in figure 3.13 illustrates new modules that were implemented. In this architectural view type the elements are the modules (classes, pieces of code). The Decomposition architectural style is useful to show the structure of the project, the parts of the system that are implemented and responsibilities as well as testing information. It is used to explain the system’s functionality providing a top-down representation of the system’s responsibilities.

First, a Twitter4j module is responsible for fetching information from Twitter and it runs in the LdoD Application Server component. This module represents the code responsible for collecting tweets and writing them to JSON files. The implementation of this module corresponds to the first step of Phase 1 (see Section 3.3.1).

Second, there is a Citation extraction module, that also runs inside the LdoD Application Server component. This module uses Lucene library and it is divided into two sub modules: a Citation detection module and a Citation extraction module. The former is responsible for mapping the fetched information with the fragments present in our database, the latter for creating instances of Citation. Both sub modules interact with the LdoD Object-Oriented Repository while mapping the tweets with the fragments and by creating instances of Citations in our database. The implementation of this module corresponds to the second step of Phase 1 (see Section 3.3.3).

In third place there is a Aware Annotations module which is responsible for providing awareness properties to the current annotations in LdoD Archive. These awareness properties are implemented by extending the previous class Annotation. This module also runs inside the LdoD Application Server component. This module is divided into two sub modules: a Quote matching module and an Annotation creation module. The former is responsible for finding a substring of a tweet text in one of the fragments of the book, it uses a string matching algorithm that given a text and a pattern finds the several positions where the pattern appears in the text; the latter is responsible for creating instances of Aware Annotation in the correspondent fragment interpretation. Both these sub modules interact with the LdoD Object-Oriented Repository while creating instances of InfoRante and AwareAnnotation in our database. The implementation of this code corresponds to the third step of Phase 1 (see Section 3.3.4) and fifth step of Phase 2 (see Section 3.4.1). During the fifth step this module uses the following module (Social Media Criteria module) in order to create aware annotations according to a set of criteria.

Finally, there is a Social Media Criteria module that is responsible for providing the user the ability to define his own set of criteria and for presenting the aware annotations according to the set of criteria previously defined. This module also runs inside the LdoD Application Server component and interacts with the LdoD Object-Oriented Repository when creating instances of SocialMediaCriteria. This module is divided into four different sub modules. Each of these four sub modules represents a social media criterion. There can only be one instance of each criterion per virtual edition. Therefore, you can only have
four instances of SocialMediaCriteria, one instance for criterion. For example, you can not have two time windows or two frequencies. However, you can select several countries or media sources from where you want to retrieve your citations. Although one can select several countries, this is still represented by one instance of the class GeographicLocation. The implementation of this module corresponds to the code described in both steps of Phase 2 (see Section 3.4.1 and 3.4.2).

### 3.6 Problems faced

In this section we describe the problems faced during the development of our solution. As explained in Sections 2.4.3 and 2.5.2 we initially thought about using Social Bus tool to collect tweets and posts, but sooner we realized that it would not be possible since Twitter and Facebook’s API had changed. Then we experimented Spring Social during some weeks, but as explained in Section 2.4.3 it was not possible to collect entire texts from tweets that were truncated for having more than 140 characters. That is why we chose Twitter4j library to implement the module responsible for collecting tweets.

Another problem was due to the fact that some of the quotes on social media are not correctly spelled. People sometimes misspell some the words or write the words in an incorrect order. The solution we found for these situations was to highlight the complete sentence were the partial quote was found.

![VickLinn](https://example.com/vicklinn)

Pedi tão pouco à vida e esse mesmo pouco a vida me negou. Uma rãsta de parte do sol, um campo, um bocado de sossego (...), não me pesar muito o conhecer que existo, e não exigir nada dos outros, nem exigirem eles nada de mim.

Bernardo Soares

**Figure 3.14:** Tweet text that did not quote the complete sentence

![Tweet text](https://example.com/tweet)

Figure 3.15 illustrates, in the red box, what would be highlighted by the substring pattern finding algorithm if we did not make corrections to the starting position. In yellow it is highlighted the complete sentence. This solution guarantees that the sentence where the quote appears is completely highlighted. The tweet is represented in Figure 3.14. This is a success example.

