John Fields
Dr. Norma Palomino
IST664 - Homework #2
ContactFinder
11/06/19

Introduction

An essential skill for Natural Language Processing (NLP) is the ability to extract text from a variety of sources including internet web sites. Contact information like e-mail addresses and phone numbers are two examples of information that is of interest to those working in NLP. This information is typically included on web sites which allow people to share their contact information and communicate with other users.

This paper utilizes a test data set where the Regular Expressions (regex) capability in Python can be used to find e-mail addresses and phone numbers in a dataset created from internet data. Some information is in the standard phone format of XXX-XXX-XXXX and e-mail format *name*@stanford.edu. However, many of the e-mails and addresses are obfuscated to hide this information from "bots" and other programs used to scrape this information for "spam" e-mails and automated "robo-calls".

The ability to "scrape" this data is a popular way for marketers and companies to gather information on current and potential customers. However, there are many ethical issues related to this type of activity and it is important to reference helpful guidelines provided by the Direct Marketing Association and other organizations prior to using these techniques for commercial purposes.[i] This paper is intended to discuss the technical techniques related to this topic, but it is important to highlight the ethical considerations that should be evaluated prior to beginning this type of activity in a business context. This paper is also a useful resource for those who don't want their contact information to be used by marketers and provides insights on steps that can be taken to obfuscate data.

Analysis

About the Data

- For this assignment, the following files were provided:
    - ContactFinder.base.py Python Program
    - Folder with 46 files that contain the "scraped" data from web pages
    - "Gold" file with 117 correct phone numbers and e-mail addresses

Methodology

- To find the correct phone numbers and e-mail addresses, the Python code was modified from the base regex patterns that were provided.
- The following format will be used throughout this paper to show each new pattern that is evaluated and the results:
    - Pattern #
    - Description
    - True Positives X
    - Before Summary: tp = X, fp = X, fn = X
    - After Summary: tp = X, fp = X, fn = X

Results

Pattern 0	epatterns = []
Description	The result of running this command is True Positives = 0, False Positives = 0, and False Negatives = 117.
True Positives	0
False Positives	0
Before Summary	tp = 0, fp = 0, fn = 117
After Summary	tp = 0, fp = 0, fn = 117

Pattern 1	epatterns.append('([A-Za-z]+)@([A-Za-z]+)\.edu')
Description	The first step to edit the file was to insert the command - epatterns.append('([A-Za-z]+)@([A-Za-z]+)\.edu') which will locate simple e-mail addresses such as balaji@stanford.edu which are not obfuscated.
True Positives	4
False Positives	1 young@stanford.edu ~ patrick.young@stanford.edu
Before Summary	tp = 0, fp = 0, fn = 117
After Summary	tp = 4, fp = 1, fn = 113

Pattern 2	epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.edu')
Description	The False Positive is Patrick Young. His e-mail was not correctly identified because it is patrick.young@stanford.edu. The current regex expressions does not recognize the period between patrick and young. The command was changed to include a period before and after the @ symbol. This pattern added 16 more e-mails.
True Positives	19
False Positives	0
Before Summary	tp = 4, fp = 1, fn = 113
After Summary	tp = 20, fp = 0, fn = 97

Pattern 3	epatterns.append('([A-Za-z.]+)\s@\s([A-Za-z.]+)\.edu')
Description	The user ashihg's e-mail was not identified correctly due to space before and after the @ sign. A second pattern was added to include these spaces.
True Positives	22
False Positives	0
Before Summary	tp = 19, fp = 0, fn = 98
After Summary	tp = 22, fp = 0, fn = 95

Now that we have the basic regex instructions for e-mails with 22 true positives, similar basic techniques will be applied to phone numbers.

Pattern 4	ppatterns.append('(\d{3})-(\d{3})-(\d{4})')
Description	This pattern looks for phone numbers in the format XXX-XXX-XXXX. 19 additional phone numbers were identified with this pattern.
True Positives	41
False Positives	0
Before Summary	tp = 22, fp = 0, fn = 95
After Summary	tp = 41, fp = 0, fn = 76

