Lab 2

Goal: To put python to real world use. A good use of a scripting language like python is to scrape web for data collection. So we want you to learn write parsing scripts and put the data to use.

Problem: Govind is a CS PhD student at the U of M. For his dissertation work he needs help in data collection for one of his project. The aim of the project is to find related words for a given set of keywords. For example: related words for 'tree' are 'root', 'wood', 'leaves'. Related words may not necessarily means synonyms. For a human this task is easy to do but if we want to automate this task then we require a lot of information. And what's the biggest source of information today- the web. So Govind has an idea to use information from the web to know the strength of relatedness of words. Govind has written down the following steps for your guidance, follow them and let's see if we can make it.

Expectations: This lab is intended to give you experience of an interesting real world use case of python as scripting language. As you do this lab, you will surely learn many new and interesting things.

Note: For this lab, we are going to use only python 2.x version since several of the imports are compatible with this version rather than the higher version.

Step 1: Reading from a file (read_file())

Write a function to read strings from a text file. The text file is given(test_file.txt). It basically contains two columns. The first column is the keywords and the second column is the candidate set of words from which we have to find related words. The two columns are separated by '***' and the words in the second column are separated by comma (','). Read from the file one line at a time i.e iterate over each line.

Forest***fire, cloud, rain, ocean, burn, relief, enjoy, party

Keyword Candidate words

You have to write a file read command which is basically open(file-name,mode). To read all the lines in the file use the readlines() function. Now write an iterator to go to each line and pass the line to the term extraction function in Step 2.

Step 2: Term extraction (extractor())

Read the keyword in a string and read the candidate words in an array of strings (list of strings). This functions gets a line (in string format) and the purpose of this function is to extract the keyword in a string variable and the candidate words in an array of strings. You may use string functions such as strip(), split(), remove() etc. to extract the word from the line you have read from the file in the previous step. The way to think about this is that there are two part in the line which have to be separately extracted (keywords and candidate words). The above mentioned function will help to set the logic. You will have to think how to extract the words from candidate words in a string of array. (Hint: the functions mentioned above are sufficient for this). This function calls the web_data() function in step 3 with the keywords and array of strings.

Step 3: Web calling module (web_data())

This function is basically the core of the web calls in this project. The parameters to this function are keywords and the list of candidate words. The purpose of the function is to compute the relatedness between the keyword and the words in the candidate list. For the keyword and the candidate words passed to this code, the next step is to find the relatedness of the keyword with each of the word in the candidate words. Following sub-steps have to be followed to accomplish it:

Step 3.a: Get web count for terms (web_count(word))

Write a function **web_count(word)** which count the number of web hits for the input word. Within this function, you will require a temporary file to store the web content. We are now writing a python call to a search engine (preferably Bing since it does not block very soon, Google does). Here you have to use the urllib library, so import that first. Next formulate your query. The query consists of the URL of the search engine (copy and paste it from a sample search on a search engine) and replace the test word with the input word.

As you know that any search engine return the number of hits for the query you made. So we want to extract the number of hits from the web page. In order to get the whole web page you can use the urllib.urlretrieve() function. This function will store the web page in the temporary file you made. There are other ways to make a URL call as well (http://effbot.org/librarybook/urllib.htm).

Once the web page is in the file, your task is to read the file and parse it to find where the hits counts are returned. You may have to read the file and then store it in a single string and search for the Hits results.

This should be very simple parsing, you just have to identify which strings are near the hit counts and see how you can get the hit results using some strong operations. For this find(), replace() functions will be helpful.

The function finally returns the count (float type). In case the word does not exists in the web(!), return 0.0.

Step 3.b: Compute Distance

After you have written the web_count(word) function, you are supposed to use it to get web hits for the keyword and the candidate words which were input to the web_data() function. First store the hits for the keyword (say in variable x) and the iterate over each word in the list of candidate words and for each one compute their hits(call it y) and also compute the hits for keyword+` `+candidate word (call it xy). Basically, we are checking how many hits does the web has for the keyword, the number of hits for the candidate word and also the number of hits for the combination of both.

Based on scientific research(the Normalized Google distance), it is found that we can compute relatedness between any two words if we know their individual web hits and hits when they are in combination. The code for this function (Distance()) is already given to you. This function takes three parameters:

x→ number of hits for word 1

y→ number of hits for word 2

xy→ number of hits for combination of word1 and word2

Since, we have already computed these we can pass them to find the relatedness score. Note that relatedness score is a float value. Just a reminder: we are finding the two words in the candidate words that are most related to the given keyword. So you will require to compare the relatedness score of each word. An easy way to do it is to use a dictionary. Store the relatedness score of each word in the list of candidate words in the dictionary. Think- what should be the key and what should be the value!

Finally, you need to use the dictionary to find the two words which are most related. A lower relatedness score in the distance function means more relatedness. You may want to sort the dictionary for this. The dictionary can be sorted using:

sorted(dict_c.iteritems(), key=operator.itemgetter(1))

dict_c is the dictionary

Finally print the two words which are most related.

You are done.

Note: For different search engine the parsing script will be different.

Useful Links

Dictionary: http://www.tutorialspoint.com/python/python_dictionary.htm

File Input output: http://www.tutorialspoint.com/python/python_files_io.htm

url call: http://effbot.org/librarybook/urllib.htm