Unfortunately, Figure 3.16 illustrates a situation where our correction to the starting position did not work. For example, we noticed that several users wrote the quote “Adoramos a perfeição, porque não a podemos ter; repugná-la-íamos, se a tivéssemos” instead of the original quote “Adoramos a perfeição, 
porque a não podemos ter; repugná-la-lamos, se a tivéssemos”. The difference was in the position of the word “não” but this small change influences the way the substring pattern finding algorithm works.

This happened because the quote was in the first sentence of the fragment. To highlight the complete sentence, our solution consists of finding the last and next full stop from where the quote was identified. However, this solution does not work completely if the quote is found in the first sentence since there is not a last full stop. To create a workaround we though about finding the last capital letter from the starting position of the quote. The problem is that we converted the whole text to lower case (to improve Jaro-Winkler results in Section 3.3.4). Consequently, we are not able to highlight the beginning of the sentence in the situations where the quote occurs in the first sentence of a fragment.
3.7 Summary

Summing up, in this chapter we started by explaining the new concept of “Socially Aware Virtual Editions” which are virtual editions that are sensitive to citations of *The Book of Disquiet* on social media. The we specified the requirements as well as the two phases of our solution. Phase 1 consists in collecting information from social media, process the tweets collected, find out if they are citations of the book or not and then find then starting and ending positions of quote in the text. Phase 2 consists in creating aware–annotations in the text (highlighting in yellow the cited sentences) as well as defining a set of criteria which users can personalize in order to visualize annotations in the fragments according to this set of criteria.
4 Evaluation

Contents

4.1 Results ................................................. 51
4.2 Twitter4j performance ................................... 53
4.3 Lucene performance .................................... 54
4.4 String matching algorithm performance ............... 57
4.5 Summary and comparisons .............................. 60
From March 16th to September 27th we collected 81,833 different tweets (including retweets). In this chapter we present statistical results about the absolute number of citations, info ranges and annotations created while varying the Lucene score threshold and keywords used; as well as performance levels of the different steps of our solution using precision and recall measures.

In section 4.1 we show the partial and total number of citations, info ranges and annotations created depending on the Lucene score that was used and keywords. Additionally we also present the number of annotations created for each one of the four authorial editions.

In section 4.2 we show the precision of Twitter4j module depending on the keyword used to collect quotes of *The Book of Disquiet* from Twitter.

In section 4.3 we describe how Lucene performance is tested for different thresholds.

In section 4.4 we describe how the substring matching algorithm performance is tested using different Lucene thresholds and word windows.

### 4.1 Results

In this section we start by showing a table with the total number of collected tweets (Table 4.1). Then we show three tables (each representing a different Lucene threshold) that illustrate how the number of citations and info ranges vary depending on the Lucene thresholds and keywords used (Tables 4.2, 4.3 and 4.4). Additionally, we show one more table that illustrates how the number of annotations vary for each one of the authorial editions depending also on the Lucene threshold (Table 4.5).

<table>
<thead>
<tr>
<th>Tweets</th>
<th>&quot;livro do desassossego&quot;</th>
<th>1067</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;bernardo soares&quot;</td>
<td>2272</td>
</tr>
<tr>
<td></td>
<td>&quot;fernando pessoa&quot;</td>
<td>79,271</td>
</tr>
<tr>
<td></td>
<td>&quot;vicente guedes&quot;</td>
<td>104</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>82,714</td>
</tr>
<tr>
<td>Total (without repeated tweets)</td>
<td></td>
<td>81,833</td>
</tr>
</tbody>
</table>

*Table 4.1: Total number of tweets collected by keyword (retweets included in all of them)*

<table>
<thead>
<tr>
<th>Tweets</th>
<th>Citations</th>
<th>Info Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;livro do desassossego&quot;</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>&quot;bernardo soares&quot;</td>
<td>605</td>
</tr>
<tr>
<td></td>
<td>&quot;fernando pessoa&quot;</td>
<td>6069</td>
</tr>
<tr>
<td></td>
<td>&quot;vicente guedes&quot;</td>
<td>17</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>6993</td>
</tr>
<tr>
<td>Total (without repeated tweets)</td>
<td></td>
<td>6621</td>
</tr>
</tbody>
</table>

*Table 4.2: Number of citations and info ranges created for Lucene score > 20*
By observing tables 4.2, 4.3 and 4.4 it is possible to observe certain patterns and take some conclusions.

