Natural Language Processing Class
Course Project Report

Team: KB Analytics
Uri Smashnov

December 6, 2017

Contents

Chapter 1

Introduction

In today's world large amount of data is being generated every day, including humans for humans generated information content. The information content includes news, blogs, academic articles, literature and so on. The information might be stored on the Internet, companies' Intranet, or on access protected document libraries. With such a large information content arises the need to provide efficient search solutions that will allow people to find relevant information fast.
This project work is to implement several Natural Language Processing (**NLP**) techniques in order to demonstrate various strategies for employing efficient and relevant search and retrieval of news. I use Open Source Search Platform SOLR[1] and Reuters news corpus which comes as part of NLTK Python open source library[2].

[1] `http://lucene.apache.org/solr/`

[2] `http://www.nltk.org/nltk_data/`

license: The copyright for the text of newswire articles and Reuters annotations in the Reuters-21578 collection resides with Reuters Ltd. Reuters Ltd. and Carnegie Group, Inc. have agreed to allow the free distribution of this data *for research purposes only*. If you publish results based on this data set, please acknowledge its use, refer to the data set by the name 'Reuters-21578, Distribution 1.0', and inform your readers of the current location of the data set.

Chapter 2

Proposed solution

Throughout this project I use SOLR[1] as my search engine. Each document is being parsed and stored at the sentence level. The project is to demonstrate efficient and relevant retrieval of articles sentences. Each sentence becomes independent SOLR "document". Each sentence is being processed, its relevant features are extracted and stored in SOLR. I then utilize command line interface to provide search and retrieval functionality to fetch top ten matches from the SOLR database.
The process can be summarized as following:

- Loading Articles:
  - Retrieve document and split into sentences
  - Tokenize sentence and extract relevant features
  - Load original sentence and article title information into SOLR along with sentence features
- Command line user interface to search SOLR
  - Parse user input
  - Tokenize user input and extract relevant features
  - Generate SOLR query based on the above
  - Return top ten results provided by SOLR

2.1 SOLR set up

I utilized standard SOLR installation guidelines for Windows 10 machines. SOLR "Core" creation and configuration is managed "progmatically" in the Python code responsible for the processing and loading the news corpus. For each of the three major tasks, separate SOLR "Core" is created. Every SOLR "Core" has following common fields:

[1] `http://lucene.apache.org/solr/`

- **Non-indexed fields**:
  - **article_id**: original data-set's article id. Example:*test/14826*
  - **article_title**: article title extracted from each document
  - **full_sentence**: every sentence in an unprocessed form. This is the search result displayed to user
- **Indexed fields - task 2 (words tokenization only)**:
  - **words**: array of lower case words extracted from the sentence
- **Indexed fields - task 3 (words tokenization plus features)**:
  - **lemmas**: array of lemmas extracted from the sentence
  - **stems**: array of stems extracted from the sentence
  - **words_pos_tags**: array of POS tags extracted from the sentence
  - **head_words**: array of head words extracted from the sentence
  - **phrases**: array of phrases extracted from the sentence
  - Similarities features:
    - **hypernyms**: array of hypernyms extracted from the sentence
    - **hyponyms**: array of hyponyms extracted from the sentence
    - **part_meronyms**: array of meronyms extracted from the sentence
    - **part_holonyms**: array of holonyms extracted from the sentence
  - **Indexed fields - task 4 (plus advanced features)**:
    - **synsets**: array of synsets extracted using LESK algorithm in order to attempt assigning correct word meaning based on the sentence context
    - **hypernyms**, hyponyms, meronyms and holonyms are extracted using identified synset from the previous item

2.2 Documents processing and loading

I have extracted news articles from the Reuters corpus which is part of NLTK corpus database[2]. Only news articles longer than 50 characters have been used, and total of 10751 articles loaded. Each news article has been processed in the following manner using NLTK[3] and SpaCy[4] open source Python libraries.

[2] `http://www.nltk.org/nltk_data/`
license: The copyright for the text of newswire articles and Reuters annotations in the Reuters-21578 collection resides with Reuters Ltd. Reuters Ltd. and Carnegie Group, Inc. have agreed to allow the free distribution of this data *for research purposes only*. If you publish results based on this data set, please acknowledge its use, refer to the data set by the name 'Reuters-21578, Distribution 1.0', and inform your readers of the current location of the data set.

