Use of Regular Expression Patterns in a Contact Finder Application

IST-664 Assignment #2  
Leonard Armstrong

# Introduction

Span prevention is a common problem faced by Internet Service Providers (ISP), corporations, and home users. Spam, in this context, is defined as an unsolicited communication from a third party that has access to a person’s contact details, specifically, a person’s e-mail addresses and/or phone numbers. In today’s world it is common for organizations seeking potential targets[[1]](#footnote-1) to use simple bots to scrape the web for contact information. Given that the vast majority of e-mail addresses and phone numbers fit into certain character patterns, this type of information retrieval is relatively easy. Barely an inconvenience.

One mechanism used to combat the resulting spam that comes from this type of data mining is for an end-user to keep her/his contact details secret. However, this is impractical in a world where businesses want to make it easy for customers to discover their services and individuals want to provide open access to their friends and family. However, a hybrid approach is to obfuscate the common patterns within a computer’s files (such as the HTML code for a web page) such that the patterns show proper to the human eye after software rendering or as a result of minor human interpretation.

For example, displaying my email address as

**learmstr** at **Syr**acuse dot **edu**cational institute

would allow most readers to understand this as [learmstr@syr.edu](mailto:learmstr@syr.edu). However, many spam bots may have a more difficult time interpreting the underlying HTML script as a simple email address as I have avoided using a common [user@domain.tld](mailto:user@domain.tld) email address.

The research described herein uses a series of ordinary regular expression (regex) patterns on a corpus of 46 documents containing 117 individual e-mail addresses and phone numbers. The goal of the research is to discover as many e-mail addresses and phone numbers as possible (true positives) while not missing any available e-mails and phone numbers (false negatives) and avoiding any incorrectly found e-mails and phone numbers (false positives).

## Assumptions

The remained of this text assumes the reader is familiar with common regex syntax.[[2]](#footnote-2) Common phrases used in our English descriptions of the regular expression patterns that were created are defined as follows.

* The phrase *Word Character* is equivalent to the regex \w character class.
* The phrase *Digit* is equivalent to the regex \d character class.
* The phrase *String of X* refers to a succession of one or more of some set of X characters.

## Process

The process used in this research started with a predefine Python program is a provided file named ContactFinder.py. The global section of this file contained two list variables named epatterns (for e-mail regex patterns) and ppatterns (for phone number regex patterns). Regex patterns were incrementally added to these lists under the following rules.

Patterns added to epatterns were required to have exactly two capturing groups. The first capturing group was required to capture the user’s account name. The second capturing group was required to capture the email address’ subdomain. The top-level domain (TLD) was assumed to be .edu.

Patterns added to ppatterns were required to have exactly three capturing groups. The first capturing group was required to capture the three-digit area code. The second capturing group was required to capture the three-digit prefix. And the third capturing group was required to capture the four-digit line number.[[3]](#footnote-3)

Each pattern or pattern iteration will be presented using the following result table:

|  |  |  |
| --- | --- | --- |
| **Pattern** | *The actual regex pattern used.* | |
| **Description** | *An English description of the regex pattern.* | |
| **Sample Matches** | **Document** | **String** |
| *One or two sample strings that were capture with the pattern and the documents in which the strings were found.* | |
| **TP/FP/FN** | *Counts of the true positives (tp), false positives (fp) and false negatives (fn) that were discovered as a result of adding the regex in question to the prior list of regex expressions.*  *The result is shown in the form*: tp=*n1*, fp=*n2*, fn=*n3*. | |

Prior to starting the tp/fp/fn counts are: tp=0; fp=0; fn=117 with the ultimate goal being tp=117; fp=0; fn=0.

# E-mail Address Pattern Matching

E-mail address patterns are tackled first. In hindsight, not all e-mail addresses were able to be retrieved given the simple two-capturing group constraints of the initial version of ContactFinder.py. Therefore, the discussion of e-mail pattern matching is divided into two parts. Section 2.1 will present the patterns that conformed to the initial epatterns two-capturing group constraints. Then section 2.2 will present additional patterns that required updates ContactFunction.py functions.

