Faculty of Computer Science, Dalhousie University *26/28-Feb-2020*

CSCI 4152/6509 — Natural Language Processing

Lab 6: Python NLTK Tutorial 2

Lab Instructor: Dijana Kosmajac, Tukai Pain
Location: rm1102, Mona Campbell building
Notes author: NLTK Book, Dijana Kosmajac and Vlado Keselj

Python NLTK Tutorial 2

This is the second NLTK tutorial. Remember that NLTK (Natural Language Toolkit) is a platform for building Python programs to work with human language data (Natural Language Processing). We will also repeat a couple of useful URLs related to NLTK:

For installation instructions on your local machine, please refer to:
`http://www.nltk.org/install.html`
`http://www.nltk.org/data.html`

For a simple beginner Python tutorial take a look at:
`http://www.tutorialspoint.com/python/python_tutorial.pdf`

Lab Overview

– Part-of-speech (POS) taggers: HMM and CRF, Brill;
– Tree model and Text chunker for capturing;
– Named-entity recognition (NER);
– Jupyter and JupyterHub

Files to be submitted:

1. `hmm_tagger.py`
2. `crf_tagger.py`
3. `brill_demo.py`
4. `ne_chunker_exercise.py`
5. `first_notebook.ipynb`

NLTK Tagging and Chunking Overview

Representing Tagged Tokens

By convention in NLTK, a tagged token is represented using a tuple consisting of the token and the tag. We can create one of these special tuples from the standard string representation of a tagged token, using the function `str2tuple()`:

```
from nltk.tag import str2tuple

tagged_token = str2tuple('fly/NN')
print(tagged_token)
```

```
# ('fly', 'NN')
print(tagged_token[0])
# 'fly'
print(tagged_token[1])
# 'NN'
```

We can construct a list of tagged tokens directly from a string. The first step is to tokenize the string to access the individual word/tag strings, and then to convert each of these into a tuple (using `str2tuple()`).

```
from nltk.tag import str2tuple

sent = '''
    The/AT grand/JJ jury/NN commented/VBD on/IN a/AT number/NN of/IN
    other/AP topics/NNS ,/, AMONG/IN them/PPO the/AT Atlanta/NP and/CC
    Fulton/NP-tl County/NN-tl purchasing/VBG departments/NNS which/WDT it/PPS
    said/VBD "/" ARE/BER well/QL operated/VBN and/CC follow/VB generally/RB
    accepted/VBN practices/NNS which/WDT inure/VB to/IN the/AT best/JJT
    interest/NN of/IN both/ABX governments/NNS "/" ./.
    '''
print([str2tuple(t) for t in sent.split()])
# [('The', 'AT'), ('grand', 'JJ'), ('jury', 'NN'),
# ('commented', 'VBD'),
# ('on', 'IN'), ('a', 'AT'), ('number', 'NN'), ... ('.', '.')]
```

Reading Tagged Corpora

Several of the corpora included with NLTK have been tagged for their part-of-speech. Here's an example of what you might see if you opened a file from the Brown Corpus with a text editor:

```
The/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd
Friday/nr an/at investigation/nn of/in Atlanta's/np$ recent/jj
primary/nn election/nn produced/vbd ``/`` no/at evidence/nn ''/''
that/cs any/dti irregularities/nns took/vbd place/nn ./.
```

Other corpora use a variety of formats for storing part-of-speech tags. NLTK's corpus readers provide a uniform interface so that you don't have to be concerned with the different file formats. In contrast with the file fragment shown above, the corpus reader for the Brown Corpus represents the data as shown below. Note that part-of-speech tags have been converted to uppercase, since this has become standard practice since the Treebank Corpus was published.

```
from nltk.corpus import brown

print(brown.tagged_words())
# [('The', 'AT'), ('Fulton', 'NP-TL'), ...]

print(brown.tagged_words(tagset='universal'))
# [('The', 'DET'), ('Fulton', 'NOUN'), ...]
```

Exploring Penn-Treebank corpus

The Treebank corpora provide a syntactic parse for each sentence. The NLTK data package includes a 10% sample of the Penn Treebank (in `treebank` package), as well as the Sinica Treebank (in `sinica_treebank`).

