Notes on Homework 2

Homework 2: Development of Regular Expressions in the SpamLord program

In Blackboard, under Assignments, **download the SpamLord.zip** file and put it in your NLP class folder. To unzip this file:

Mac: double click on the .zip file and it should put the unzipped folder in the same directory.
Some versions of Windows: Right click on the .zip file and choose Extract and it should put the folder there in the directory. (On some versions, selecting the 7zip extract can be faster.) Since this was zipped on a Mac, you may see extra files like .DSstore, which you can safely delete or ignore.

On some systems, the SpamLord folder will have an extra layer: inside the SpamLord folder is another folder named SpamLord, and this is the one with the SpamLord.py program and the data folder.

We are going to look at the program SpamLord.py and work on the program in this help session. You can edit the latter program either by opening it in a program like NotePad++ or Sublime, or by editing it in TextEdit on a Mac.

We are going to run this program as a stand-alone Python program from the command line. If you wish to develop in Jupyter, note that you will need modify the main program and to introduce variables for the command line arguments, data_path, gold_path.

[Windows: open a Command Prompt window by going to menu and typing in “cmd”, which should give a menu with “Command Prompt” and then should open a window for you to type commands. Use the “change directory” or cd command to go to the SpamLord folder in your NLP class folder.

```
%  cd NLPclass\SpamLord
```

or whatever path you need to type to get to the SpamLord directory. If you want to see the contents of whatever directory you are in, type the command ‘dir’

```
% dir
```

and you should see the two SpamLord programs and the data directory. But if you see another directory called SpamLord, just use cd SpamLord to get down one more level until you are in the same directory as the programs.
End Windows]

[Mac:
open a Terminal window by going to your applications folder, under Utilities, and clicking on Terminal.

First, we want to go to the SpamLord directory. Use the “change directory” or cd command to go to the SpamLord folder in your NLP class folder.

```
%  cd NLPclass/SpamLord
```

or whatever path you need to type to get to the SpamLord directory. If you want to see the contents of whatever directory you are in, type the command ‘ls’

```
% ls
```

and you should see the two SpamLord programs and the data directory.
End Mac]

If we look at the data directory, we see the files for each of the faculty and staff names in the dev directory along with the devGOLD file.

We can run the program by just issuing the python program as a command, giving the SpamLord base program and two arguments, which are the dev folder and the devGOLD file. For purposes of the demo, I am actually going to use the file SpamLord.base.py, which starts from the very beginning, and the file SpamLord.py has the results of these demo notes.

```
% python SpamLord.py data/dev data/devGOLD
```

The output of the program lists the number of correct matches as True Positives (0), the number of incorrent matches as False Positives (0) and the number of unmatched gold answers as False Negatives (117).   Note that each gold/correct answer has the form (filename, ‘e’ or ‘p’, standard format email or phone).

To start making patterns, we look at some of the unmatched examples. Let’s pick the email address for ‘balaji’.

Now in Windows, with your Explorer computer window (NOT Internet Explorer), go to the SpamLord directory.
On the Mac, open a Finder window and go to the SpamLord directory.

First, make a copy of the SpamLord.py program so that you can always go back to the original.

Now go to the data/dev directory and open the file called ‘balaji’ in a text editor like NotePad++ or TextEdit. But note that we are only going to read this file and not edit it.

Find an occurrence of an email address for balaji; it’s about 2/3 of the way down the file after a mailto: tag. This email address is not obscured, so we can write a regular expression that matches it directly. Let’s allow any alphabetic character before the @

sign and any alphabetic character after the @ sign and before .edu. So we will add the following line to SpamLord.base.py after the line
epatterns = [ ].
And remember that we are supposed to have one set of parentheses around the "someone" part and another set around the "somewhere" part but not including the .edu.

```
epatterns.append('([A-Za-z]+)@([A-Za-z]+)\.edu')
```

This line should already be in your program and you just need to remove the # comment sign at the beginning of the line. We save our program and run it again. This pattern not only matched the balaji email address but correctly matched three others.

But we note that we incorrectly matched young. We'll go to the data/dev folder again and this time we'll open the file psyoung in a text editor. When we find the email address, we see that the problem is that we didn't match ALL the email address. We also need to be able to match a '.' before the @, and we might as well allow a '.' after the @ as well. We change our pattern to:

```
epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.edu')
```

That change not only corrected psyoung, but it added a bunch of additional correct email addresses that had a '.' after the @.

Now let's work on one of the unmatched examples. Open the file for ashishg and find his email address after Email: in the file. We can see that he has put a space before and after the @ sign. Now we can add a second pattern or we can change our first pattern to have an optional space. Here is a second pattern:

```
epatterns.append('([A-Za-z.]+)\s@\s([A-Za-z.]+)\.edu')
```

Save the program and run it again. That captured both of ashishg's email addresses.

Important! Remember that there are two parts to capturing the email addresses:
1. Write a regular expression pattern to match the obfuscated email.
2. Make sure that each regular expression has exactly 2 parentheses to capture the userid and the domain and that the result will be followed by .edu.

If the text can't be matched in this format, then the epattern list isn't sufficient to do it.

To start you on your homework assignment, I have put this epattern and a phone pattern

```
ppatterns.append('(\d{3})-(\d{3})-(\d{4})')
```

into the file SpamLord.py. You can start with this program and keep developing patterns to increase the number of correct matches.

1. Select an unmatched gold example.
2. Find the data/dev file that the example should have come from and look at the obfuscated example.
3. Decide if you can add to an existing pattern, write a new pattern, or if the example cannot be matched with the format of the epatterns and ppatterns.
4. If you can add to or write a pattern, edit the program and run it again.
5. Observe the results!
6. If you can't write a pattern to match it with only 2 parentheses, save that obfuscated example for optional parts 2 and 3. If you did match an example, note the obfuscated example for your report.

If you continue this process with only the 2 parentheses for the .edu patterns, note that you should be able to get all but about 12 False Positives and False Negatives.

As you write patterns, note which examples of email do correctly match and also save those for Part 1 of your report.

Optional Part 2:

If you choose to explain the non-matches, for each of your False Positives and False Negatives, look at the obfuscated email of phone in the file and explain why you can't match it. Examples of reasons are that there are more than two parts of the email to be captured, or it is not a .edu email address. If possible, show a regular expression that would match, but doesn't capture all of the parts of the result. (And also do part b about how to better obfuscate email addresses.)

Optional Part 3:

If you choose to do further python programming to match more patterns, you can have additional lists, such as a list that captures 3 parts of email addresses or one that matches .com patterns. For each list, add a part to the process_file function to find the results of the matches and put the correct email addresses or phone numbers in the res (result) list. You may also use things like the string replace function to make the strings easier to match.