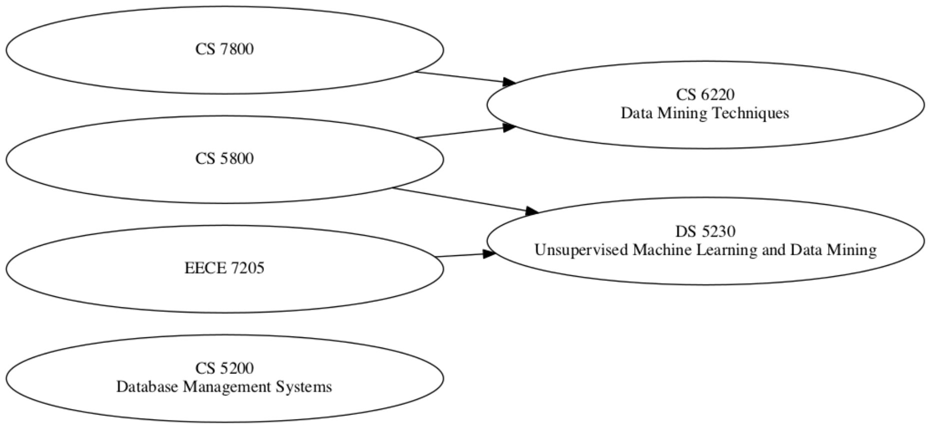
Northeastern University
CS6220 – Data Mining Techniques
Fall 2017, Derbinsky

Homework 2, Part 1

This is the first part of Homework 2, which focuses on *web scraping*, or the automated extraction of data from websites. The final task will be to write a program to extract and visualize course relations in the Northeastern dynamic search site. The following is an example output, based on the search of all courses taught this semester by "Derbinsky, Nathaniel L."...


CS 7800
CS 5800
EECE 7205
CS 5200
Database Management Systems
CS 6220
Data Mining Techniques
DS 5230
Unsupervised Machine Learning and Data Mining

1 Web Scraping in Python

Automated web scraping requires two main components: (1) issuing HTTP requests and (2) parsing HTML.

1.1 HTTP Requests

Your Python distribution likely comes with the `requests` module. This module[1] makes it easy to issue requests using various methods, get/set cookies/headers, authenticate, etc. The starter code for this assignment includes pre-written functions for GET/POST, but you may want to look at the Quickstart guide (`http://docs.python-requests.org/en/master/user/quickstart/`) for a quick idea. Here's a small example of getting the HTML of the Google homepage:

```
>>> import requests
>>> r = requests.get('http://www.google.com')
>>> r.text
```

It is similarly simple to get data from web APIs:

[1]`http://docs.python-requests.org`

```
>>> import requests
>>> r = requests.get('http://api.population.io:80/1.0/population/Brazil/today-and-tomorrow/')
>>> r.status_code
200
>>> r.json()
{'total_population': [{'population': 211659618, 'date': '2017-10-02'},
{'population': 211664119, 'date': '2017-10-03'}]}
```

It may be useful to see the results of your query outside Python. For *nix environments, `curl` is a commonly used command-line tool. Postman[2] is a free, cross-platform application that has a GUI.

1.2 Parsing HTML

The `BeautifulSoup` module[3] makes it very easy to operate on HTML documents (e.g. those retrieved via `requests.text`). The first step is to parse the HTML:

```
>>> import requests
>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(requests.get('https://www.google.com').text, 'html.parser')
```

Then very easy attributes and functions to search/access HTML elements:

```
>>> soup.title
<title>Google</title>
>>> soup.title.string
'Google'
>>> for link in soup.find_all('a'): print(link['href'])
https://www.google.com/imghp?hl=en&tab=wi
https://maps.google.com/maps?hl=en&tab=wl
https://play.google.com/?hl=en&tab=w8
https://www.youtube.com/?gl=US&tab=w1
https://news.google.com/nwshp?hl=en&tab=wn
https://mail.google.com/mail/?tab=wm
https://drive.google.com/?tab=wo
https://www.google.com/intl/en/options/
http://www.google.com/history/optout?hl=en
/preferences?hl=en
https://accounts.google.com/ServiceLogin?hl=en&passive=true&continue=https://www.google.com/
/advanced_search?hl=en&authuser=0
/language_tools?hl=en&authuser=0
/intl/en/ads/
/services/
https://plus.google.com/116899029375914044550
/intl/en/about.html
/intl/en/policies/privacy/
/intl/en/policies/terms/
```

To understand the structure of a document visually, it is very useful to "Inspect" from within common browsers – here you'll be able to visually click hierarchically within the DOM.