Some NLTK corpora are not available with the initial install of the library. Treebank is such a copurs. To install the NLTK corpus run Python 3 console with:

```
python3
```

and then add the following lines:

```
import nltk
nltk.download('treebank')
quit()
```

Reading the Penn Treebank (Wall Street Journal sample):

```
from nltk.corpus import treebank

print(treebank.fileids()) # doctest: +ELLIPSIS
# ['wsj_0001.mrg', 'wsj_0002.mrg', 'wsj_0003.mrg', 'wsj_0004.mrg', ...]

print(treebank.words('wsj_0003.mrg'))
# ['A', 'form', 'of', 'asbestos', 'once', 'used', ...]

print(treebank.tagged_words('wsj_0003.mrg'))
# [('A', 'DT'), ('form', 'NN'), ('of', 'IN'), ...]

# doctest: +ELLIPSIS +NORMALIZE_WHITESPACE
print(treebank.parsed_sents('wsj_0003.mrg')[0])
    (S
      (S-TPC-1
        (NP-SBJ
          (NP (NP (DT A) (NN form)) (PP (IN of) (NP (NN asbestos))))
          (RRC ...)...)...)
        ...
        (VP (VBD reported) (SBAR (-NONE- 0) (S (-NONE- *T*-1))))
        (. .))
```

Note: If you have access to a full installation of the Penn Treebank, NLTK can be configured to load it as well. For your own purposes, you can download the `ptb` *package, and in the directory* `nltk_data/corpora/ptb` *place the BROWN and WSJ directories of the Treebank installation (symlinks work as well).*

Ready-made POS Tagger

```
from nltk import tag

sent = ['Today','you',"'ll",'be','learning','NLTK','.']
tagged_sent = tag.pos_tag(sent)
print(tagged_sent)
# [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'),
#  ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')]
```

The Penn Treebank POS tags are covered in the class, and a list of tags without punctuation is also available at:
http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Ready-made NE (Named Entity) Chunker

```
from nltk import chunk, tag

sent = ['Today','you',"'ll",'be','learning','NLTK','.']
tagged_sent = tag.pos_tag(sent)

tree = chunk.ne_chunk(tagged_sent)
print(tree)
# Tree('S',
#   [('Today', 'NN'), ('you', 'PRP'),
#    ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),
#    Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')])

ne_subtrees = tree.subtrees(filter=lambda t: t.label() in
   ["ORGANIZATION", "PERSON", "LOCATION", "DATE", "TIME",
    "MONEY","PERCENT", "FACILITY","GPE"])
ne_subtrees_list = [tree for tree in ne_subtrees]
print(ne_subtrees_list)
# [Tree('ORGANIZATION', [('NLTK', 'NNP')])]

ne_phrases = [' '.join(word for word, pos in tree.leaves()) for
                      tree in ne_subtrees_list]
print(ne_phrases)
# ['NLTK']
```

Step 1. Logging in to server bluenose

- Login to the server bluenose
  As in previous lab, login to your account on the server bluenose.
- Change directory to csci4152 or csci6509
  Change your directory to csci4152 or csci6509, whichever is your registered course. This directory should have been already created in your previous lab.
- Create the directory `lab6` and change your current directory to `lab6`:
  ```
  mkdir lab6
  cd lab6
  ```

Step 2. Training HMM POS tagger

You have learned about Hidden Markov Models (HMM) in the lecture. Now, you will learn how to use NLTK to train HMM POS tagger using `treebank` corpus. Create a file called `hmm_tagger.py`. Type the following code:

```
# Import the toolkit and tags
from nltk.corpus import treebank
# Import HMM module
from nltk.tag import hmm

# Train data - pretagged
train_data = treebank.tagged_sents()<slice_first_3000>
# Test data - pretagged
```

```
test_data = treebank.tagged_sents()<slice_other_than_first_3000>

print(train_data[0])

# Setup a trainer with default(None) values
# And train with the data
trainer = hmm.HiddenMarkovModelTrainer()
tagger = trainer.train_supervised(train_data)

print(tagger)
# Prints the basic data about the tagger

print(tagger.tag("Today is a good day.".split()))

print(tagger.tag("Joe met Joanne in New Delhi.".split()))

print(tagger.tag("Chicago is the birthplace of Marry".split()))

print(tagger.evaluate(test_data))
```

The code above has syntax errors. It is supposed to split the data into train and test sets. Using Python list slicing ([:] notation) split the data in a way so the train corpus has first 3000 samples and test corpus has the rest.

Additionally, add a comment below the print statement where the POS tags are wrong and specify what would be the correct tag.

Submit: Submit the script `hmm_tagger.py` which you just wrote using the `submit-nlp` command.

