Lab 4: POS Tagging

You can use the online version of this lab if you want to easily access the links or copy and paste commands.

Goals and motivation of this lab

Part-of-Speech (POS) tagging is the task of identifying the part-of-speech for each word in a given text. 'Tagging', because the symbol used to label part-of-speech is called a 'tag'. Tagging is one of the basic steps in developing Natural Language Processing (NLP) tools such as parsers, question answerers etc. In this lab, we will explore POS tagging and build a simple POS tagger.

As before, we have written most of the code for the lab already and included a lot of explanatory comments, but we will ask you to add a few things here and there. For students with more programming background, the 'Going Further' section will give you a chance to explore some more advanced topics.

Preliminaries

As usual, create a directory for this lab inside your `labs` directory:

```
cd ~/anlp/labs
mkdir lab4
cd lab4
```

Download the file lab4.py into your `lab4` directory: From the Lab 4 web page, right-click on the link and select *Save link as...*, then navigate to your lab4 directory to save.

Open the file with gedit and start up ipython:

```
gedit lab4.py &
ipython
```

NLTK

In this lab, we use NLTK library. NLTK (Natural Language Toolkit) is a popular library for language processing tasks which is developed in python. It has several useful packages for NLP tasks like tokenization, tagging, parsing etc. In this lab, we use very basic functions like loading the data, reading sentences. You will be asked to write few lines of code in a fuction to answer some questions in this lab. Some of these functions are already implemented in NLTK. You can explore those options for questions in 'Going Further' section as Home work.

POS Tagset

POS tagset is the set of Part-of-Speech tags used for annotating a particular corpus. Penn Tagset is one such tagset which is widely used in NLP tasks. Have a look at the tagset. Based on your intution guess the most and least frequent tags in a data. What is the difference between the tags DT and PDT? Can you distinguish singular and plural nouns using this tagset? If so, how? How many different tags are available for main verbs and what are they?

Running the code

You can run this week's lab as follows:

```
%run lab4.py
```

It should print the first tagged sentence, and first word, tag tuple. It also should print basic statistics about the data like total number of sentences and the average sentence length (number of words per sentence).

For subsequent runs, you can skip the printing by doing:

```
%run lab4.py -q
```

For this lab, we consider a small part of the Penn Treebank POS annotated data. This data consists of around 3900 sentences, where each word is annotated with its POS tag using the Penn POS tagset. We use nltk libraries to load this data and extract tagged sentences. We first import the `dependency_treebank` from nltk.corpus package using the command `from nltk.corpus import dependency_treebank`. We can extract the tagged sentences using the following command:

```
tsents = dependency_treebank.tagged_sents()
```

`tsents` contains a list of `tagged sentences`. A `tagged sentence` is a list of pairs, where each pair consists of a word and its POS tag. A pair is just a Tuple with two members, and a `Tuple` is a data structure that is similar to a list, except that you can't change its length or its contents. The Python Tuple documentation provides a useful summary introduction to tuples.

Once you've loaded lab4.py, `tsents[0]` contains the first tagged sentence. `tsents[0][0]` gives the first tuple in the first sentence which is a word, tag pair, and `tsents[0][0][0]` gives you the word from that pair, `tsents[0][0][1]` its tag.

Distribution of sentence lengths

Construct a frequency distribution of sentence length by completing the code in the `sent_length_distribution` function, which returns a dictionary with sentence lengths as keys and the number of sentences with a particular length as values. Use the `plot_histogram` function to plot this distribution, sorted in order of sentence length. Remember to expand the plot window so you can see what's going on, and to close it when you want to get back to ipython.

(Don't forget to do:

```
%run lab4.py
```

after you've done some edits, in order to get your new function definitions.)

What are the minimum and maximum sentence lengths that you see? What kind of distribution is this?

Distribution of tags

Construct a frequency distribution of POS tags by completing the code in the `tag_distribution` function, which returns a dictionary with POS tags as keys and the number of word tokens with that tag as values. How many distinct tags are present in the data? List the tags in the descreasing order of frequency. What are the 5 most frequent and least frequent tags in this data. How does this compare with your intution in the previous section? Using `plot_histogram`, plot a histogram of the tag distribution with tags on the x-axis and their counts on the y-axis, ordered by descending frequency.