Pattern 5	ppatterns.append('\D?([\d]\d{2})\D?\D?(\d{3})\D?(\d{4})')
Description	There were several phone numbers with similar differences in format (not obfuscation) such as: • (650)723-1614 • (650) 724-6354 • +1 650 723 5666 • [650] 723-5499 • 650-724-1915 The pattern above was validated in regexpal.com with the following logic: • \D - not digit such as (+ [• ? - optional preceding token • [\d] - digit character set • \d {2} - digit match 2 preceding tokens • \D? - optional not digit such as parenthesis, or dash • \D? - optional not digit such as parenthesis, or dash • \d {3} - digit match 3 preceding tokens • \D? - optional not digit such as dash • \d {4} - digit match 4 preceding tokens The above pattern resolved these types of phone numbers to increase true positives from 41 to 94. However, Pattern 5 also generated 25 false positives as shown below.
True Positives	94
False Positives	25 ('nass', 'p', '111-618-2714') ('nass', 'p', '103-311-4910') ('rajeev', 'p', '052-147-4655') ('latombe', 'p', '110-730-4683') ('nass', 'p', '111-099-5710')

	('latombe', 'p', '552-190-2777') ('rajeev', 'p', '020-144-1241') ('pal', 'p', '122-963-8381') ('latombe', 'p', '870-145-1791') ('latombe', 'p', '689-107-3717') ('nass', 'p', '157-586-0538') ('manning', 'p', '157-586-0368') ('bgirod', 'p', '161-904-3898') ('pal', 'p', '052-189-6061') ('bgirod', 'p', '177-572-9915') ('tim', 'p', '052-187-2820') ('nass', 'p', '026-214-0926') ('rajeev', 'p', '627-020-1441') ('nass', 'p', '104-337-5742') ('bgirod', 'p', '194-257-0626') ('latombe', 'p', '698-691-6769') ('latombe', 'p', '101-812-1023') ('latombe', 'p', '161-061-1985') ('bgirod', 'p', '134-217-6593') ('latombe', 'p', '693-676-9868')
Before Summary	tp = 41, fp = 0, fn = 76
After Summary	tp = 94, fp = 25, fn = 23

Pattern 6	ppatterns.append ('\D?([2-9]\d{2})\D?\D?(\d{3})\D?(\d{4})')
Description	Since Pattern 5 included false positives that started with the numbers 0, 1 or 2, the regex pattern was updated to exclude these digits. • \d was replaced with 2-9 to eliminate matches with numbers starting with 0-2 The number of false positives was reduced from 25 to 10.
True Positives	94
False Positives	10 ('nass', 'p', '375-742-1353') ('rajeev', 'p', '627-020-1441') ('latombe', 'p', '698-691-6769') ('nass', 'p', '910-769-9828') ('latombe', 'p', '689-107-3717') ('latombe', 'p', '693-676-9868') ('latombe', 'p', '611-985-1107') ('bgirod', 'p', '593-177-5729') ('latombe', 'p', '870-145-1791') ('latombe', 'p', '552-190-2777')
Before Summary	tp = 94, fp = 25, fn = 23

After Summary	tp = 94, fp = 10, fn = 23

Pattern 7	ppatterns.append('\D?([2-9]\d{2})\D?\D?(\d{3})\D?(\d{4})[^0-9]')
Description	Pattern 6 included false positives from sequences of numbers such as 190277794 67698691 67698693 from the latombe file. [^0-9] was added to the end of the pattern to eliminate these numbers which were not phone numbers. However, 2 false positives remain from the nass file which were resolved later in Pattern 16.
True Positives	94
False Positives	2 ('nass', 'p', '910-769-9828') ('nass', 'p', '742-135-3522')
Before Summary	tp = 94, fp = 10, fn = 23
After Summary	tp = 94, fp = 2, fn = 23

With all of the phone numbers matched except for 2 false positives, the focus will be shifted back to the e-mails to reduce the number false negatives which is now 23.

Pattern 8	epatterns.append('([A-Za-z.]+)<del>@([A-Za-z.]+)\.edu')
Description	A new pattern was added to address the <del> characters in the latombe file. This resulted in 3 additional matches for e-mails.
True Positives	94
False Positives	2
Before Summary	tp = 94, fp = 2, fn = 23
After Summary	tp = 97, fp = 2, fn = 20

Pattern 9	epatterns.append('([A-Za-z.]+)\s<at symbol>\s([A-Za-z.]+)\.edu')
Description	When reviewing the manning file, the @ symbol was replaced with <at symbol> for the e-mail manning <at symbol> cs.stanford.edu. This also included spaces so the pattern with spaces was updated and added as a new pattern. This pattern identified two e-mails in the manning file.
True Positives	99
False Positives	2
Before Summary	tp = 97, fp = 2, fn = 20
After Summary	tp = 99, fp = 2, fn = 18