Step 3. Training CRF POS Tagger

First, you need to install the CRF suite for Python with:

```
pip3 install --user python-crfsuite
```

Now we will see a similar structure for training Conditional Random Field (CRF) based tagger:

```
# Import the toolkit and tags
from nltk.corpus import treebank

# Import CRF module
from nltk.tag import crf

# Train data - pretagged
train_data = treebank.tagged_sents()<same_as_previous>

# Train data - pretagged
test_data = treebank.tagged_sents()<same_as_previous>

# Setup a trainer with default(None) values
# Train with the data
tagger = crf.CRFTagger()
tagger.train(train_data,'model.crf.tagger')
```

```
print(tagger)
# Prints the basic data about the tagger
print(tagger.evaluate(test_data))
# <your comment section>
#
#
```

Submit: Save the previous code as the script `crf_tagger.py`. You first need to correct the code (the data splitting using slice), comment what is the result of evaluate method for the CRF tagger and previous HMM tagger. Which performed better? Submit the file `crf_tagger.py` using the `submit-nlp` command.

Step 4. Brill Tagger Demo

Brill tagger is different than other part of speech taggers. The Brill tagger uses the initial POS tagger (can be a dummy tagger which assigns same label to every token, or other) to produce initial part of speech tags, then corrects those POS tags based on Brill transformational rules. These rules are learned by training the Brill tagger with a trainer algorithm and rules templates.

The next example is a built-in demo of Brill tagger training on Treebank Corpus. `postag` function will output the details about the run. Take a look at the output and answer the following questions as a comment in the source. What is the accuracy? What are the three top rules? Is it better than HMM and CRF taggers we trained earlier?

```
from nltk.tbl.demo import postag

postag(num_sents=None, train=0.7665)
# if we set num_sents to None, it will use the whole treebank corpus.
# We want this, so we can compare the results to the CRF and HMM we
# tested earlier. If we set train ratio to 0.7665, the train set will have
# 3000 sentences, just like in previous taggers. The other params are default.

# <your_comments_here>
#
#
```

If you would like to take a look how the built-in demo is set up, and if you wish to tweak other parameters, take a look at the source: `http://www.nltk.org/_modules/nltk/tbl/demo.html`

Submit: Save the previous code as the script `brill_demo.py`. Add your answers to the question in the comments part and submit the file `brill_demo.py` using the `submit-nlp` command.

Step 5. Named Entity Chunking Task

In this task you will be asked to write some code for Named Entity extraction from `treebank` corpus. You can find below the structure and some helpful comments. Please refer to the example from page 3 how to do chunking and selecting relevant subtrees. Also, remember, this example deals with a list of sentences, and at some point you will have to flatten the list (using list comprehension). You should print three most common named entities in `treebank` corpus.

```
from nltk import FreqDist
# import treebank
```

```
# import ne chunker

data = # load treebank data
chunkd_data = # chunk the data
chunkd_trees = # select subtrees which are NE

word_fd = FreqDist([' '.join(word for word, pos in tree.leaves()) for
                                              tree in chunkd_trees])

print("Three most common named entities are: ")
for token, freq in word_fd.most_common(3):
    print("%s : %d"%(token, freq))
```

Submit: Save the previous code as the script `ne_chunker_exercise.py` and submit it using the `submit-nlp` command.

Jupyter Overview

Before we move to the final step of the lab, which involves the Jupyter system, let us first introduce the concept of Jupyter and Jupyter notebook.

Jupyter notebook: A "notebook" or notebook document represents document that contains code and rich text elements combined, such as images, links, math equations, etc. Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis with textual description, and its results. On top of it they can be executed to perform the data analysis in real time.

The Jupyter Notebook App helps to produce these documents.

Jupyter is an acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.

Jupyter Notebook App: Jupyter Notebook App is a server-client application that allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access, or it can be installed on a remote server, where you can access it through the Internet. We will briefly describe later how to use preinstalled Jupyter on Bluenose. If you wish to explore the installation options, please refer to the official documentation (`https://jupyter.org/install`).

Its two main components are the kernels and a dashboard.

A kernel is a program that runs and introspects the users code. The Jupyter Notebook App has a kernel for Python code, but there are also kernels available for other programming languages.

