NLTK: Information Extraction

Tingyu Zeng (tzeng11)
CS410, UIUC

- **Topic Category**: Useful software toolkits for processing text data or building text data applications.
- **Topic**: NLTK for information extraction.

Introduction

Information comes with many forms and we can categorize them into structured and unstructured ones. Extracting information from unstructured data, such as human language texts, can be a challenge for machines due to their complexity and ambiguity; while querying from structured data, such as a tabular form, is more straightforward.

Information extraction is a method of retrieving meaning from texts, by first converting unstructured data into structured machine-readable documents so that it can be processed and understood by machine query tools like SQL.

A simple pipeline architecture for an information extraction system consists of 5 parts:

1. The input, the free and raw text, is segmented into sentences.
2. The sentences get further tokenised into words.
3. The system tags each sentence with part-of-speech tags.
4. The named entities are detected in each sentence.
5. The possible relations between different entities in the text are discovered.

NLTK, Natural Language Toolkit, is an excellent resource for general work in corpus linguistics. It provides useful APIs to tackle all 5 tasks in information extraction mentioned above.

Preprocessing

The first 3 tasks from the simple pipeline are commonly seen in NLP related workflows. They parse the sentences and attach labels to each word, preparing for the next steps.

While NLTK has provided recommended algorithms for these tasks by default, it is still possible to perform customization based on the project. Preprocessing the input from task 1 to 3 can be easily achieved by NLTK's default sentence segmenter (`PunktSentenceTokenizer`), word tokenizer (`TreebankWordTokenizer` and `PunktSentenceTokenizer`) and part-of-speech tagger (`PerceptronTagger`).

Named Entity Recognition

Named entities are those special noun phrases that clearly distinguishes one item from the others that have the same or similar attributes. Normally they refer to specific brands, organizations, persons, dates and so on. The goal of named entity recognition is to identity these noun phrases: their boundaries and types.

Chunking

We often use the technique of *chunking* with NLTK to further categorize the pos tags retrieved from last steps into non-overlapping linguistic groups. For instance, with POS tagging we label the words with categories like *nouns*, *pronouns*, *adjectives*, *verbs* and etc; with these pos tags as input, chunking produces chunks that should be understood together in its context, such as *"South Africa"*. Note that one chunk should not contain another chunk of the same type.

The rules that define the boundaries of different chunks are called chunk grammar. The most commonly seen method is the regular expression rule and NLTK has native support for it with its `chunk` module.

We can use either one single regular expression rule or multiple ones to parse the pos tags into chunks. Normally the rule describes a certain tag pattern that one chunk should follow. For instance, we can define that a noun phrase chunk should contain the following tags in order: `<DT>?<JJ>*<NN>`. It means that a noun phrase can optionally start with a determiner (`DT`) followed by 0 or more adjectives (`JJ`) and always ends with a noun (`NN`).

Other than the regular expression chunk parser, we can also use n-gram chunkers and classifier-based chunkers.

Besides tree graphs we can use IOB tags to represent chunk structures: each token is assigned one of the 3 chunk tags - `I` (inside), `O` (outside) and `B` (begin). For instance, chunking the sentence*"I saw the yellow submarine"* into noun phrases would be represented as:

```
I PRP B-NP
saw VBD O
the DT B-NP
yellow JJ I-NP
submarine NN I-NP
```

An alternative to chunking is *chinking*: instead of grouping tags together, sometimes it is easier to define and execute the rules to remove what should not be included in the chunk.

NLTK provides a dataset `conll2000`, containing 270k words of Wall Street Journal in IOB format, to train and evaluate the chunkers.

Identifying named entities

Once we labelled the text with chunk tags, it is easier to identify named entities. We may need to rely on well recognized dictionary to look up each item in a list of names, such as *gazetteer* for geography.

However, this approach can be prone to errors since a single word may refer to another meaning in a different context and an entity's name may contain multiple words.

In practice, classifier-based approach performs well with named entity recognition. You can build your own classifier based on your need or just use the one that NLTK has trained and provides with the function `nltk.ne_chunk()`.

Relation Extraction

As the final step, we look for interesting patterns in the text that may reveal some relation between entities. NLTK can perform such a task by either a rule-based system or a machine-learning system.

In general, the task can be seen as seeking patterns like `(X, a, Y)` where `X` and `Y` are named entities of certain types and `a` is the string between them.

We can use regular expression to detect such patterns. For instance, we can try to look for a group of named organizations and named locations which are connected by the word `in`. Such a pattern might reveal the address of the organization for us.

We can also use machine learning to train a model that detects the possible relations between these two named entities from a training corpus, and use it for future prediction.

References

1. Extracting Information from Text, *Natural Language Processing with Python*
2. Learning POS Tagging & Chunking in NLP, Jocelyn D'Souza
3. Chunking with NLTK, sentdex
4. NLP | Classifier-based Chunking
5. Named Entity Recognition with NLTK and SpaCy, Susan Li