Exercise 2 May 10, 2018

Please upload your results to the L2P until May 27 15:00 in form of a .zip or .tar.gz archive and provide clear instructions on how to setup and execute your code. Since we are using small datasets in this exercise, please bundle your data within the archive. You may forward any questions to:

Nils Lukas - hendrik.nils.lukas@rwth-aachen.de

Introduction

In this exercise, we are going to mine text documents with Natural Language Processing (NLP) techniques and create RDF Knowledge Graph (KG) representations from them. A number of sophisticated tools already exist for this task and we will try to mirror some of their functionality. You can check out the online demo for `FRED` and `DBpedia Spotlight` to get an introduction to what we want to do. In the presentation of this exercise we will use Java, but you are allowed to use Python for solving the tasks and handing in your solutions.

Language	**NLP**	**RDF store**	**Visualization**
Java	Stanford Core NLP	Apache Jena	Protege-OWLViz or `GraphStream`
Python	SyntaxNet or NLTK	RDFLib	Protege/OWLViz or `graphviz`

Setup

Stanford Core NLP: Make sure you have at least `Java 8` and `Maven` installed. Create a Maven project and add the `Stanford Parser` as a Maven dependency to your pomfile. Other ways of installing are described here: `Installation Guide` . An online demo of the framework can be found here: `Demo` .

Apache Jena: Add `Apache Jena Core` as another Maven dependy to your pomfile.

Data Source: *text1.txt* from the L2P.

Important Links: `Core NLP Reference` , `Core NLP Quickstart` , `Core NLP OpenIE` `Jena RDF` , `Jena ARQ for SPARQL`

Text Mining

Note: For the most part, this is an open programming part. Thus you are allowed to choose your own frameworks and strategies, as long as you obtain the required results to the tasks. For an efficient grading of your submissions, please use the standard output and provide helpful output. Also copy your output and put it into your submission document. Example :

```
System.out.println("-- Task b) Named Entities --")
// ... print entities and tags here ...
System.out.println("-- Task c) Coreference Annotations --")
// ... print unique subjects here
System.out.println("-- Task d) RDF Triples with OpenIE --")
// ... print how many triples were removed and triples that you kept
System.out.println("-- Task e) Entity Resolution --")
// ... print all original name and DBpedia reference
```

a) Briefly explain the functionality of the following Annotators:

- Tokenizer (*tokenize*)
- Sequence Splitter (*ssplit*)
- Part-of-Speech Annotator (*pos*)
- Lemmatization *(lemma)*
- Syntactic Dependency Parser (*depparse*)
- Natural Logic Annotator (*natlog*)
- Open Information Extractor (*openie*)

b) Extract all entities with their named entity tag and all properties. Use the coreference annotation to identify subjects referring to the same entity.

c) Use the coreference annotation to filter out subjects referring to the same individual. For each entitiy, look at their named entity tag and create reasonable triples. For example, you could create the following triples:

```
entity_tag rdf:type owl:Class
entity_tag rdfs:label entity_tag
entity rdf:type entity_tag
...
```

Note: These and all other shown triples serve only as demonstrations. You have to supplement them with correct prefixes etc.

d) Mine the RDF triples using the Information Extraction (OpenIE) annotation pipeline. Filter out only the triples for which you found a subject in task c). For simplicity, also remove all triples that yield a confidence score of less than 100%. For each object property, define the range and domain according to the usage in the text and add corresponding RDF triples of the form:

```
property rdf:type rdf:Property
property rdf:type owl:ObjectProperty
property rdfs:domain domain_entity_tag
property rdfs:range range_entity_tag
...
```

e) An elementary task for generating useful triples lies in referring to existing entities in some ontology and then adding new informations about them. For this, collect all your knowledge about a subject and formulate a dynamic SPARQL query to `DBpedia` and try link to the entity that fits best. Validate the accuracy of your approach by comparing samples to the results obtained from `DBpedias Spotlight` service. Describe your findings.

f Describe briefly how a more sophisticated approach to retrieving triples from text could look like and what data sources it would have to consume.

g) Visualize your RDF as a graph (e.g. by importing it into Protege and using the OWLViz plugin) and if applicable, pick 2 errornous triples and describe why they were created by your program.