Natural Language Parsing

In this lab, we'll explore acquisition of semi-structured data from a web service, browsing the data and then parsing free-text content.

The data come from wikipedia which has a very simple REST API. While the focus is on natural language parsing, we'll also explore JSON and Wikipedia's own markup format.

We'll be using network access for this assignment, so MAKE SURE YOUR LAPTOP NETWORK IS UP before starting the VM.

Once your VM is up and running, you can download this notebook file by clicking on icon at the top right of this page. Create a directory ~/labs/lab4 to hold it.

Stanford Parser Setup

First we need to install some parsing tools. Download the Stanford parser from **here (https://ufl.instructure.com/courses/320501/files/25762281/download)** . If your network is not working, download from a browser on your host machine and then use drag-and-drop.

Either way, you can put the parser in your "Downloads" directory. Unpack it with

```
tar xvzf stanfordparser.tar.gz
```

and then move it to the /opt directory with

```
sudo mv StanfordParser /opt
```

It will be helpful to have links to the parser scripts from your bin directory. If you havent already, create a directory ~/bin. Then

```
cd ~/bin

ln -s /opt/StanfordParser/lexparser.sh lexparser.sh

ln -s /opt/StanfordParser/lexparser-gui.sh lexparser-gui.sh

ln -s /opt/StanfordParser/dependencyviewer/dependencyviewer.sh
dependencyviewer.sh
```

These files will be in your path the next time you login. You can logout from the start button at the top right of the VM window. Then log back in again.

Mediawiki Parser Setup

The mediawiki parser is memorably named "mwparserfromhell". To install it with a working network, all you need to do is

```
sudo pip install mwparserfromhell
```

Accessing the Wikipedia Web API

Start by looking over the **Mediawiki API documentation** **(http://www.mediawiki.org/wiki/API:Main_page)** which describes Wikipedia's RESTful API.

The code below implements an API call with options:

- format=json to receive JSON data
- action=query to query Wikipedia content
- titles=string to specify a list of page titles to search for
- prop=revision to return the revisions of the page
- rvprop=content to return the full page content

We'll start with the title search string 'parsing' to retrieve the page about parsing.

```
import requests

title='parsing'

response = requests.get("http://en.wikipedia.org/w/api.php?
format=json&action=query&titles="+str(title)+"&prop=revisions&rvprop=content")

response
```

The response object is an HTTP GET response. It turns out the requests package contains a json interpreter, which we can invoke as:

```
jsondata = response.json()
```

If you dont have a working network, copy the file **parsing.json** **(https://ufl.instructure.com/courses/320501/files/25782592/download)** into your ~/labs/lab4 directory. Then you can load it with:

```
import json

fp=open('/home/datascience/labs/lab4/parsing.json','r')

jsondata=json.load(fp);

fp.close()
```

The JSON object is hierarchically structured. To view it, its helpful to define a couple of helper routines:

```
import json

def pretty(jdata):

    str = json.dumps(jdata, sort_keys=True,
indent=4).decode('string_escape')

    return str

def saveas(sdata, fname):

    f = open(fname,'w')

    f.write(sdata)
    f.write(sdata)

    f.close()
```

The first routine converts the JSON to a carefully-formatted string. The second writes a string to a file. We can use them together to save the JSON data to a better format for viewing.

```
saveas(pretty(jsondata), '/home/datascience/labs/lab4/'+title+'.json')
```

Now open the file '/home/datascience/labs/lab4/parsing.json' by right-clicking on it and using "open-with" with emacs or gvim. Note the structure.

The JSON parser converts JSON data nodes and lists of nodes. The nodes are represented as Python "Dict" objects, and the lists are Python lists. Each Dict maps the names of the nodes children to their values. We can query the type of each node using the "type" function. For each Dict, we can enumerate the keys using the keys() method. In this way we can explore the JSON tree (although its much quicker to eyeball it from the JSON file we just saved). But anyway we can browse with:

```
type(jsondata)

jsondata.keys()
```

which is a list of just one string (a unicode string, hence the "u" prefix). We can then extract that node with

```
jsondata['query']
```

and continue exploring:

```
type(jsondata['query'])

jsondata['query'].keys()
```

From the pretty-printed file, we know we are looking for the 'pages' child, which has a page id number. We dont know what this number is, so we cant use it as a key. But instead we can use the 'values()' method on the dictionary to get a list of all the nodes below it. We only need one page, so we take the first of those.

```
jsondata['query']['pages'].values()[0]
```

Continue down the tree, next to the "revisions" node. This time, take the *last* revision in the list.

```
# content =  ???

content
```

The content is now a text string in Mediawiki's own format. To make sense of it we can use the mwparserfromhell (MWPH for short).

```
import mwparserfromhell as mwph wikicode = mwph.parse(content)
```