First, independently of the Lucene threshold, the keyword that originated more instances of citation is “fernando pessoa”, followed by “bernardo soares” and “livro do desassossego”, while “vicente guedes” produced poor results. However, by observing Table 4.3, one can notice that the number of info ranges produced was bigger for the keyword “livro do desassossego” than “bernardo soares”.

Second, as it was expected, the number of citation and info ranges instances was bigger when we established Lucene threshold to 20 (Table 4.2). This was the most flexible and tolerant threshold of the three that were tested, therefore producing 3.6 times more citations and 1.7 times more info ranges than when the threshold was set to 30 (Table 4.3).

With the previous three tables we can conclude that the keyword “vicente guedes” was practically useless because it did not originate almost any citation and originated 0 info ranges (which means that not a single annotation is going to come from this keyword). Additionally, we observed that changing Lucene threshold to 20 would also originate more info ranges (consequently, more annotations) than the values of 30 or 40.

Bellow, we present a table that shows the number of annotations created for each authorial edition, also depending on the Lucene threshold.

By observing Table 4.5 we can see that by using a lower Lucene threshold we produce a bigger number of annotations, naturally. Besides that, some authorial editions were more cited (during March 16th and September 27th) than the other. For example, the most cited authorial edition was the Teresa Sobral Cunha’s edition, followed by Richard Zenith’s edition, then Jernónimo Pizarro’s edition, and finally Jacinto do Prado Coelho’s edition. A possible explanation for this to happen is due to the fact that...
Table 4.5: Number of annotations for each authorial edition depending on the Lucene threshold

<table>
<thead>
<tr>
<th>Authors</th>
<th>Lucene &gt; 20</th>
<th>Lucene &gt; 30</th>
<th>Lucene &gt; 40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard Zenith</td>
<td>1428</td>
<td>814</td>
<td>564</td>
</tr>
<tr>
<td>Teresa Sobral Cunha</td>
<td>1455</td>
<td>834</td>
<td>572</td>
</tr>
<tr>
<td>Jacinto do Prado Coelho</td>
<td>1292</td>
<td>701</td>
<td>459</td>
</tr>
<tr>
<td>Jerónimo Pizarro</td>
<td>1366</td>
<td>773</td>
<td>545</td>
</tr>
<tr>
<td>Total</td>
<td>5541</td>
<td>3122</td>
<td>2140</td>
</tr>
</tbody>
</table>

different editions contain different fragments as well as different fragment interpretations of the same fragment.

Summing up, we conclude that by using a lower Lucene threshold we produce more citations, hence more info ranges, therefore more annotations. However, in this section we only explored absolute results about the number of instances created for each of these entities. We did not take into account performance levels until now. In the sections below (Sections 4.2, 4.3 and 4.4) we present Precision and Recall measures for Lucene and our string matching algorithm.

## 4.2 Twitter4j performance

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;livro do desassossego&quot;</td>
<td>28%</td>
</tr>
<tr>
<td>&quot;bernardo soares&quot;</td>
<td>15%</td>
</tr>
<tr>
<td>&quot;fernando pessoa&quot;</td>
<td>7%</td>
</tr>
<tr>
<td>&quot;vicente guedes&quot;</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 4.6: Twitter4j precision according to keywords used for fetching tweets

In this section we present a Table 4.6 that shows the precision of Twitter4j module according to the keywords that were used to collect citations of *The Book of Disquiet* from Twitter. To compute these values we selected, for each one of the keywords, 100 random tweets that were collected from March 16th to August 30th and determined how many were in fact citations from the book. In other words, these percentages correspond to citations that originate at least one info range.