## Standard E-mail Address Patterns

A total of 11 E-mail standard patterns were created that allowed for 37 e-mail addresses to be discovered. The first two patterns went through a series of iterations to broaden the scope of the addresses that could be discovered with a single pattern.

### Email-pattern #1

The first regex pattern went through a series of five iterations, generalizing the pattern to catch more and more samples. This pattern basically focuses on variations on the common e-mail address syntax using the “@” character and concluding with “.edu” directly tagged onto the subdomain.

#### Iteration #1.1

|  |  |  |
| --- | --- | --- |
| **Pattern** | ([\w]+)@([\w]+)\.edu | |
| **Description** | A string of word characters; Followed by an “@” character; Followed by another string of word characters; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| balaji | [balaji@stanford.edu](mailto:balaji@stanford.edu) |
| nass | [nass@stanford.edu](mailto:nass@stanford.edu) |
| **TP/FP/FN** | tp=4, fp=1, fn=113 | |

#### Iteration #1.2

This iteration extends iteration #1.1 to allow dot characters within the username or subdomain.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)@([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by an “@” character; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| nick | [nick.parlante@cs.stanford.edu](mailto:nick.parlante@cs.stanford.edu) |
| psyoung | [patrick.young@stanford.edu](mailto:patrick.young@stanford.edu) |
| **TP/FP/FN** | tp=19, fp=0, fn=98 | |

#### Iteration #1.3

This iteration extends iteration #1.2 to optionally allow one space character before and/or after the “@” character.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\ ?@\ ?([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by zero or one whitespace characters; Followed by an “@” character; Followed by zero or one whitespace characters; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| ullman | ullman @ cs.stanford.edu |
| **TP/FP/FN** | tp=22, fp=0, fn=95 | |

#### Iteration #1.4

This iteration extends iteration #1.3 to optionally allow *multiple* whitespace characters before and/or after the “@” character.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\ \*@\ \*([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by zero-to-many whitespace characters; Followed by an “@” character; Followed by zero-to-many whitespace characters; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| dabo | dabo @ cs.stanford.edu |
| **TP/FP/FN** | tp=23, fp=0, fn=94 | |

#### Iteration #1.5

The fifth and final iteration of the first pattern extends iteration #1.4 to optionally allow the TLD to be case independent. (i.e., *“edu” or “EDU”)*.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\ \*@\ \*([\w\.]+)\.[eE][dD][uU]** | |
| **Description** | A string of word characters and/or dots; Followed by zero-to-many whitespace characters; Followed by an “@” character; Followed by zero-to-many whitespace characters; Followed by another string of word characters and/or dots; Followed by the string “.edu” spelled in any variation of uppercase and/or lowercase characters. | |
| **Sample Matches** | **Document** | **String** |
| cheriton | uma@cs.stanford.EDU |
| **TP/FP/FN** | tp=24, fp=0, fn=93 | |

### Email Pattern #2

The second pattern deals with strings that use “at” instead of the “@” character. This pattern has two iterations.

#### Iteration #2.1

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\s+at\s+([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by one or more whitespace characters; Followed by the string “at”; Followed by one or more whitespace characters; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| lam | lam at cs.stanford.edu |
| **TP/FP/FN** | tp=25, fp=2, fn=92 | |

#### Iteration 2.2

This iteration provides no new true positives, but it does remove the two false negatives that appeared as a result of iteration #2.1

|  |  |  |
| --- | --- | --- |
| **Pattern** | **mail:\s+([\w\.]+)\s+at\s+([\w\.]+)\.edu** | |
| **Description** | The string “mail:”; Followed by one or more whitespace characters; Followed by a string of word characters and/or dots; Followed by one or more whitespace characters; Followed by the string “at”; Followed by one or more whitespace characters; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| jure | Removes false negatives such as “Server at cs.stanford.edu” found in jure. |
| **TP/FP/FN** | tp=25, fp=0, fn=92 | |

### Email Pattern #3

The third pattern looks for e-mail addresses that use the string “<at symbol>” in place of the “@” character.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\** **?<at symbol>\ ?([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by an optional whitespace character; Followed by the string “<AT SYMBOL>”; Followed by an optional whitespace character; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| lam | lam at cs.stanford.edu |
| **TP/FP/FN** | tp=27, fp=0, fn=90 | |

