CAB431 Week 4 Workshop
Pre-Processing: Stemming

****

Stemming refers to a crude heuristic process that removes the ends of words in the hope of finding the stemmed common base form correctly most of the time. This process often includes the removal of derivational affixes. Lovins (1968) defines a stemming algorithm as "a procedure to reduce all words with the same stem to a common form, usually by stripping each word of its derivational and inflectional suffixes (and sometimes prefixes)". A common stemming algorithm is the Porter2 (Snowball) algorithm. You can read the details of the Porter2 algorithms form here:
http://snowball.tartarus.org/algorithms/english/stemmer.html

For Python, please go to https://pypi.python.org/pypi/stemming/1.0 to download Python implementations of porter2 stemming algorithms, follow the instruction to import and use stemmer in your Python code (or see the Blackboard).

Task 1: Stemming – using porter2 stemming algorithm to update your last week function parse_doc(input, stops) to ensure all terms are in stemmed forms. The following is an example of using the return value of parse_doc() for file "6146.xml".

Document itemid: 6146 contains: 133 words and 72 terms
------Terms and Their Frequencies-----
addit : 1
air : 1
amid : 1
argentin : 1
argentina : 2
await : 1
axel : 1
bank : 1
between : 1

bocon : 1
bond : 1
bounc : 2
bueno : 1
bugg : 1
chang : 1
congress : 1
deficit : 1
deleg : 1
denomin : 1
dollar : 1
due : 2
dure : 1
earli : 1
econom : 1
economi : 1
event : 1
expect : 3
fernandez : 1
fiscal : 1
foreign : 1
frb : 1
friday : 1
fund : 1
general : 1
govern : 1
higher : 1
includ : 1
intern : 1
larg : 1
low : 1
…

Task 2: Define a **Doc_Node** Class:

```
class Doc_Node:
  def __init__(self, data, next=None):
    self.data=data
    self.next=next
```

where "data" attribute stores the document's information, i.e., tuple (docid, curr_doc), and

- *docid* is the 'itemid' in <newsitem>
- *curr_doc* is a dictionary of term_frequency pairs (review last week workshop if you do not understand the tuple).

You are also required to define a **linked list** class:

```
class List_Docs:
  def __init__(self, hnode):
    self.head=hnode
```

And its methods:

```
insert(self, nnode) # append nnode to the end of the linked list
lprint(self) # print out the linked list (see the outcomes of Task3)
```

Task 3: Design the main function to read several xml files and represent each file as a node, then create a linked list to link all nodes together. You need to update function **parse_doc()**, e.g., arguments or return value, and then use **Doc_Node** and **List_Docs** classes.

Examples of output

```
(ID-6146: 72 terms) --> (ID-741299: 96 terms)
```