In the previous labs, you have learnt how to sort dictionary *keys* based on their *values* (eg: `sorted(mydict, key=mydict.get)`). You can adapt this approach to return a sorted list of keys *and* values, which is what you need for the above plot. For example, the following command returns list of dictionary entries sorted by values:

```
sorted(mydict.items(), key=lambda x: x[1])
```

Providing `reverse=True` as the third argument gives the result in reverse order. `help(sorted)` is your friend.

What kind of distribution do you see in the plot?

Distribution of tags and words

Construct a conditional frequency distribution (CFD) using the `word_tag_distribution` function. A CFD is a dictionary whose values are themselves distributions, keyed by context or condition. In our case we want words as conditions == keys, with values a frequency distribution of tags *for that word*.

For example, for the word *book*, the value of your CFD should be a frequency distribution of the POS tags that occur with *book*.

How many entries are there in your big CFD? Given what each entry corresponds to, what does this number tell us about the corpus?

What is the number of tags and the most frequent tag for the word *the*? Does what you see make you wonder about the tagging process, and how it was checked?

What about *in*, *book*, *eat*? Which word out of the whole corpus has the greatest number of distinct tags? (Hint: fill in the blank:

```
pprint(sorted(word_tag_dist.items(),key=lambda (w,fd):_________,reverse=True)[:10])

(``pprint`` (think *pretty print*) is often useful if you have lots
of nested dictionaries, lists and tuples to look at. . .)
```

What are the tags which occur with the tag-ambiguous word and what parts of speech do they represent?

NLTK Frequency Distribution functions

NLTK has two in-built functions FreqDist() and ConditionalFreqDist() which compute frequency distribution and conditional frequency distribution respectively. These functions are similar to the ones we just implemented. They have several additional useful features, such as `tabulate()`, which prints a frequency distribution in tabular form and `plot()`, which plots it in inverse frequency order.

For these functions, we need a single long list of word and tag pairs. `dependency_treebank.tagged_words()` can be used to get such a list from our corpus:

```
wtPairs = dependency_treebank.tagged_words()
```

Using `FreqDist` and a simple comprehension, we can then get a frequency distribution for just the words in our corpus:

```
wfd=FreqDist(w for (w,t) in wtPairs)
wfd
wfd.plot(50)
```

And finally using `ConditionalFreqDist()`, we can compute the conditional frequency distribution of word and tag pairs for our corpus like this:

```
cfd = ConditionalFreqDist(wtPairs)
the_fd = cfd['the']
the_fd
len(the_fd)
the_fd.tabulate()
```

`cfd['the']` gives the frequency distribution of tags for the word *the*. `fd.tabulate()` prints this distribution in tabular form. Compare this with the conditional frequency distribution computed in the previous section, for the word *the*. Both should give same results.

Unigram Tagger

We shall now build a simple POS tagger called a *unigram* tagger using the function `unigram_tagger`. This fuction takes three arguments. The first one is a conditional frequency distribution, which can be generated using the nltk functions described above. The second argument is the most frequent POS tag. The third argument is a sentence that needs to be tagged. The goal of this function is to tag the sentence using probabilities from the CFD and most frequent POS tag. In order to make it work, you must complete its helper function, called `ut1`, to process a single word. If the word is seen (present in the CFD), it should assign the most frequent tag for that word. For unseen words (not present in the CFD), it should assign the overall most frequent POS tag, as passed in as the 3rd argument.

Why is this called a Unigram tagger? How does it differ from an HMM tagger?

Run this simple tagger and tag the sentences a) "book a flight to London" and b) "I bought a new book". Look at the POS tags. Are there any errors in the POS tags? If so, what could be the reason for them?

Going further

1. Run the simple tagger developed on the sentence "I bought two new books." What error do you see, and what improvement can you think of that can handle it?

2. Do you think the Penn tagset will work well for social media text such as twitter data which contains non-standard English text?

3. NLTK has libraries to train different taggers. Using these libraries, build unigram and Hidden Markov Model taggers and evaluate them. First, split the data into two parts(90%, 10%). Consider first 90% of the data as training data and the remaining 10% of the data as testing data. Build taggers using the training data. Then run the taggers on the test data and evaluate the performance of the tagger

`help(nltk.tag.hmm.HiddenMarkovModelTagger)` and `help(nltk.UnigramTagger)` are your friends. Note particularly the `evaluate` function for evaluating the tagging of a collection of test sentences.