Named Entity Recognition in the WSJ

Brendan Hart

December 5, 2016

1 Introduction

This report will discuss the ideas used to create a system capable of performing Named Entity Recogniton, particularly in the WSJ. The system is able to be loaded by importing it either in the Python shell or a Python file. After this, the user has acccess to the `NERTagger` class, with the most important methods being `train` for training the system, `tag` for tagging named entities in texts and `test` for testing the system. `stitch` can be used for turning the result of `tag` into a document format.

2 Preprocessing

The first task was turning a text document into a useful form which could then be stitched back together with the named entities tagged. The system initially used `nltk.sent_tokenize`[1] to turn the text into a list of sentences, and then use `nltk.word_tokenize`[1] to create a list of tokens for each sentence. After this, it would then proceed to assign a part of speech (POS) tag each list of words. It was decided that `nltk.pos_tag` would be used, as evaluations carried out in an article on StreamHacker show that it is quite an accurate tagger, potentially reaching 98% accuracy whilst maintaining speed, achieving 483 words/sec[2]. This system meant that losing the new line characters since `nltk.word_tokenize` removes them, meaning the document could not be accurately reassembled. The solution used was to first split the text at a `\n` character, and then carrying out the above process on the result. This allows to easily preserve the new line characters as well as having sentences separated and then POS tagged. Some more processing was carried out on the result of this, such as ensuring days of the week and month abbreviations were not tagged as NNP, ensuring they are not later classified as an organisation, person or location.

3 Chunking

The next step was to now extract interesting parts of the text. A full parse of the text would be computationally expensive and give more information than was needed. Since only the named entities were of interest, chunking would be a better solution. To do this, NLTK's implementation of a regular expression-based parser called `nltk.RegexpParser` was used. When creating an instance of the class, it takes as an argument the grammar rules to be applied. The system used five rules, which were:
`NE: {<NNP>.?>+}` to catch entities such as "Brendan Hart"
`NE: {(<NNP.?>|<NE>)<PERIOD><NE>}` to catch entities such as "B.H"
`NE: {<NE><AMP><NE>}` to catch entities such as "S&P"
`NE: {<NE><POS><NE>}` to catch entities such as "Brendan's Company"
`NE: {<NE><IN><NE>}` to catch entities such as "United States of America"
The system then looped through each sentence and chunked each of them, replacing the position in the array of the original sentence with the newly chunked version which were parse trees of the type `nltk.Tree`.

4 NER Classification

Now we have a way to extract the named entities from the text by looping through the parse trees and finding the subtrees which have a label of "NE". The system contains a method to flatten a given tree into a list of text, so a tree defined as `nltk.Tree("NE", [nltk.Tree("NE", ["United", "States"]), "of", nltk.Tree("NE", ["America"])])` would become "United States of America", ready to be classified.

4.1 Handwritten Rules

Each of these flattened named entities need to be classified as organisation, person or location. There were some hand written rules in the system that will classify some of the named entities with these distinctive features. These included tagging entities as organisations if they contain certain words, such as "corp.". A handwritten rule also attempted to classify some locations, which attempted to match named entities such

as "123 Fake Street".

4.2 DBpedia

If none of the handwritten rules matched, DBpedia was used to retrieve the labels which will, ideally, indicate which class each named entity belongs to. DBpedia is a community effort to turn data from Wikipedia into a structured, accessible form. Since Wikipedia is a large collection of information, it seems logical that DBpedia would be an ideal tool for NER considering the classifications of queries and was shown to have a noticeable increase in accuracy[3].
The system makes a request to DBpedia with the named entity appended to the URL[4]. A regular expression `<Label>(.+)</Label>` is then used to retrieve a list of all the text enclosed a label tag. Organisations would have a label "organization", people would have a label "person" and locations would have the label "location" or "place". Although, some named entities may be ambiguous and contain labels such as "location" and "organization". The solution used was to take the first label which matches a class to resolve these conflicts.
Once the class of the named entity had been determined, the original tree with label "NE" was replaced with a new tree which had the label of the class and the named entity, such as `nltk.Tree("LOCATION", ["United States of America"])`.

4.3 Naive Bayes

Whilst DBpedia seemed effective, some named entities were not able to be found. Therefore, a backup method of classification was needed. It was decided a Naive Bayes classifier which would be used when DBPedia could not determine the type of a given named entity. The features used were on a token level rather than the named entity phrase as a whole, with inspiration from Tkachenko and Simanovsky (2012)[3], in hope this would pick up on some of the unique tokens to a distinct class or tokens which follow certain classes. The features were current token, next token, whether it has any "person" features or any "location" features, as well as the previous token's entity classifcation from the same named entity phrase. Organization features did not seem useful as many of the most common features of an organisation are already covered by the handwritten rules. The classifier then chooses the most frequent classification of the words in the entity as the class for the named entity phrase.

5 Postprocessing

The final step was to produce a document from the tagged data which is currently in list of lists of trees form. This essentially involved undoing the steps took before the classification. A method `stitch` can be used to turn the result into such a format. This works by first flattening each sentence tree into a string using `flatten` which is also used in various other methods as well as `stitch`. This a similar position to splitting after a new line character, therefore the next step is to use `"\n".join()` with the argument as the previous result. This resembled the original document (now with tagged named entities) but some additional steps were in an attempt make the output correct.

6 Evaluation

The system was trained with 70% of the data and tested on 30% of the data. For comparison, the results of tagging every entity on the same 30% as an organisation are also displayed. Two types of tests were carried out, one for when the named entities were extracted from the text perfectly and their classification was correct and another for when, for example, "United States" was picked up and classified correctly but "United States of America" was the target.

	Identical	Loose
Default	40.57%	51.06%
DBpedia	54.83%	9
DBpedia w/ NB	8	9

References

[1] (2016) nltk.tokenize package. Available at: http://www.nltk.org/api/nltk.tokenize.html

[2] Perkins, J. (2010) Part of speech tagging with NLTK part 4 Brill Tagger vs Classifier Taggers. Available at: http://streamhacker.com/2010/04/12/pos-tag-nltk-brill-classifier/ (Accessed: 4 December 2016).

[3] Tkachenko, M. and Simanovsky, A. (2012) Named Entity Recognition: Exploring Features, .

[4] (No Date) DBpedia look-up example. Available at: http://lookup.dbpedia.org/api/search/KeywordSearch?QueryString=Microsoft (Accessed: 4 December 2016).