Pattern 10	epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.\D\D\D')
Description	Pattern 2 was changed from .edu to \D\D\D to find domains that end in something other than .edu. This corrected the e-mail uma@cs.stanford.EDU which ended with EDU in capital letters.
True Positives	100
False Positives	2
Before Summary	tp = 99, fp = 2, fn = 18
After Summary	tp = 100 , fp = 2, fn = 17

Pattern 11	epatterns.append('([A-Za-z.]+)\s?\s@\s?\s([A-Za-z.]+)\.edu')
Description	The dabo file has an e-mail with two spaces before and after the @ symbol. Pattern 3 was changed to \s?\s@\s?\s to handle one or two spaces and identified the dabo e-mail.
True Positives	101
False Positives	2
Before Summary	tp = 100 , fp = 2, fn = 17
After Summary	tp = 101, fp = 2, fn = 16

Pattern 12	epatterns.append('([A-Za-z.]+) AT ([A-Za-z.]+)\s*DOT\s*edu')
Description	In the subh file, there is the following e-mail: subh AT stanford DOT edu. Pattern 12 was able to find this e-mail.
True Positives	102
False Positives	2
Before Summary	tp = 101, fp = 2, fn = 16
After Summary	tp = 102, fp = 2, fn = 15

Pattern 13	epatterns.append('([A-Za-z.]+)\sWHERE\s([A-Za-z.]+)\sDOM\sedu')
Description	In the engler file, there is the following e-mail: engler WHERE stanford DOM edu Pattern 13 found this e-mail.
True Positives	103
False Positives	2
Before Summary	tp = 102, fp = 2, fn = 15
After Summary	tp = 103, fp = 2, fn = 14

Pattern 14	epatterns.append('([A-Za-z.]+)\s*\(\D*@([A-Za-z.]+)\.edu')
Description	The ouster file had two e-mails with similar forms: ouster (followed by &ldquo;@cs.stanford.edu&rdquo;) teresa.lynn (followed by "@stanford.edu") Pattern 14 was able to find both of these e-mails.
True Positives	105
False Positives	2
Before Summary	tp = 103, fp = 2, fn = 14
After Summary	tp = 105, fp = 2, fn = 12

Pattern 15	epatterns.append('([A-Za-z.]+)\&\#x40\;([A-Za-z.]+)\.edu')
Description	In the levoy file, the e-mails with the extra characters &#x40 were resolved with Pattern 15. This corrected ada&#x40;graphics.stanford.edu and melissa&#x40;graphics.stanford.edu.
True Positives	107
False Positives	2
Before Summary	tp = 105, fp = 2, fn = 12
After Summary	tp = 107, fp = 2, fn = 10

The focus will now shift back to the phone numbers to see if the 2 false positives can be resolved.

Pattern 16	ppatterns.append('[^2-9]\D?([2-9]\d{2})\D?\D?(\d{3})\D?(\d{4})[^0-9]')
Description	Two false patterns were generated in Pattern 7: ('nass', 'p', '910-769-9828') ('nass', 'p', '742-135-3522') [^2-9] was added to the beginning of Pattern 7 and this resolved the 2 false positives.
True Positives	107
False Positives	0
Before Summary	tp = 105, fp = 2, fn = 12
After Summary	tp = 107, fp = 0, fn = 10

After creating the 16 patterns above, the remaining 10 false negative e-mail addresses were evaluated and determined to be too difficult to resolve using the two-parenthesis format in

Python. Option 2 of the instructions will be followed below where the unresolved email addresses will be attempted for resolution in regexpal (regexpal.com).

Summary of False Negative E-mail Addresses

(Note: ✓ indicates the e-mail patterns were found with regexpal as shown on the next page)