### Email Pattern #4

This pattern looks for e-mail addresses that replace the “@” character with the character hex code “&x40;”.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)****&#x40;([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by the string “&#X40;”; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| levoy | ada&#x40;graphics.stanford.edu |
| levoy | melissa&#x40;graphics.stanford.edu |
| **TP/FP/FN** | tp=29, fp=0, fn=88 | |

### Email Pattern #5

This pattern is similar to the first pattern but includes the string “<del>” before the “@” character.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)<del>@([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by the string “<del>@”; Followed by another string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| latombe | latombe<del>@cs.stanford.edu |
| latombe | asandra<del>@cs.stanford.edu |
| **TP/FP/FN** | tp=32, fp=0, fn=85 | |

### Email Pattern #6

This pattern finds email addresses that are interspersed with HTML-style comments. A regular expression pattern for finding HTML-style comments is “**<![^>]+>**”. That is: a less-than sign; followed by an exclamation point; followed by a string of one or more characters excluding the greater-than sign character; Finally, the RegEx is terminated when a greater-than sign is encountered.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\s+at\s+****<![^>]+>\s+([\w\.]+)\s+<![^>]+>\s+dot\s+<![^>]+>\s+edu** | |
| **Description** | A string of word characters and/or dots; Followed by one or more whitespace characters; Followed by the string “at”; Followed by one or more whitespace characters; Followed by an HTML comment (as defined previously); Followed by one or more whitespace characters; Followed by another string of word characters and/or dots; Followed by one or more whitespace characters; Followed by an HTML comment; Followed by one or more whitespace characters; Followed by the string “dot”; Followed by one or more whitespace characters; Followed by an HTML comment; Followed by one or more whitespace characters; Followed by Followed by the string “edu”. | |
| **Sample Matches** | **Document** | **String** |
| vladen | vladlen at <!-- die!--> stanford <!-- spam pigs!--> dot <!-- die!--> edu</div>[[4]](#footnote-4) |
| **TP/FP/FN** | tp=33, fp=0, fn=84 | |

### Email Pattern #7

This pattern looks for e-mail addresses that replace the “@” character with the string “at” and the “.” character with the string “dot”.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\ ?[aA][tT]\ ?([\w\.]+)\ [dD][oO][tT] edu** | |
| **Description** | A string of word characters and/or dots; Followed by an optional space; Followed by the string “at” spelled in any combination of uppercase and/or lowercase letters; Followed by an optional space; Followed by another string of word characters and/or dots; Followed by one space; Followed by the string “dot” spelled in any combination of uppercase and/or lowercase letters; Followed by the string “edu”. | |
| **Sample Matches** | **Document** | **String** |
| subh | subh AT stanford DOT edu |
| **TP/FP/FN** | tp=34, fp=0, fn=83 | |

### Email Pattern #8

This pattern is similar to the first pattern but also allows a descriptive string to be inserted between the username and the “@” character and also after the TLD.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\ [\(\w\"\ ]+@([\w\.]+).edu[\"\)]+** | |
| **Description** | A string of word characters and/or dots; Followed by a whitespace character; Followed by a non-null string containing any combination of word characters, left parentheses, double quotes or spaces; Followed by an optional whitespace character; Followed by the string “.edu”. Followed by a string containing one or more occurrences of double quotes or right parentheses. | |
| **Sample Matches** | **Document** | **String** |
| ouster | teresa.lynn (followed by "@stanford.edu") |
| **TP/FP/FN** | tp=35, fp=0, fn=82 | |

### Email Pattern #9

This is similar to pattern #8 but allows for the character hex code string “&ldquo;” to be used instead of the double quote character.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+)\ \(followed by \&ldquo;@([\w\.]+)\.edu** | |
| **Description** | A string of word characters and/or dots; Followed by a whitespace character; Followed by the string “(followed by &ldquo;@“; A string of word characters and/or dots; Followed by the string “.edu”. | |
| **Sample Matches** | **Document** | **String** |
| ouster | ouster (followed by &ldquo;@cs.stanford.edu |
| **TP/FP/FN** | tp=36, fp=0, fn=81 | |