By observing the table we notice that the keyword that was more helpful and precise while searching for tweets when we used Twitter’s API was "livro do desassossego", followed by “bernardo soares” and “fernando pessoa”. On the other hand, searching for the keyword “vicente guedes” did not bring any results. These results were expected since “livro do desassossego” is more specific and precise than all the others. And “bernardo soares” is more precise than “fernando pessoa” naturally because it was the heteronym Bernardo Soares that wrote *The Book of Disquiet*. Unfortunately, searching for Vicente Guedes (that was another heteronym that wrote this book) did not produce any results.

It is also interesting to compare this table with the absolute results from the previous section, where the majority of citations, info ranges and annotations were created using the keyword “fernando pessoa”,...
followed by “bernardo soares” and “livro do desassossego”.

Summing up, when we search for the keyword “livro do desassossego” we have the highest probability of finding a tweet that is a quote of the book (corresponding to 28% of probability). On the contrary and based on section 4.1 it is the keyword “fernando pessoa” that originates more annotations since it is also this keyword that collected more tweets. This happens because the keyword “fernando pessoa” is more often written than the keyword “livro do desassossego”.

### 4.3 Lucene performance

As mentioned in the section above (Section 4.2), we randomly chose 100 tweets for each keyword from March 16th to September 27th. We then looked at each one of them, carefully, to determine if they were citations from the book, or merely tweets that contained that keyword and were just thoughts or opinions. We did this using the search tool from the LdoD Archive (see Figure 4.1) by manually verifying if the tweet texts were citations from the book.

![Figure 4.1: Simple search of a sentence in the LdoD Archive](image)

After manually identifying each tweet as being a citation from the book or not, when then proceeded to test the performance of Lucene. As mentioned in Sections 2.4.4 and 3.3.3, the content of the tweets in the JSON files is compared to the inverted index of fragments produced by Lucene and, depending on a threshold, a twitter citation is created if the content is similar enough.

Currently we are using Apache Lucene 6.2.0. As mentioned in [3] and described in [2], Lucene score combines the Boolean Model and Vector Space Model (VSM) alongside normalization measures, for a given query (q) and document (d):

1. [https://lucene.apache.org/core/6.0.0/index.html](https://lucene.apache.org/core/6.0.0/index.html)
2. [https://lucene.apache.org/core/6.0.0/core/org/apache/lucene/search/package-summary.html#package_description]
\[ \text{score}(q,d) = \text{coord}(q,d) \times \text{queryNorm}(q) \times \sum_{t \in q} (\text{tf}(t \in d) \times \text{idf}(t)^2 \times \text{t.getBoost}()) \]

As explained in \(^3\):

- \(\text{coord}(q,d)\): “is a score factor based on how many of the query terms are found in the specified document. Typically, a document that contains more of the query’s terms will receive a higher score than another document with fewer query terms.”

- \(\text{queryNorm}(q)\): “is a normalizing factor used to make scores between queries comparable. This factor does not affect document ranking (since all ranked documents are multiplied by the same factor), but rather just attempts to make scores from different queries comparable.” For example, as described in [3], this normalization factor is helpful when we are comparing smaller documents with larger ones.

- \(\text{tf}(t \in d)\): term frequency, “number of times term \(t\) appears in the currently scored document \(d\). Documents that have more occurrences of a given term receive a higher score.”

- \(\text{idf}(t)\): inverse document frequency, this value corresponds to “the inverse of the number of documents in which the term \(t\) appears. This means rarer terms give higher contribution to the total score. \(\text{idf}(t)\) appears for \(t\) in both the query and the document, hence it is squared in the equation.”

- \(\text{t.getBoost()}\): “is a search time boost of term \(t\) in the query \(q\) as specified in the query text.” By default this value is 1.

What we did in this section was changing the threshold of the Lucene score in order to see the impact in terms of Precision and Recall of Lucene.