The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can see which ones are running and shut them down if necessary.
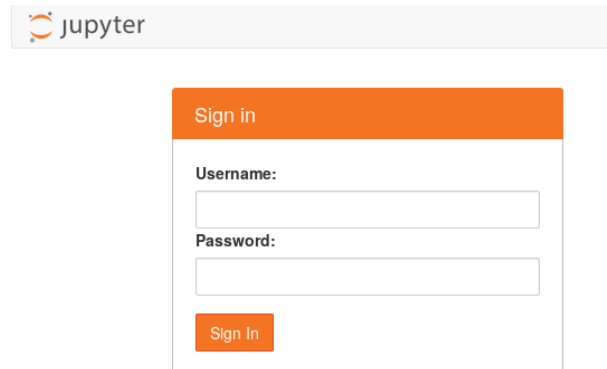
You can look into installing Jupyter on your computer and experiment with it. However, you can also try it on the server Bluenose, using a secure app called JupyterHub. Whenever you logged in to Bluenose, you probably noticed this message:

```
JupyterHub is now available for running Jupyter notebooks. Log in with your
CSID at https://bluenose.cs.dal.ca:8000
```

which tells you how to use JupyterHub on Bluenose. However, it seems that at the moment JupyterHub is not working completely correctly on Bluenose, so you will do the next step using the new server Timberlea.

Step 6. Using JupyterHub on Timberlea

Open the JupyterHub on Timberlea using a similarURL that we saw in the previous section:
`https://timberlea.cs.dal.ca:8000`. You should see the following page:

Use your CSID and password to login, and then you should see the contents of your home directory on bluenose.
The top of the screen should look something like this:

Find your `lab6` directory and click on the button "New" to create a new Python 3 notebook. You should see a page
similar to the following:
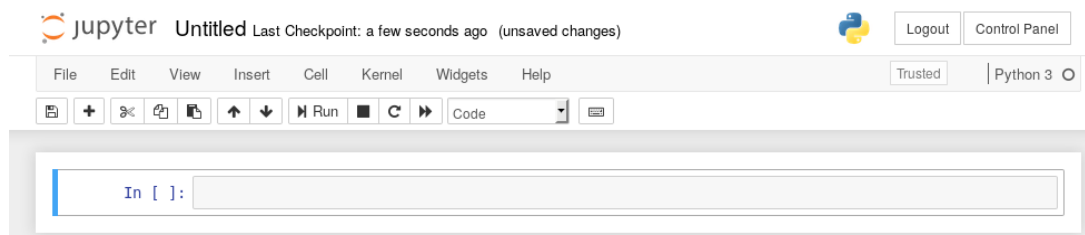
We will first rename the notebook by going to the menu option "File" and choosing "Rename…" and renaming the
notebook to `first_notebook`.

Let us add a title to our notebook. The text field that starts with "In [ ]:" is called a *cell* and we first need to change
this cell to a Markdown cell, where Markdown is a markup language that we already saw in GitLab. Go to the
menu, click on "Cell', then "Cell Type" and choose "Markdown". Now, in the Cell type in

```
# My First Jupyter Notebook

In this example, we will show how to demostrate results of a POS
tagger.
```

If you click on "Run" in the menu, or press Ctrl+Enter keys, you will "run" the cell and it will appear nicely formatted. You should also see the next cell below it.

The new cell should again have a label "In [ ]:" and you should see type "Code" in the menu. In this cell, enter your code that was the solution in the file ne_chunker_exercise.py. Remember that it started with the following template:

```
from nltk import FreqDist
# import treebank
# import ne chunker

data = # load treebank data
chunkd_data = # chunk the data
chunkd_trees = # select subtrees which are NE

word_fd = FreqDist([' '.join(word for word, pos in tree.leaves()) for
                                                    tree in chunkd_trees])

print("Three most common named entities are: ")
for token, freq in word_fd.most_common(3):
    print("%s : %d"%(token, freq))
```

Once you enter the code (with appropriate snipped filled in), you should run it. If there are any errors, they will be shown. Otherwise, if you do not see any output, click on the menu option "Cell", then "Current Outputs", and "Toggle", and then you can click on the dots after the cell and see the output. The page should look something like this (where parts that you should fill in are shaded):

Save your notebook by clicking at the first icon in the menu in the second row (if you go over it with the mouse it will show the tooltip "Save and Checkpoint"). If you go to you Bluenose terminal, you should be able to see the file `first_notebook.ipynb` in your `lab6` directory. You can check its contents using the `more` command. The contents are in the JSON format which has a lot of metainformation, but you should still be able to inspect that your code and output are there. You can now logout from your JupyterHub account in the browser.

Submit: Save the previous code as the script `first_notebook.ipynb` and submit it using the `submit-nlp` command.

This is the end of Lab 6.