### Email Pattern #10

This pattern allows for an odd variation that uses the string “WHERE” in place of the “@” character and “DOM” in place of the “.” character.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **((\w+) WHERE (\w+) DOM** | |
| **Description** | A string of word character; Followed by a space character; Followed by the string “WHERE”; Followed by a space character; Followed by a string of word characters; Followed by a space character; Followed by the string “DOM”. | |
| **Sample Matches** | **Document** | **String** |
| engler | engler WHERE stanford DOM edu |
| **TP/FP/FN** | tp=37, fp=0, fn=80 | |

### Email Pattern #11

The 11th pattern searches for the common e-mail format, but accepts “.com” as the TLD instead of “.edu” While this pattern does catch one email address, it only does so in union with changes to the email pattern generator found inside the match loop inside the process\_file function of ContactFinder.py. Thus, no new results are shown here but the results for this pattern are included with the patterns caught in section *2.2 Preprocessed Email Address Patterns*.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **(\w+)@(\w+).com** | |
| **Description** | A string of word characters; Followed by an “@” character; Followed by a string of word characters; Followed by the string “.com”. | |
| **Sample Matches** | **Document** | **String** |
| ullman | support@gradiance.com |
| **TP/FP/FN** | N/A | |

## Preprocessed Email Address Patterns[[5]](#footnote-5)

The remaining email patterns were unable to be recognized given the constraints of the two capture group regular expression model. Two capture these patterns the following algorithm changes were applied:

1. At the top of the processing loop in the process\_file function, a new preprocessing function was invoked. (See comment labeled # UPDATE #1 in the code.)
2. The preprocessing function (preprocess) looks for four patterns that either required more than one capture group or that needed to rearrange capture groups. (See definition and inline documentation for function preprocess in the code.)
3. If any of the four preprocessing patterns were found, the preprocessing function rewrites the discovered e-mail address into a canonical form of *user*@*subdomain.tld* where the user and subdomain could contain any combination of word characters or dots.
4. The preprocessing function returns either the updated canonical form e-mail address or the original line, to be processed normally.
5. Additionally, the process\_file function’s inner match loop was altered so the match pattern that accepted the user and subdomain portion of an e-mail address also allows .com address as well as .edu addresses. (See comment # UPDATE #2 in the code.)

### Preprocessing Pattern #1

In this sequence the first capture group was the user id. The second capture group, after replacing “dot” combinations with actual dots, was the subdomain. The sixth capture group was the top-level domain. See comment # Extended Pattern #1 in the code.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+) at (((([\w\.]+)|dot|dt)\s)+)(edu|com)\b'** | |
| **Description** | A string of word characters and/or dots; Followed by the string “ at ”; Followed by a sequence of one or more of the following: (a) a string of word characters and or dots or (b) the string “dot” or (c) the string “dt” a whitespace character; Followed by either the string “edu” or “com”. | |
| **Sample Matches** | **Document** | **String** |
| hager | hager at cs dot jhu dot edu |
| **TP/FP/FN** | tp=42, fp=0, fn=75 | |

### Preprocessing Pattern #2

This pattern was used to accept email addresses that were specified in reverse order (domain followed by user) via an obfuscate() function. In this sequence the first capture group was the entire domain and the second capture group was the user. See comment # Extended Pattern #2 in the code.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **obfuscate\('([\w\.]+)','([\w\.]+)'\)** | |
| **Description** | The string “obfuscate(‘”; Followed by a string of word characters and/or dots; Followed by the string “’,’”; Followed by a string of word characters and or dots; Followed by the string “’)”. | |
| **Sample Matches** | **Document** | **String** |
| jurafsky | obfuscate('stanford.edu','jurafsky') |
| **TP/FP/FN** | tp=43, fp=0, fn=74 | |