Obfuscated E-mail	Correct E-mail from Gold Standard	Obfuscation Challenge
✓d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u	dlwh@stanford.edu	By placing dashes between each character, this a difficult pattern to develop with two sets of parentheses.
✓hager at cs dot jhu dot edu ✓serafim at cs dot stanford dot edu ✓uma at cs dot stanford dot edu ✓lam at cs.stanford.edu ✓pal at cs stanford edu ✓support at gradiance dt com ✓jks at robotics;stanford;edu	hager@cs.jhu.edu serafim@cs.stanford.edu uma@cs.stanford.edu lam@cs.stanford.edu pal@cs.stanford.edu support@gradiance.com jks@robotics.stanford.edu	These e-mails all use a similar obfuscation strategy of removing the @ symbol and using ; or dot for some periods (except lam). The support e-mail also uses dt instead of the more common dot.
function obfuscate(domain, name) { document.write('<a href="mai' + 'lto:' + name + '@' + domain + '">' + name + '@' + domain + '</' + 'a>') <script> obfuscate('stanford.edu','jurafsky')	jurafsky@stanford.edu	Dr. Jurafsky (author of the textbook Speech and Language processing) wrote a small program to obfuscate his e-mail. Python code could be written to merge jurafsky and stanford.edu to jurafsky@stanford.edu but this would likely only work for Dr. Jurafsky's customized website.
✓vladlen at <!-- die!--> stanford <!-- spam pigs!--> dot <!-- die!--> edu	vladlen@stanford.edu	Dr. Vladlen is obviously not a fan of spammers and used some humor in his obfuscation. However, regexpal will be attempted to find a pattern for this e-mail.

The final step in the process is to use Regex Pal patterns to resolve the remaining 10 obfuscated e-mails.

Note: The table format was changed to only include the False Negatives (fn) since these new regex patterns could create new false positives. This couldn't be evaluated without making changes to the Python code which are beyond the abilities of the author.

Regex Pattern 1	.* (?:dot .*)?at .* d.*t \w{3}
Description	This regex pal pattern matches 4 e-mails: hager at cs dot jhu dot edu serafim at cs dot stanford dot edu uma at cs dot stanford dot edu support at gradiance dt com
True Positives	110
False Positives	0
Before Summary	fn = 10
After Summary	fn = 6

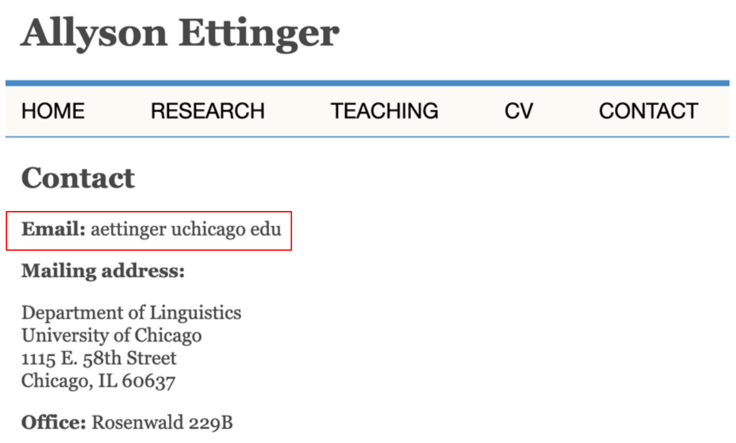
Regex Pattern 2	.*(?:at.?)?.*edu
Description	This regex pal pattern matches 3 e-mails: lam at cs.stanford.edu pal at cs stanford edu jks at robotics;stanford;edu
True Positives	
False Positives	0
Before Summary	fn = 6
After Summary	fn = 3

Regex Pattern 3	.* (?:dot .*)?at .* d.*t .* \w{3}
Description	This regex pal pattern matches vladlen at <!-- die!--> stanford <!-- spam pigs!--> dot <!-- die!--> edu The hager, serafim and uma patterns are also matched as with Regex Pattern 1.
True Positives	115
False Positives	0
Before Summary	fn = 6
After Summary	fn = 2

Regex Pattern 4	.*@.*\.-e-d-u
Description	This regex pal pattern matches d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u
True Positives	116
False Positives	0
Before Summary	fn = 2
After Summary	fn = 1

After changing the Python code patterns and utilizing regexpal.com, 116 of 117 e-mail and phone numbers have been matched correctly. The remaining false negative e-mail for jurafski@standord.edu will be discussed in the conclusion.

For the second part of Option 2, other examples of obfuscated e-mails will be reviewed. The first example is from Allyson Ettinger at the University of Chicago. Dr. Ettinger did a recent podcast on Data Skeptic where she discussed the limitations of the new Natural Language Processing technique called BERT (Bidirectional Encoding Representation in Transformers). Below is a screen shot of how Dr. Ettinger obfuscated her e-mail using the format aettinger uchicago edu. This would not have been detected by any of the patterns used in this exercise since both the @ symbol, AT/at and dot/. were not included. This would be difficult to find a pattern since many website URL's have similar patterns and many false positives might be created.