<table>
<thead>
<tr>
<th>TP</th>
<th>“livro do desassossego”</th>
<th>“bernardo soares”</th>
<th>“fernando pessoa”</th>
<th>“vicente guedes”</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>69</td>
<td>83</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td>0.63</td>
<td>0.48</td>
<td>0.33</td>
<td>0</td>
</tr>
<tr>
<td>Recall</td>
<td>0.61</td>
<td>1.0</td>
<td>0.71</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.7: Lucene performance for Lucene score > 20

In terms of Precision, by looking at the Tables 4.7, 4.8 and 4.9 it is possible to conclude that the Precision is higher when the Lucene threshold is also higher (independently of the keyword used). This happens because higher Lucene thresholds originate a less amount of False Positives. For example, the keyword “livro do desassossego” originated 0 False Positives for a Lucene score of 40 versus the

\(^3\)https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html
| TP | "livro do desassossego" | 14 | 7 | 5 | 0 |
| FP | 3 | 7 | 0 | 2 |
| TN | 69 | 78 | 93 | 65 |
| FN | 14 | 8 | 2 | 0 |
| Precision | 0.82 | 0.5 | 1.0 | 0 |
| Recall | 0.5 | 0.47 | 0.71 | - |

Table 4.8: Lucene performance for Lucene score > 30

| TP | "livro do desassossego" | 11 | 7 | 4 | 0 |
| FP | 0 | 4 | 0 | 1 |
| TN | 72 | 81 | 93 | 66 |
| FN | 17 | 8 | 3 | 0 |
| Precision | 1.0 | 0.64 | 1.0 | 0 |
| Recall | 0.39 | 0.47 | 0.57 | - |

Table 4.9: Lucene performance for Lucene score > 40

10 False Positives originated by a Lucene score of 20. This means that to obtain a higher Precision it would be recommended to use a higher Lucene threshold.

In terms of Recall, by looking at the same three tables we can observe the inverse phenomenon. The higher the threshold, the lower the Recall. This happens because higher thresholds are too rigorous, therefore producing a great amount of False Negatives and reducing drastically the number of True Positives. For example, by looking at the keyword "bernardo soares" it is possible to observe that, for a Lucene threshold of 20, the number of False Negatives is 0 and the number of True Positives is 15 versus the 8 False Negatives and only 7 True Positives obtained by using a Lucene threshold of 40. This means that to obtain a higher Recall it would be recommended to use a lower Lucene threshold since it would preserve the majority of True Positives and reduce the number of False Negatives because lower thresholds are more flexible and tolerant.

Summing up, to obtain more precise results it would be more appropriate to use a Lucene threshold of 40 because it would reduce the number of False Positives (tweets that Lucene considered as being citations but in fact they were not). On the other hand, we would be losing several citations with this extremely rigorous approach, resulting in an increase of False Negatives (tweets that Lucene considered as not being citations but in fact are LdoD citations).

Since *The Book of Disquiet* is not so frequently cited in social media, our goal was to collect every citation we could, therefore lowering the Lucene threshold (even if it meant less precise results by Lucene). However, we still needed to present correct and precise results to users. That is why we present in the following Section 4.4 the performance levels of our string matching algorithm that solves this problem by rigorously finding a substring pattern of a given tweet in the most likely correspondent fragment.
## 4.4 String matching algorithm performance

<table>
<thead>
<tr>
<th></th>
<th>&quot;livro do desassossego&quot;</th>
<th>&quot;bernardo soares&quot;</th>
<th>&quot;fernando pessoa&quot;</th>
<th>&quot;vicente guedes&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TP</strong></td>
<td>15</td>
<td>13</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>TN</strong></td>
<td>10</td>
<td>16</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td><strong>FN</strong></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.89</td>
<td>0.87</td>
<td>0.8</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.10: String matching algorithm performance for Lucene score > 20 and word window > 10

<table>
<thead>
<tr>
<th></th>
<th>&quot;livro do desassossego&quot;</th>
<th>&quot;bernardo soares&quot;</th>
<th>&quot;fernando pessoa&quot;</th>
<th>&quot;vicente guedes&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TP</strong></td>
<td>13</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>TN</strong></td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>FN</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.93</td>
<td>1.0</td>
<td>0.8</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.11: String matching algorithm performance for Lucene score > 30 and word window > 10

<table>
<thead>
<tr>
<th></th>
<th>&quot;livro do desassossego&quot;</th>
<th>&quot;bernardo soares&quot;</th>
<th>&quot;fernando pessoa&quot;</th>
<th>&quot;vicente guedes&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TP</strong></td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>TN</strong></td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>FN</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.12: String matching algorithm performance for Lucene score > 40 and word window > 10

In this section we describe how the substring matching algorithm performance is tested using different Lucene thresholds (Tables 4.10, 4.11, 4.12) and word windows (Tables 4.13, 4.14 and 4.15). Tables 4.10, 4.11 and 4.12 illustrate the exact same process that was done in the previous Section 4.3 but in this case we are testing the Precision and Recall of our string matching algorithm described in detail in Section 3.3.4. For that, we used the exact same group of tweets and we computed Precision and Recall measures.