### Preprocessing Pattern #3

This pattern looks for subdomains that are separated by semicolons instead of dots. The username comes from the first capture group and the complete domain comes from the second capture group. See comment # Extended Pattern #3 in the code.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **([\w\.]+) at ((\w+;)+(\w+))** | |
| **Description** | A string of word characters and or dots; Followed by the string “ at ”; Followed by a string of word characters and/or semicolons; Followed by a “+” character; Followed by a string of word characters. | |
| **Sample Matches** | **Document** | **String** |
| jks | jks at robotics;stanford;edu |
| **TP/FP/FN** | tp=44, fp=0, fn=73 | |

### Preprocessing Pattern #4

This final preprocessing pattern looks for usernames and domains where individual characters are separated with dashes. The entire email address comes from the first capture group 1 after dashes are removed. See comment # Extended Pattern #4 in the code.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **((((\w)-)+)@-(((\w)-)+)\.((-(\w))+))** | |
| **Description** | A string of word-character-followed-by-a-dash sequences; Followed by the string “@-”; Followed by a string of word-characters-followed-by-a-dash sequences; Followed by a “.” character; Followed by a string of dash-character-followed-by-a-word-character sequences. | |
| **Sample Matches** | **Document** | **String** |
| dlwh | d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u |
| **TP/FP/FN** | tp=45, fp=0, fn=72 | |

At this point, all email addresses have been successfully recognized as true positives, there are no false positives and only phone numbers remain as the 72 false negatives.

# Phone Number Pattern Matching

Phone number pattern matching proved to be far simpler than email pattern matching. A total of three phone number patterns were used and all phone numbers were able to be captured with standard regex patterns that met the three-group rule for the ppatterns list.

## Phone Number Pattern #1

This pattern looks for phone number where the parts are separated by hyphen characters.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **(\d{3})[-\s](\d{3})[-\s](\d{4})** | |
| **Description** | A string of three digits; Followed by the dash character and a whitespace character; Followed by a string of three digits; Followed by the dash character and a whitespace character; Followed by a string of four digits. | |
| **Sample Matches** | **Document** | **String** |
| cheriton | 650-725-3726 |
| hager | 410-516-8000 |
| **TP/FP/FN** | tp=68, fp=0, fn=49 | |

## Phone Number Pattern #2

The second regex matches common phone number strings where the area code is surrounded by parentheses.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **\((\d{3})\)\s\*(\d{3})-(\d{4})** | |
| **Description** | An open parenthesis; Followed by a string of three digits; Followed by a close parenthesis; Followed by 0 or more whitespace characters; Followed by a string of three digits; Followed by the dash character; Followed by a string of four digits. | |
| **Sample Matches** | **Document** | **String** |
| zm | (650) 723-4364 |
| ok | (650) 723-9753 |
| **TP/FP/FN** | tp=115, fp=0, fn=2 | |

## Phone Number Pattern #3

The final phone number pattern is a variation on pattern #2 where the area code can be surrounded by square brackets.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **\[(\d{3})\]\s(\d{3})-(\d{4}** | |
| **Description** | An open square bracket; Followed by a string of three digits; Followed by a close square bracket; Followed by a whitespace character; Followed by a string of three digits; Followed by the dash character; Followed by a string of four digits. | |
| **Sample Matches** | **Document** | **String** |
| nass | [650] 723-5499 |
| nass | [650] 725-2472 |
| **TP/FP/FN** | tp=117, fp=0, fn=0 | |

# Conclusions

The work described here proves regex is able to find all email and phone numbers encountered in the 46-document corpus. A total of 14 regex expressions found all but eight e-mail addresses while conforming to a two-capture group requirement. Preprocessing the data with four more regex expressions and a change to the code’s standard e-mail address creation template allowed for the additional eight e-mail addresses to be found as well.

# Appendix A – Final Output of ContactFinder.py

This appendix provides the final output of ContactFinder.py where all eighteen regex expressions and preprocessing were applied.