[2]https://www.getpostman.com
[3]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

2 Graph Visualization with Graphviz

There are many tools available for visualizing graphs, including D3[4] and Gephi[5]. For this assignment you will learn a bit of the DOT language[6], which can be visualized using Graphviz[7] and other software.

First, download and install Graphviz for your platform.

2.1 DOT for Directed Graphs

The DOT language can be used to describe a wide variety of graphs[8]. For basic di-graphs, the syntax is relatively straightforward:

```
digraph G {
    rankdir="LR";
    node [width=5, height=1];
    CS_5200 [ label="CS 5200\nDatabase Management Systems" ];
    CS_5800 [ label="CS 5800\n" ];
    CS_6220 [ label="CS 6220\nData Mining Techniques" ];
    CS_7800 [ label="CS 7800\n" ];
    DS_5230 [ label="DS 5230\nUnsupervised Machine Learning and Data Mining" ];
    EECE_7205 [ label="EECE 7205\n" ];
    CS_5800 -> CS_6220;
    CS_5800 -> DS_5230;
    CS_7800 -> CS_6220;
    EECE_7205 -> DS_5230;
}
```

The graph is first declared to be directed (digraph), with a name (G in this case). The next line indicates to orient the graph as left-to-right. Next, default nodes are given base size information, and each distinct node is listed with a label (many other modifiers, such as color, exist as well). Finally, edges are described as directed arrows between nodes.

To render this DOT as an image, simply open it in a DOT-compatible application, or execute the following from a command line: `dot -Tpng -ooutput.png input.gv`

[4] https://d3js.org
[5] https://gephi.org
[6] http://graphviz.org/content/dot-language
[7] http://graphviz.org/
[8] http://graphviz.org/Gallery.php

3 Your Task

You are to complete a program that queries the Northeastern "Dynamic Schedule"[9] for course data, scrapes the page for courses and their associated pre/co-requisites, and outputs the result in DOT:

1. Play with the existing web form – understand what inputs/outputs look like. Examine the HTML in your browser to understand parameters for the search, as well as format of the results. You have been provided additional output examples for CS undergraduate and CS graduate courses. Be sure you understand what data is being presented.

2. Complete the `coursesearch` function in the supplied starter code. This function should make an HTTP request according to the function parameters, pass the resulting HTML to the `_parse_course_listing` function, and return the result. Note that insufficiently specific queries may take several seconds to respond.

3. Now implement the `_parse_course_listing` function, which takes as input HTML and outputs a tuple of course data (format is your choice, but relates to the next step). For simplicity, ignore all but each course's subject/number (e.g. CS6220), title (e.g. "Data Mining Techniques"), and associated pre/co-requisites (hint: these appear in the course fields with a link!)

4. The result of `coursesearch` will be passed to the `print_course_dot` function – now write it! Given the course information, produce DOT output that visualizes pre/co-requisites as nodes that have outgoing edges to their associated course. **To facilitate grading, the order of nodes and edges must be sorted.**

5. Once complete, test your code with several queries. Then clean up, document, and submit your code. For reference, you have been supplied example outputs produced via the following commands:

```
./courses.py --instructor "Derbinsky, Nathaniel L." "Fall 2017 Semester" > derbinsky.gv
./courses.py --subject CS --level UG "Fall 2017 Semester" > cs_undergrad.gv
./courses.py --subject CS --level GR "Fall 2017 Semester" > cs_grad.gv

dot -Tpng -oderbinsky.png derbinsky.gv
dot -Tpng -ocs_undergrad.png cs_undergrad.gv
dot -Tpng -ocs_grad.png cs_grad.gv
```

For this part of the homework you will only submit the `courses.py` file. The grading process will include performing a `diff` between the output of your program with the example outputs – differences will result in a loss of credit.

[9]`https://wl11gp.neu.edu/udcprod8/NEUCLSS.p_disp_dyn_sched`