Natural Language Processing **Summer 2017 Module 5**

Assignment 2

The goal of this assignment is to implement a practical Information Extraction (IE) system. Your implementation will find relationships between musical bands, the music produced by those bands and the people who make up the musical bands.

Requirements

Your implementation will be one or more cooperating object-oriented python modules. It must run on python 2.7. There are four (4) major requirements for your implementation. The module containing the "main" function for your implementation is assumed to be called "`a2.py`" for illustrative purposes in this section.

Requirement 1: Download or use cached Wikipedia pages. Your implementation is allowed to use the Wikipedia pages of the following ten (10) musical bands below, along with any links from those pages to people your model(s) will use for training. The links to the Wikipedia pages are provided as clickable links in the list:

- 10,000 Maniacs
- Belly
- Black Star
- Bob Marley & the Wailers
- The Breeders
- Lupe Fiasco
- Run the Jewels
- Talking Heads
- Throwing Muses
- Tom Tom Club

The second musical band listed (Belly) contains four members: Tanya Donelly, Thomas Gorman, Chris Gorman and Gail Greenwood. Your implementation is allowed to find the individual Wikipedia pages for each of these band members people and use the content of those pages for training. (It is challenging to train on the content of the band pages because they often list names in a way which makes binary relation detection difficult. Training on the person pages may be easier.) Your implementation is not allowed to venture beyond links from the pages in the list above.

By default, your implementation must download the content of the pages listed above from Wikipedia. A command-line option, `cached`, to your implementation may indicate a location on disk where the pages and all links from those pages may be read from disk. (You may want to use the `cached` option when testing since it will reduce the load on Wikipedia and keep you in their good graces.) For example, the implementation may be called with:

```
python a2.py -cached=/home/NLP-data/
```

Your implementation should find the pages in the directory provided by the name of the musical band or person, punctuation removed and spaces replaced by underscores, suffixed with a ".html" string. For example, the cached page for the musical band "10,000 Maniacs" will be `10000_Maniacs.html`, and the cached page for Tanya Donelly will be `Tanya_Donelly.html`. If other data is present in the cached directory, your implementation is not allowed to train on it. If the page(s) required by your implementation are missing, it should end and provide the name of the missing file.

Requirement 2: Process the pages with Beautiful Soup 4. With the pages described above are available, your implementation will extract the HTML to text using Beautiful Soup, version 4 (bs4). Work from the Data Acquisition course should provide examples about how to use bs4.

Requirement 3: Train a MITIE-based Information Extraction system. We have seen examples of how to create an IE system using the MITIE tool. Your implementation must use MITIE for extracting named entities — in this case, persons and bands — and establishing relationships between them. This includes deciding which entities are equivalent to each other. (Musicians such as singer "Natalie Merchant" are often referred to by last name only.) Your implementation may use any approach for this entity resolution, including Levenshtein distance and clustering.

Your implementation may train a custom NER algorithm or re-use an existing one. Regardless of the choices for NER, your implementation will require a custom model for the relationship between bands and people.

Requirement 4: Find band or person. Once your models have been trained on the data above, your implementation will prompt the user for input and use the input to extract additional information and display it to the user. The input will begin with one of three possible commands: ***band***, ***person*** or ***quit***.

The "***band***" command will be immediately followed by the name of a musical band. Given the name of a musical band, your implementation will find and display the names of all the people in that band, past or present. Similarly, the "***person***" command will be immediately followed by the name of a musician. Given the name of a musician, your implementation will find and display the names of all the musical bands to which the person belongs.

The first line of an example of the running implementation is below, with user input in bold and the prompts and replies from the implementation in regular (non-bold). The [...] represents additional text not listed:

```
$ python a2.py -cached=/home/NLP-data/
Find: person John Lennon
Lennon was in Quarrymen
[...]
```

Your implementation may be commanded to provide information on the musical bands on which it was trained. For example: **`band 10,000 Maniacs`** is a legitimate command.

Where the command-line option `cached` has been used (cf. Requirement 1 above), the names of the musical bands and the names of the people will be on disk at the same location as the training data and in the same format as described above — i.e. a page for John Lennon will be available at `John_Lennon.html` in the `/home/NLP-data/` directory.

The "***quit***" command ends the program gracefully.

Implementation Required

Your implementation will be an object-oriented solution to the requirements above. No starting implementation is provided for this assignment, but classes for `band` and `person` are recommended, along with others as necessary.

Any implementation should comply with the Style Guide for Python Code, found online at https://www.python.org/dev/peps/pep-0008/, for the Style portion of the grade for this assignment.

Submission

Submit the source code for the `name_cluster_manager.py` file. You may also add any comments in a README file (text, PDF or Word document) to help the grader understand or execute your implementation.

Grading

Your grade for this assignment will be determined as follows:

- 35% = Required Implementation: your implementation must run successfully with the source files provided. It must produce the output described in this assignment.
- 30% = Performance Implementation: your implementation must produce the coverage expected for the required queries as measured in Micro-Averaged F-measure (MAF).
- 15% = Decomposition: in the eyes of the grader, your solution follow the suggestions above or otherwise must represent a reasonable object-oriented decomposition to this problem.
- 10% = Performance: your implementation must be algorithmically tractable, meaning the training and hypothesis generation must run in a reasonable amount of time.
- 10% = Style: your code must be readable to the point of being self-documenting; in other words, it must have consistent comments describing the purpose of each class and the purpose of each function within a class. Names for variables and functions must be descriptive, and any code which is not straightforward or is in any way difficult to understand must be described with comments. These points and more are described in the Style Guide for Python Code.

Late assignments will not be accepted.