Our string matching algorithm receives as an input only the Positives detected by Lucene (as illustrated in Figure 3.2). For example, in Table 4.10 we receive as an input 27 (15+0+10+2) citations created by Lucene using a threshold of 20 and the keyword “livro do desassossego”. These 27 correspond the 27 (17 True Positive and 10 False Positive detected by Lucene in Table 4.2 for the keyword “livro do desassossego”).

In terms of Precision, by analyzing Tables 4.10, 4.11 and 4.12 we conclude that our algorithm is...
100% precise for every keyword and for every Lucene threshold. This happened because our algorithm did not produce any False Positives.

In terms of Recall, it is important to remember that it depends on the number of False Negatives (tweets that our algorithm considered as not being citations but are in fact LdoD citations). By observing Tables 4.10 and 4.11 we can see that there were 1 or 2 False Negatives per keyword. These False Negatives were due to the fact that the word window of 10 was not small enough to detect quotes that were small sentences with less than 10 words. By observing Table 4.12 we obtained a Recall of 100% for every keyword because there were 0 False Negatives. This happened because (as we saw in Section 4.3) by using a Lucene threshold of 40 we are obtaining high levels of Precision, consequently filtering the majority of False Positives in the previous step.

<table>
<thead>
<tr>
<th>TP</th>
<th>&quot;livro do desassossego&quot;</th>
<th>&quot;bernardo soares&quot;</th>
<th>&quot;fernando pessoa&quot;</th>
<th>&quot;vincente guedes&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>16</td>
<td>14</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>FN</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TN</td>
<td>7</td>
<td>12</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>

| Precision | 0.84 | 0.78 | 0.5 | - |
| Recall    | 0.94 | 0.93 | 0.8 | - |

Table 4.13: String matching algorithm performance for Lucene score > 20 and word window > 5

<table>
<thead>
<tr>
<th>TP</th>
<th>&quot;livro do desassossego&quot;</th>
<th>&quot;bernardo soares&quot;</th>
<th>&quot;fernando pessoa&quot;</th>
<th>&quot;vincente guedes&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>13</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>FN</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TN</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

| Precision | 0.93 | 0.78 | 1.0 | - |
| Recall    | 0.92 | 1.0  | 0.8 | - |

Table 4.14: String matching algorithm performance for Lucene score > 30 and word window > 5

<table>
<thead>
<tr>
<th>TP</th>
<th>&quot;livro do desassossego&quot;</th>
<th>&quot;bernardo soares&quot;</th>
<th>&quot;fernando pessoa&quot;</th>
<th>&quot;vincente guedes&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>FN</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TN</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| Precision | 1.0 | 0.86 | 1.0 | - |
| Recall    | 1.0 | 1.0  | 1.0 | - |

Table 4.15: String matching algorithm performance for Lucene score > 40 and word window > 5

Finally, in Tables 4.13, 4.14 and 4.15 we used Lucene thresholds of 20, 30 and 40, respectively, but changed the word window to more than 5 words (in order to detect smaller quotes compared to the word window of 10). Is is possible to observe that Precision values decreased, as it was expected, since we are now more tolerant. This originated False Positives. On the other hand, the Recall remained basically
the same because the False Negatives were the same (this means that False Negatives were quotes with less than 5 words).

With the results from this section we conclude that if we use a word window of at least 10 words our algorithm is 100% precise, independently of the Lucene threshold used in the previous step of our solution. In terms of Recall, we obtained better results when we combined our algorithm with a higher Lucene threshold. However, using a higher Lucene threshold of 40 has the cost of discarding useful tweets as concluded in the Section 4.3.