MWPH supports a variety of methods to explore mediawiki content. The main class is the Wikicode class, which is the type returned by mwph.parse(). e.g. try

```
wikicode.filter_comments()
```

```
wikicode.filter_headings()
```

```
wikicode.filter_wikilinks()
```

But since we want to parse the english text from the article, we want to ignore all these metadata. MWPH has a method to do this:

```
text = wikicode.strip_code()
```

```
text
```

This data is clean enough now that we can save it for parsing:

```
 saveas(pretty(text), '/home/datascience/labs/lab4/'+title+'.txt')
```

Running the Stanford Parser

From a terminal window, type

```
lexparser-gui.sh
```

This brings up a GUI interface to the Stanford parser. To use it, click on "Load Parser" which brings up a file selection dialog. Navigate to

```
/opt/StanfordParser/stanford-parser-3.4.1-models.jar
```

and open it.

Then you will see a list of parsers to use. Select

```
englishPCFG.ser.gz
```

You're now ready to parse some text!

Click on the "Load File" button, and browse to the lab4 directory and load the parsing.txt file. Click on "Parse" to parse the current sentence (highlighted in yellow).

Content Analysis

Now lets try to analyze some content from Wikipedia. To make our lives simpler, we'll use a simplified english version of wikipedia. Change the URL in the first code box in this file to:

```
simple.wikipedia.org
```

and change the query title to 'cat'. Rerun all the cells above. This should produce a file "cat.txt" in the lab4 directory. Load that file into the parser, and parse some of the sentences.

If you cant access the network, download the file **cat.json (https://ufl.instructure.com/courses/320501/files/25782822/download)** into your lab4 directory, and repeat the commands used earlier to load the parsing.json file.

[illegible]

We'll now convert the parser output to XML, so we can process it further. Find the script

```
  /opt/StanfordParser/lexparser.sh
```

and edit it so that its outputFormat is:

```
-outputFormat "xmlTree"
```

and add a new option:

```
-outputFormatOptions "xml"
```

save the new script as

```
parsetoxml.sh
```

and create an alias to it in your ~/bin directory. Now run from your lab4 directory

```
parsetoxml.sh cat.txt > cat.xml
```

you're ready now to analyze the cat data. We'll use Python's builtin ElementTree parser.

```
from lxml import etree parser = etree.XMLParser(recover=True) tree =
etree.parse('/home/datascience/labs/lab4/cat.xml',parser)
```

We can examine the root of this tree:

```
root=tree.getroot() root.tag
```

```
len(root)
```

```
root[0].tag
```

i.e. we have found the first sentence. The xmlTree representation is a little tricky however, as POS tags are stored as attributes of nodes rather than node tags. To get to the actual root node, we need to dig a little deeper (and we'll use the second sentence which is a bit more conventional):

```
root[1][0][0].attrib['value']
```

going down one level gets us to the actual sentence node:

```
s=root[6][0][0][0] s.attrib['value']
```

and to get its children we can do:

```
s[:]
```

This is not too helpful, because the node types are hidden in the value attribs of these nodes. To see them, we can use a python anonymous function and map it over the list.

```
map(lambda (x): x.attrib['value'], s[:])
```

Now lets see if we can find sentences starting with noun phrases containing a given noun. The final function supports a flexible syntax (similar to xpath) for locating elements of given type or attributes. A slash "/" is like a directory specifier, and defines a child node. A double slash "//" specifies any descendent, child, grandchild, great-grandchild etc specifier, and defines a child node. A double slash "//" specifies *any* descendent, child, grandchild, great-grandchild etc. The "node[@value='NP']" specifies a node with the given attribute value.

```
agent = s.findall("./node[@value='NP']//node[@value='NN']//leaf[@value='cat']")
agent
```

finds all the nodes starting with an 'NP' child of s, and having a 'NN' node above a leaf with 'cat' value.

We can similarly look for a verb in a verb phrase under the root node:

```
verb = s.findall("./node[@value='VP']//node[@value='VBZ']//leaf[@value='is']")
verb
```

Putting these together, we can discover sentences containing a given pair of (agent,action) pairs:

```
def printnode(node):

    for i in node.findall(".//leaf"):

        print(" " + i.attrib['value']),

    print('')

def testnode(node, agent, action):

    aa =
node.findall("./node[@value='NP']//node[@value='NN']//leaf[@value='"+agent+"']")

    bb = node.findall("./node[@value='VP']//leaf[@value='"+action+"']")

    if (len(aa) > 0 and len(bb) > 0):

        printnode(node)

def agentact(node, agent, action):

    testnode(node, agent, action)

    snodes = node.findall(".//node[@value='S']")

    for snode in snodes:

        testnode(snode, agent, action)


agentact(s, title, 'is')
```

Next we can map the agentact function across all the sentences in the Wikipedia entry:

```
map(lambda (nn): agentact(nn[0][0][0], title, 'is'), root) []
```