COMP38210 Lab 2 Report
Documents and Data on the Web

Shurong Ge

11/01/2021

Table of Contents

1. Introduction

Resource Description Framework (RDF) plays an important role in search applications. The use of the RDF language facilitates the formation of human-readable and machine-processable documents on the web. It uses a simple and straightforward triples form and an interconnected graph structure that is flexible enough to describe many subjective, distributed, and differently expressed resource objects on the network.
In this lab, I designed an indexing method of RDF documents and learned about the benefits of RDF search. The detailed functionalities and performance of this program are described below.

2. Design

After analyzing the RDF input file, I think someone might want to search season, airdate, guest star, etc about the Simpsons episodes, which means the index should include different data types (i.e. Integer, String, Date). The sentences in the RDF file are supposed to be tokenized so that the tokens are able to be indexed. This reminds me of the last lab, so I refer to the way I deal with the tokens in the previous experiment (Punctuation removal, stemming, stopword removal, case folding). Removing punctuation is necessary, but there is no need to do the stemming since it is meaningless and time-consuming, and although stop words will exist after tokenization, keeping them is better than filtering them since I have tried using the isStopWord() method under StopAnayser, it will take more than 10 minutes to process. I also took case folding into consideration, but it would be different from what I did for the last lab, that is I will not only keep the original form of the tokens but also do the indexing again by turning all the letters in each token into the lower case just in case for some "lazy" searchers (case insensitive). For example, if the user would like to know the writer Bill, he can either search Bill or just bill.

In order to get the output in the form: *(predicate, object), [subject1, subject2, ...]*, which implies that we should put the predicate and object in pairs. The predicate will be an RDF URI reference while the object is likely to be a literal. There are two types of literals, one is plain literal which is "a string combined with an optional language tag [1]" and another is typed literal which is "a string combined with a datatype URI [1]", thus we have to deal with them differently. For plain literals, I will get the lexical form of them using getLexicalForm() and tokenize them in the way that I mentioned above, and I do not think other information like the language code or datatype URI is useful to store so I did not add it in the index; there is nothing to deal with the typed literals so I just index them directly since they will along with their datatype URI automatically.

3. Performance

The program performs as expected, it is able to output the formed result in alphabetical order, capital letter first if both letters are the same, and then the integer/date. It usually takes around 10 seconds to finish to RDFInvertedIndex job, and the total run time is about 27 seconds, so it is efficient.

The majority of searches are allowed in this program. However, there are exceptions. First of all, since we remove all the punctuations, the searchers cannot search word with a hyphen, for instance, there is a word ‘twenty-first’ in 22 Short Films About Springfield, it becomes ‘twentyfirst’ after tokenization. Secondly, this program only works when the language of literal in the RDF file in English, I actually tried with other RDF file with Chinese literal, and it seems not to work.

Compared with searching over web pages, searching over RDF helps users get the relationship between information inside, and more purposeful. For example, if we search 22 on the web page as we would like to know something about *22 Short Films About Springfield*, the web page will offer you millions of results but it is hard to find what we need while the RDF will give the related information.
The index method could still be improved, such as filtering out the stop words or ranking instead of ordering alphabetically.

4. Conclusion

In conclusion, this laboratory gives us an overview of RDF search, I learned that the RDF triples are good for linking different types of data to achieve more accurate searches. My program realized the basic functionalities, there is still a way to improve it.

5. Reference

- Brian, M., Graham, K., Jeremy, J., C. (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. Available at: https://www.w3.org/TR/rdf-concepts/#section-Literals (Accessed date: 11 January 2021).