4.4.1 Examples

Figure 4.2: Example of a tweet that originated a True Positive citation

Considerar a nossa maior angústia como um incidente sem importância, não só na vida do universo mas da nossa mesma alma, é o princípio da sabedoria. ~Fernando Pessoa~

Figure 4.3: Annotation created by a true positive citation

Figure 4.2 illustrates a tweet that originated an annotation (Figure 4.3) in the LdoD Archive (True Positive).

Figure 4.4 illustrates a tweet that originated a False Positive citation. However, this citation did not originate any annotation because our string matching algorithm did not find any pattern in the correspondent fragment (incorrectly) discovered by Lucene.

Figure 4.5 illustrates a tweet that should have originated a citation but Lucene classified it as if it was not a citation from *The Book of Disquiet*. Figure 4.6 illustrates, in the red box, the annotation that should have been created by this tweet.
4.5 Summary and comparisons

In this section we make an overview of all the advantages and disadvantages of using certain parameters in the different steps of our solution.

Tables 4.16 and 4.17 represent the absolute number and percentage of citations detected by Lucene that latter originated info ranges. For example, by using a Lucene threshold of 20 and a word window of 10 only 23% of the citations that we collected from Twitter were in fact LdoD citations. It is important
to remember that when a citation originates an info range it means that the citation really exists in the fragment text (we make this assumption by the results of 100% of precision from Tables 4.10, 4.11 and 4.12 when we used a word window of > 10). On the other hand, we can not assume this with a word window of > 5 (Table 4.16). As described in Section 4.4, a word window of 5 is not 100% precise, consequently creating incorrect annotations sometimes. Since we want 100% precise results in what concerns the creation of annotations it would not be recommended to use a word window of > 5.

<table>
<thead>
<tr>
<th>Citations that originated Info Ranges</th>
<th>Lucene &gt;20</th>
<th>Lucene &gt;30</th>
<th>Lucene &gt;40</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3072/6621 (46%)</td>
<td>1112/1835 (61%)</td>
<td>718/925 (78%)</td>
</tr>
</tbody>
</table>

Table 4.16: Number of citations that originated info ranges (word window > 5)

<table>
<thead>
<tr>
<th>Citations that originated Info Ranges</th>
<th>Lucene &gt;20</th>
<th>Lucene &gt;30</th>
<th>Lucene &gt;40</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1542/6621 (23%)</td>
<td>908/1835 (50%)</td>
<td>631/925 (69%)</td>
</tr>
</tbody>
</table>

Table 4.17: Number of citations that originated info ranges (word window > 10)

By analyzing 4.16 we conclude that to obtain an higher percentage of useful citations and to not fill the database with useless citations, it would be appropriate to use a Lucene threshold of 40. However, we are not interested in this, because by using a Lucene threshold of 40 it would be too rigorous and we would be discarding useful citations increasing the amount of False Negatives. If we can discard citations that are useless (by running a script that cleans citations that did not originate info ranges) we prefer a Lucene threshold of 20 because it is the one that creates the greatest number of citations that originate info ranges (1542) and we also guarantee that the database is clean over time.

Summing up, by analyzing Twitter4j performance (Section 4.2) we conclude that the most precise keyword while searching for tweets was “livro do desassossego” and the keyword that was responsible for collecting more tweets was “fernando pessoa” (see Tables 4.1 and 4.6). On the other hand, the keyword “vicente guedes” was useless since we did not collect any LdoD citation from Twitter during 6 months.

By analyzing the Results from Section 4.1 we conclude that a Lucene threshold of 20 was the one that originated more citations, therefore more info ranges and consequently more annotations. This threshold is more flexible and tolerant than the others which has the disadvantage of creating more instances of incorrect citations in the database. Nonetheless, this is not a big problem since we can later run a script that cleans citations that did not originate info ranges. Since our goal is to generate the biggest number of annotations in the fragments we conclude that the most appropriate Lucene threshold would be 20. We then tested Lucene and our string matching algorithm performance (Sections 4.2 and 4.3) and concluded that the better combination would be using a Lucene threshold of 20 with a word window of at least 10 words obtaining a Precision of 100% ensuring that every info range and annotation created are in fact citation from *The Book of Disquiet*.
5 Conclusion

Contents

5.1 Summary ......................................................... 65
5.2 Future work opportunities ...................................... 66
5.1 Summary

The main goal of this thesis was to add functionality to the LdoD Archive project. Our solution is called “SAVE” and is composed by two phases.