[3] `http://www.nltk.org`

[4] `http://spacy.io`

- Retrieve document, extract its title and split into sentences
- Tokenize sentence and extract relevant features:
  - Lemmas, stems, POS tag feature
  - Syntacticly parse sentence(utilizing SpaCy) and extract phrases and head words
  - Using WordNet, extract hypernyms, hyponyms, meronyms and holonyms as features
- Load sentence to SOLR along with its features

2.3 Content Search And User Interface

I have developed command line interface, in order to provide user with ability to query SOLR with no prior knowledge of SOLR internal DB structure and SOLR query syntax. The command line interface provides with Unix style command line interface and flags. Following options are supported:

```
python search.py -h
usage: search.py [-h] [--type {0,1,2}] [--results_cnt Q_CNT] [--verbose]
search_pattern [search_pattern ...]
search Search words in article database. Example: search --type 1 table chair
positional arguments:
search_pattern        One or more words to search.
optional arguments:
-h, --help            show this help message and exit
--type {0,1,2}, -t {0,1,2}
Search type: 0: basic, 1: advanced, and 2: expert.
--results_cnt Q_CNT, -c Q_CNT
Limit number of results from SOLR query
--verbose, -v         Sets verbose mode. Prints input and generated Solr query
--art_title, -a       Set to skip article title print
```

Flag *–type or -t* directs SOLR query to use functionality developed for task 1,2 and 3, where *basic* option corresponds to task 1, *advanced* to task 2 and *expert* to task 3. The flag is optional and defaults to *basic* option. Once user input is accepted, it is parsed based on the *–type* flag value, and SOLR query is generated accordingly. For example, when *advanced* type selected, the SOLR query will include all additional features listed in section 2.1 *Indexed* fields part, and will be executed against *advanced* SOLR "Core".

2.4 Task 4 - Deeper NLP pipeline features

In addition to features extracted as part of the task 2, described in section 2.1, I have investigated combination of deeper NLP features as well as tweaks in the SOLR search query. List deeper NLP features tried is as following:

- Identify proper meaning (Synset) of the word by implementing simplified LESK algorithm defined in Home work 4. Use it as additional stand along feature. **(Implemented).**
- Utilize Synset identified in previous item to extract hypernyms, hyponyms, meronyms and holonyms. In task 3 we have used default words' Synset **(Implemented)**.
- Utilize *spaCy* functionality Named Entity recognition feature to extract "real" world object; for example, a person's name, a country, a product or a company. **Considered, but not implemented due to time constraint.**
- Utilize *spaCy* "similarity" functionality to rank Solr output. For example, retrieve 100 top matches and use *spaCy* to rank the results. **Considered, but not implemented due to poor results.**

Chapter 3

Implementation: Technical Details

3.1 Tools and Libraries

Following tools and libraries were used for the project:

- Python 3.5 - all coding
- Python *argparse* library for Unix style command line arguments parsing
- NLTK open source Python library - parsing and feature extraction
- spaCy open source Python library - parsing and feature extraction
- Reuters-21578, Distribution 1.0 news articles corpus - processed and loaded into SOLR
- SOLR open source search engine

3.2 Results

The search results improvement in task 2 over task 1 was substantial. Implementation of task 2 features allowed for more of a contextual type of search. The improvement from task 3 implementation was not so evident, and mainly manifested in better ranking of the top results. For example, in below query task 3 ranked more relevant sentence first:
Query:python search.py -t 2 africa development and expansion project

"The very probability of a growth in demand set against the massive investment required for expansion - and that expansion only being viable in South Africa or Russia - leads me to conclude that the price of platinum will be substantially underpinned in the medium to long-term," Clark said.

While task 2 first choice was the following sentence (which appeared fifth on the task 3 search):
Utah's assets include stakes of 40.25 to 52.25 pct in seven large Central Queensland coking mines, 49 pct of the Samarco iron ore operation in Brazil, 60 pct of La Escondida copper deposit in Chile, the Island Copper mine at Port Hardy in Canada, 70 pct of a coal mine and 30 pct of a gold mine in South Africa and coal and other mines in the U.S. BHP Minerals' assets include wholly and partly-owned iron ore mines, coal mines, manganese and base-metal operations or prospects and 30 pct of the Ok Tedi gold-copper project in Papua New Guinea.

3.3 Problems Encountered

Several problems were encountered while working on the project:

- Learning how to set up SOLR was not straight forward. However once the task of creating "Cores" and adding fields with proper attributes was accomplished, working with SOLR became very simple.
- At one point of the project I have encountered performance problem with parsing articles and populating SOLR. The problem was identified and fixed, it was related to overhead of handling exceptions in Python.

3.4 Pending Issues

None.

3.5 Potential Improvements

Potential ideas for improvements:

- Generate 100 matches from SOLR instead of 10. Utilize *spaCy* functionality to calculate "similarity" between returned sentence and input sentence. Sort and display only top 10 matches based on "similarity" value
- Introduce "topic" generation based on sentence content.
- Introduce article features to be added at sentence level.