

## NLP Homework 2

Due Thursday, October 12, 2017 at midnight

### SpamLord! Using Regular Expressions to find email addresses and phone numbers

This homework is based on a similar homework called SpamLord from the Stanford NLP course, in particular, I thank them for the starter programs and the data to carry out this task. The data is modified from the original to change the task each semester, but please don't make any of the programs or data public.

On the web, many people who want to give their email addresses or phone numbers for people to contact them will try to make it hard for spammers to automatically search text for email addresses and phone numbers by giving non-standard forms of them, sometimes called obfuscation. You may have seen examples like:

jurafsky(at)cs.stanford.edu  
jurafsky at csli dot stanford dot edu

For this homework, we are going use regular expressions to try to find these email addresses anyway, or as the Stanford assignment puts in "Here's your chance to be a SpamLord! Yes, you too can build regexes to help spread evil throughout the galaxy!"

For the obscured email addresses like those above, your job is to return an email address in the standard form:

[jurafsky@cs.stanford.edu](mailto:jurafsky@cs.stanford.edu) and [jurafsky@csli.stanford.edu](mailto:jurafsky@csli.stanford.edu)

Similarly, for phone numbers, given examples like:

TEL +1-650-723-0293  
Phone: (650) 723-0293

You should return the standard form of a phone number 650-723-0293. (We'll assume that all of the phone numbers are in North America.)

For the homework, there is a directory called SpamLord that has the following structure:

```
SpamLord/  
  SpamLord.py  
  data/  
    dev/  
      aiken  
      ashishg  
      ...  
    devGold
```

In the data/dev directory are text files that have html text with obscured emails and phone numbers from faculty and staff at Stanford. In the data/devGold file are the corresponding standard form emails and phone numbers; these are the correct answers.

The program SpamLord.py is set up so that you can write a set of regular expressions with parentheses to pick out the parts of the email address or phone numbers that match it in their obscured form. Then there is a function “process\_filename” that goes through lines of the file, matches all the regular expressions and builds up a result list called “res” that has the name of the file, an ‘e’ or ‘p’ for email or phone, and the standard format email address or phone number.

The rest of the program processes all the files in the dev directory and combines the standard form results, it compares them with the devGold file and reports how many your program got right.

In the evaluation part,

- the true positive section displays e-mails and phone numbers which the code correctly matches,
- the false positive section displays e-mails which the code regular expressions match but which are not correct, and
- the false negative section displays e-mails and phone numbers which the starter code did not match, but which do exist in the html files.

Your goal, then, is to reduce the number of false positives and negatives

The code that I am giving you to start is set up to make a list of regular expression patterns that find email addresses that have two matched parts: each re matches a userid and a domain, which it assumes are followed by .edu. Then in the process\_filename function, these patterns are matched against each line of the file. When a pattern matches, the match will have two parts from the parentheses representing the userid and the domain, and these are filled into the result.

In the Homework 2 notes, I will start with one simple pattern and show the process to add patterns to make correct matches by either correcting a pattern that incorrectly made a false positive or by adding to a pattern or adding a new pattern to match an example in the false negatives.

As you work, I recommend that for each example that you try, make notes. This can either be a pattern that works and save an example of an obfuscation that it matches, or an example of an obfuscation for which you did not successfully write a pattern and why you couldn’t do that.

You may wish to use regexpal to try out examples, but note that only examples with two sets of parentheses work in the current python code.

1. (Required) Write more regular expressions that correct false positives or fit false negatives. Do as best as you can using the epatterns and ppatterns lists in the program. List the regular expressions that you write, examples of email or phone numbers that match each pattern, and a short English explanation of what expressions the pattern matches. For each expression, give the results (TP, FP and FN) before and after you added the expression. And give the text that

matches, not only the results! When you are done with as many as you can do, give the output of the program.

Then choose either option 2 or option 3:

2. (Option) Do both parts a and b in this option.

a. List the examples that you found you could not match with the current regular expressions with two extracted parts, ending in .edu. For each example, or set of examples that fit the same pattern, explain briefly why it won't work. If you can make expressions in regexpal that match, but don't work in the program to extract the email or phone numbers, list those here. If you had any false positives in Part 1, include a discussion here of why that rule generated them.

b. Then search the web and find a couple of additional examples of obscured email addresses or phone numbers and report on them, or try to design a way to obscure an email address that would be extremely difficult for spamlord to match with a regular expression. For the latter, try to have something more specific than things like "To send me email, try the simplest address that makes sense."

3. (Option) Python programming: Continue working on the regular expressions to match more examples by having other lists of patterns. For example, you may want to have patterns that will match three parts of the email addresses, or you may want to make a list of patterns for email addresses that end in .com. For each list, you will need to add a part to process\_filename that matches that list and puts its parts into a standard format answer.

Write a section in the report that describes your approach and gives examples of the extended regular expressions or processing that match some of the emails. If you add code to the program, add comments to the program that you submit.

### **What to submit for Homework:**

If you do Part 1 and choose to do Option 2, write a short report that shows the epatterns that you developed for part 1 and discusses Part 2, both a and b.

If you do Part 1 and choose to do Option 3, write a short report that shows all the patterns that you developed, with examples, and the code for process\_filename.

For all sections, your report should demonstrate your understanding of regular expressions.

In both cases, please also submit your version of SpamLord.py. This program MUST be a .py program, please do not submit .ipynb programs.

### **How to Submit Homework:**

Go to the Blackboard system and the Assignment for Homework 2. Attach your report and your version of the SpamLord program.