For the first phase we developed a model responsible for collecting citations from Twitter that cited The Book of Disquiet; a model responsible for mapping the tweets collected to the most likely fragments where the tweet text could appear; and a model responsible for finding a substring of a tweet text in the fragment text, therefore highlighting the text by creating an aware annotation. One of our goals was to maintain a set of meta information associated with each one of the citations such as: the tweet URL, the date and the country (mainly).

For the second phase we dynamically define virtual editions which reconfigure themselves whenever the set of citations change by constructing a way for users to interact with this new concept of socially aware virtual edition (SAVE) based on a set of criteria. The way a user interacts with his own SAVE versions is by choosing the set of criteria to be considered at the moment of creation or even latter. After applying these criteria, the fragments present in a SAVE are modified in terms of annotations, which are added or removed automatically.

In terms of back-end we developed the models responsible for collecting tweets using Twitter4j library, the model responsible for detecting citations using Lucene tool, the model responsible for finding a substring pattern from a tweet text using Jaro-Winkler distance to calculate the similarity between two words and the model responsible for creating aware annotations in the fragments.

In terms of front-end we created new fields (illustrated in Figure 3.11) that allow users to define their own set of social media criteria.

The main contributions of this thesis are:

– Collecting citations of The Book of Disquiet from Twitter

– Developing a substring pattern finding algorithm (based on Jaro-Winkler distance)

– Add sensitivity and awareness to the LdoD Archive of external stimulus from social networks

– Allow users to define their set of social media criteria in order to visualize different meta information from aware annotations

By analyzing the results obtained we conclude that the most effective keywords to search for The Book of Disquiet quotes from Twitter are: “livro do desassossego”, “fernando pessoa” and “bernardo soares”. We also conclude that a Lucene threshold of 20 would be the most appropriate in order to not discard useful tweets obtaining less False Negatives as possible. Choosing a tolerant Lucene threshold of 20 is not a problem if we run our substring matching algorithm with a word window of 10 words, obtaining therefore a Precision of 100% on the aware annotations created.
Summing up, the idea and work done behind this thesis could also be applied to other projects that intend to collect citations from other books and present them to the users, for example.

Last but not least, we wrote an article called “Social Media Aware Virtual Editions for the Book of Disquiet” [4] that was accepted by the European Association for Digital Humanities (EADH)\(^1\). This article is going to be presented in a conference, on the theme “Data in Digital Humanities” at the National University of Ireland, Galway from 7-9 December 2018.

5.2 Future work opportunities

As first mention in Section 3.3.2, one of the future work opportunities consists of adding a module to the LdoD Archive responsible for collecting posts from Facebook. Unfortunately, since Facebook changed its API version from v1 to v2 permanently in 2015 it is not possible to query public posts. However, one could use Facebook4j library to retrieve posts from facebook pages such as: Livro do Desassossego and Livro do Desassossego - Fernando Pessoa (as well as others considered appropriate and likely to post quotes from *The Book of Disquiet*).

Additionally, it would be interesting to collect citations on websites using Google queries. There are several types of websites such as blogs, newspapers, educational websites, personal websites, school websites and social news websites which are more likely to contain LdoD references. From these websites it is possible to extract data, depending of the type of website, about the website’s name, authorship, the author’s gender, his contacts, date of the post, date of the website foundation, geographic location and number of posts.

Another future work opportunity would be improving the string matching algorithm, correcting the problem presented in Section 3.6 concerning the correction of the starting position. This could be achieved by not converting the fragment text to lowercase as described in Section 3.3.4. If we avoid doing that, we could easily find out where the last capital letter from where the pattern was found. In order to preserve the capital letters of the fragment text we would need to reduce the Jaro-Winkler threshold or experiment other similarity distances between words.

\(^1\)https://eadh.org/
Bibliography


