Deutsche Bank
Data Scientist
Programming Task

Document Importation, Term Identification, Named Entity Extraction

By

Obinna Onyimadu

obinna240@yahoo.co.uk
07805117769

2017

Contents

Chapter 1

Setup

1.1 System Setup

- Language, DBs and Frameworks used: JAVA 8, SPRING, LUCENE, MONGO, GATE, JAPE
- OS: Windows ADDITIONAL RESOURCES: Shared dropbox folder
- A jar file: db.jar contains `.class` and `.java` files but **NOT THE CLASS DEPENDENCIES**.

1.2 Directory Setup

1. In C directory, unzip `APP_RESOURCES` into `c:/db`. So that `c:/db` should contain the GATE Processing resource `DB_GATE_APP` folder, `bean.xml` and `sample_news.txt`.
2. `gate.home` environment is already set up in `bean.xml` to point to `c:/db/DB_GATE_APP`. You can open `bean.xml` to verify this.
3. To avoid classpath issues, i have hardcoded all results to go to `c:/db`. You can also run the main methods and enter your own file locations using `java` utility.

Chapter 2

Document Importation

2.1 Converting sample˙news.txt to well formed xml

1. STEP 1: We converted `sample_news.txt` to a well formed xml.
   See `com.db.tasks.parser.UnformedParser.java`. The main method can be run and the resulting xml is written to `c:/db/rewrittenxml1.xml`. You can open the solutions folder in drop box to see a copy of this well formed xml.

2.2 Extracting Documents from XML

Two methods were implemented. One based on a GATE PR which extracts documents, their associated named entities (organizations, locations and persons) and indexes in one go and another which makes use of an XML Parser, without extracting named entities (The documents here can be passed to the GATE PR to extract named entities as well). The latter prints out the named entities onto console, a file and also indexes in Lucene. The former prints just the documents into a directory and indexes the document and named entities in Lucene [1].

2.2.1 Method 1: GATE PR

1. Simply run the main method in
   `com.db.tasks.parser.SavingXmlObject.java`
   This will call `com.db.tasks.parser.SavingXmlObjects` class.
   Each document in sample˙news.txt will be written to console, saved in a directory `c:/db/gate_processed_files` and indexed in a directory `c:/db/index`. The index contains the following fields:

   (a) id: assigned id for each document

   (b) header: document's associated header

[1] We adopted both approaches just to show different approaches. The GATE approach is quite memory intensive. We also created a mongo repositiry which you can run by uncommenting some of the code. Lucene was used to allow us perform search

(c) content: The body of the document

(d) orgainzation: List of organizations

(e) person: list of person names

(f) location: list of locations

2. You can open the GATE PR to see the processing resources used.

3. You can see sample files and written index in the drop box folders `gate_processed_files` and `index`.

2.2.2 Method 2: XML Parser

1. This makes use of a SAX parser to parse the well formed xml.

2. Simply run the main method in `com.db.tasks.parser.XML_Parser.java`. This will save all documents in the xml in the directory `c:/db/xml_parsed_Document_With_Headers`. observe that each document will include its heading, list of named entities, organization, location and persons. The documents are also indexed in a separate lucene index `c:/db/index2`.

3. Indexing for both methods is carried out by methods in `com.db.tasks.search.Indexer.java` and `com.db.tasks.search.SearchByContent.java`.

4. You can find the index in the drop box folder `index2` and the extracted documents in `xml_parsed_Document_With_Headers` folder.

2.3 Searching for Brexit and Trump

Thanks to the Lucene index we can now search for the terms. We implemented two types of queries. Occurrence of terms in the document i.e ‘`term1`’‘`term2`’ and occurrences of phrase “term1 term2”.

2.3.1 Implementation

1. Simply run the main method of `com.db.tasks.search.SearchByContent.java`. This will print out results for all searches: ‘trump’ + ‘growth’, ‘trump growth’, ‘brexit’ + ‘growth’ and ‘brexit growth’. The results will be printed to console and written to two files `c:/db/searchreport/savedReport.txt` – which is the result from the first index and `c:/db/searchreport/savedReport2.txt` from the second index. You can find these files in dropbox folder savedReport.txt and savedReport2.txt.

2. We implemented an additional method to enable you search for ‘locations’, ‘organizations’ and ‘persons’ in `com.db.tasks.search.Indexer` –methods – `searchByName`, `searchByNLocation`, `searchByOrganization`.

3. A search for documents containing 'brexit growth'. Only one document...document 52 contains 'brexit growth'

4. A search for documents containing 'trump growth'. No document was found.

5. Finally, occurrences of 'brexit', 'growth' and these include documents 52, 35, 20, 31, 33, 59, 43, 44, 14, 34. While occurrences of 'trump', 'growth' can be found in document 21, 57, 39, 59, 48, 46, 35, 52, 1, 20.

Chapter 3

LIMITATIONS

3.1 Programming Limitations

1. The db.jar file contains all of the code. I have not included the dependencies because of size and space constraints. You can simply load the pom.xml to build it in any IDE.

2. GATE and JAVA are quite memory intensive and it is not uncommon to experience classpath problems when running projects.

3. Due to time constraints the app is not as tidy as I would have liked. I did not implement logging or text scripts. I was also unable to complete the code for saving into Mongo (Although the DAO can be found in `com.db.tasks.repositories`. To this end named entities for each document were indexed and also printed in files.

4. Due to time constraints, I was unable to implement the sentiment analysis algorithm. An effective and naive approach would be to use the popular sentiwordnet API, to identify the sentiment of each sentence and aggregate the score for each document to assign a polarity to the document.