Web Document Processing

Second Practical Assignment Report

Submitted as part of the requirements for

Natural Language Engineering CE887

**Names:** Eduardo Alberto Sánchez Alvarado

& Édgar Ovidio Villarreal Treviño

**PRIDS:** SANCH26401

& FALTA EL PRID DE OVI AQUI

**Lecturer:** Udo Kruschwitz

**Date:** 10 December 2015

# Table of Contents

1 Installation 1

1.1 Input files 1

1.2 Output files 1

2 Module Discussion 1

2.1 HTML Parser 1

2.2 Sentence Detector 2

2.3 Tokenizer 2

2.4 Part-of-Speech Tagger 2

2.5 Phrase Detector 3

2.6 Ranker 3

2.7 Entity Extracter 3

# Installation

To run the project, we recommend using an IDE such as Eclipse or NetBeans and download it from the repository: <https://github.com/Protossoario/NLEassignment2.git>. As it is a Maven project, upon hitting Run the IDE should run the Maven build, which will in turn download all the dependencies for the project. Note that we had some issues with Eclipse when running our project on the lab computers; for some reason it was printing many characters as whitespace, which made us think that our program wasn’t working, when in reality it was just an issue of the Eclipse console.

## Input files

The input files are specified in the main class, App.java, within the main method, as an array of Strings.

## Output files

The program produces a set of output files for each input file processed, containing sample output from each of the main modules. Then, after it processes each file, it prints the tf.idf rankings into a file with name “tfidfText.txt”. The ranking shows all the noun-phrases found across all documents, ordered by the sum of all their tf.idf rankings, in ascending order (the bottom terms are the most frequent).

# Module Discussion

## HTML Parser

This module takes the name of a HTML file in order to take any important information from it using the library of JSoup. JSoup is a Java library that helps with the extraction and manipulation of data from an HTML file [1]. The HTMLParser gets the text from the HTML file ignoring the HTML tags and gets the information from the content of the “meta” tags.

## Sentence Detector

This module performs the task of pre-processing the text produced by the HTML parser and splitting it into an array of sentences. Each sentence is output as a String in the array. This is done using the OpenNLP class SentenceDetectorNE [2], which is quite effective at separating sentences using periods, while also ignoring periods which could be in the middle of a sentence (e.g. ‘…’).

## Tokenizer

The tokenizer utilizes the Tokenizer module from OpenNLP. It uses a pre-trained model which is freely available for download from their website [3]. This gives it an advantage over a more naïve approach to tokenization, such as using a regular expression; for instance, it can properly distinguish acronyms as a single token rather than separating the letters and dots as individual tokens. On the other hand, the training data used by the Apache Foundation to train the model has its own short-comings, since it doesn’t properly recognize entities such as email address (which it splits by its special characters such as the ‘@’ or ‘.’), and showed some trouble separating phrases like “Poesio/Kruschwitz” into three tokens, since it doesn’t seem to recognize the forward slash as a stop character. However, it still proves fairly useful at distinguishing periods which should belong within a token (e.g. “1.8 m” or “Y.M.C.A.”), and at recognizing things like phone numbers as one single token.

## Part-of-Speech Tagger

The Part-of-Speech Tagger makes use of the POSTagger module from OpenNLP. Like the previous two modules, it uses a pre-trained model that can be found on the website for OpenNLP [3]. This module takes the tokenized text extracted from the HTML file and applies the POSTagger. The module has methods to get the tagged version of the text either in an array of tokens or as a String. It tags the tokens according to their context, but it sometimes fails to properly doing it depending on the punctuation marks as well as recognizing some punctuation marks such as semicolons (;) tagging it as a colon (:).

## Phrase Detector

For this task we utilized the OpenNLP Chunker module, which given a set of tokens and their POS tags, tries to assemble “chunks” of related tokens, like noun groups or verb groups [2]. Then, using those chunks, our module extracts all which are tagged with “NP”, which is the identifier for noun-phrases, produces an array of Strings containing every noun-phrase tagged by the chunker. The resulting module seems to produce good results, being able to identify each comma separated tag as a single noun-phrase, for instance.

## Ranker

The ranker is based on a simple tf.idf implementation, using the output of the phrase detector for the terms it processes. Each individual noun-phrase is treated as a term that can be indexed. For each document we count how many times a phrase appears, and then divide that count by the number of noun-phrase instances in the document. Then, after processing all documents

## Entity Extracter

OpenNLP provides a suite of NER models for extracting several kinds of entities such as people, dates, money, etc. [3] We used several of these models along with the NameFinderME class of OpenNLP, which provides an API for using these models in order to extract entities from an array of tokens [2]. The results had varying degrees of success, depending on the model. For instance, the date entity model proved to be not very useful, since it returned many instances of “noise”, in the form of capitalized words which the model recognized as dates for some inexplicable reason. It also failed to parse the days of simple date formats such as “29th September 2015”, instead returning just the month and year. Overall, the best results seem to have been obtained by the organization entity model (see Appendix A), which are comparable to what could have been obtained by a simple regular expression analysis (e.g. a regex that matches all groups of at least two capitalized words in the middle of a sentence). It also seems to suffer some of the same drawbacks as the person entity model: they both have trouble separating entities which are very close together, and will sometimes clump, for instance, several names of different people into a single entity. This seems to happen often with the person model, though also occasionally with the organization model.

References

|  |  |
| --- | --- |
| [1] | J. Hedley, "jsoup Java HTML Parser," 2015. [Online]. Available: http://jsoup.org/. [Accessed 10 December 2015]. |
| [2] | The Apache Software Foundation, "Developer Documentation," 2014. [Online]. Available: https://opennlp.apache.org/documentation/1.6.0/manual/opennlp.html. [Accessed 10 December 2015]. |
| [3] | The Apache Software Foundation, "OpenNLP Tools Models," [Online]. Available: http://opennlp.sourceforge.net/models-1.5/. [Accessed 9 December 2015]. |