Allyson Ettinger

Contact

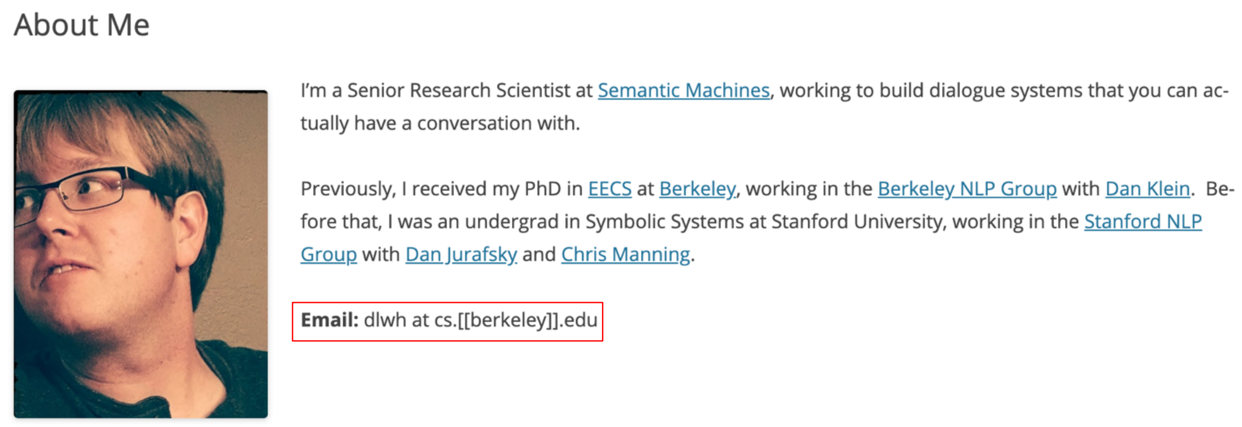
Email: aettinger uchicago edu

Mailing address:

Department of Linguistics
University of Chicago
1115 E. 58th Street
Chicago, IL 60637

Office: Rosenwald 229B

The obfuscated e-mail d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u discussed above belongs to David Hall. Mr. Hall also has his e-mail listed at dlwh.org where it is obfuscated as shown below. The use of [[]] around berkeley is better than plain text but it would be easy to develop a pattern to find this obfuscation.



About Me

I'm a Senior Research Scientist at Semantic Machines, working to build dialogue systems that you can actually have a conversation with.

Previously, I received my PhD in EECS at Berkeley, working in the Berkeley NLP Group with Dan Klein. Before that, I was an undergrad in Symbolic Systems at Stanford University, working in the Stanford NLP Group with Dan Jurafsky and Chris Manning.

Email: dlwh at cs.[[berkeley]].edu

Another example of e-mail obfuscation is from Silvan Mühlemann' blog at https://blog.xn--mhlemann-65a.ch/2008/07/20/ten-methods-to-obfuscate-e-mail-addresses-compared/. Mr. Mühlemann created 9 separate e-mails with 9 different obfuscation methods and then tracked the amount of spam to each account over 1.5 years. Here are the results:

Volume of Spam (MB) received by obfuscation method
CSS codedirection - 0
CSS display:none - 0
ROT13-Encryption - 0
Using AT's and DOT's - .084
Building with Javascript - .144
Replacing @ and . with Entities - 1.6
Splitting E-mail with comments - 7.1
Urlencode - 7.9
Plain text - 21

The top 5 methods for e-mail obfuscation generated the best results and use some type of advanced programming language similar to what Dr. Jurafsky utilized in the example above. The bottom 4 methods were similar to the ones utilized in this assignment and they can elude spam better than plain text but not as effectively as more advanced programming methods.

A search of phone number obfuscation did not provide as many solutions but a post on stackoverflow.com did provide an example where a phone number like 555-555-1234 could be obfuscated as 555-555-XXXX, where the XXXX is a clickable field that reveals the last four digits when clicked by a user. This approach also requires more advanced programming as described above for obfuscating e-mail addresses.

Conclusion

Regular Expressions are a powerful programming capability to extract text from the vast amount of data on the internet. However, smart programmers like Mr. Jurafsky (page 9) and Mr. Mühlemann (page 12) can use advanced programming techniques to obfuscate their contact information from most automated "bots" and web scraping techniques. For those who don't have these types of technical skills, the tips provided in this paper show additional methods that are most effective to obfuscate your information, so you are less susceptible to nuisance calls and e-mails.

[i] *Direct Marketing Association's Guidelines for Ethical Business Practices*, 2014, thedma.org/wp-content/uploads/DMA_Guidelines_January_2014.pdf.