Assuming ContactFinder.py called in directory with data folder

True Positives (117):

{('ashishg', 'e', 'ashishg@stanford.edu'),

('ashishg', 'e', 'rozm@stanford.edu'),

('ashishg', 'p', '650-723-1614'),

('ashishg', 'p', '650-723-4173'),

('ashishg', 'p', '650-814-1478'),

('balaji', 'e', 'balaji@stanford.edu'),

('bgirod', 'p', '650-723-4539'),

('bgirod', 'p', '650-724-3648'),

('bgirod', 'p', '650-724-6354'),

('cheriton', 'e', 'cheriton@cs.stanford.edu'),

('cheriton', 'e', 'uma@cs.stanford.edu'),

('cheriton', 'p', '650-723-1131'),

('cheriton', 'p', '650-725-3726'),

('dabo', 'e', 'dabo@cs.stanford.edu'),

('dabo', 'p', '650-725-3897'),

('dabo', 'p', '650-725-4671'),

('dlwh', 'e', 'dlwh@stanford.edu'),

('engler', 'e', 'engler@lcs.mit.edu'),

('engler', 'e', 'engler@stanford.edu'),

('eroberts', 'e', 'eroberts@cs.stanford.edu'),

('eroberts', 'p', '650-723-3642'),

('eroberts', 'p', '650-723-6092'),

('fedkiw', 'e', 'fedkiw@cs.stanford.edu'),

('hager', 'e', 'hager@cs.jhu.edu'),

('hager', 'p', '410-516-5521'),

('hager', 'p', '410-516-5553'),

('hager', 'p', '410-516-8000'),

('hanrahan', 'e', 'hanrahan@cs.stanford.edu'),

('hanrahan', 'p', '650-723-0033'),

('hanrahan', 'p', '650-723-8530'),

('horowitz', 'p', '650-725-3707'),

('horowitz', 'p', '650-725-6949'),

('jks', 'e', 'jks@robotics.stanford.edu'),

('jurafsky', 'e', 'jurafsky@stanford.edu'),

('jurafsky', 'p', '650-723-5666'),

('kosecka', 'e', 'kosecka@cs.gmu.edu'),

('kosecka', 'p', '703-993-1710'),

('kosecka', 'p', '703-993-1876'),

('kunle', 'e', 'darlene@csl.stanford.edu'),

('kunle', 'e', 'kunle@ogun.stanford.edu'),

('kunle', 'p', '650-723-1430'),

('kunle', 'p', '650-725-3713'),

('kunle', 'p', '650-725-6949'),

('lam', 'e', 'lam@cs.stanford.edu'),

('lam', 'p', '650-725-3714'),

('lam', 'p', '650-725-6949'),

('latombe', 'e', 'asandra@cs.stanford.edu'),

('latombe', 'e', 'latombe@cs.stanford.edu'),

('latombe', 'e', 'liliana@cs.stanford.edu'),

('latombe', 'p', '650-721-6625'),

('latombe', 'p', '650-723-0350'),

('latombe', 'p', '650-723-4137'),

('latombe', 'p', '650-725-1449'),

('levoy', 'e', 'ada@graphics.stanford.edu'),

('levoy', 'e', 'melissa@graphics.stanford.edu'),

('levoy', 'p', '650-723-0033'),

('levoy', 'p', '650-724-6865'),

('levoy', 'p', '650-725-3724'),

('levoy', 'p', '650-725-4089'),

('manning', 'e', 'dbarros@cs.stanford.edu'),

('manning', 'e', 'manning@cs.stanford.edu'),

('manning', 'p', '650-723-7683'),

('manning', 'p', '650-725-1449'),

('manning', 'p', '650-725-3358'),

('nass', 'e', 'nass@stanford.edu'),

('nass', 'p', '650-723-5499'),

('nass', 'p', '650-725-2472'),

('nick', 'e', 'nick.parlante@cs.stanford.edu'),

('nick', 'p', '650-725-4727'),

('ok', 'p', '650-723-9753'),

('ok', 'p', '650-725-1449'),

('ouster', 'e', 'ouster@cs.stanford.edu'),

('ouster', 'e', 'teresa.lynn@stanford.edu'),

('pal', 'e', 'pal@cs.stanford.edu'),

('pal', 'p', '650-725-9046'),

('psyoung', 'e', 'patrick.young@stanford.edu'),

('rajeev', 'p', '650-723-4377'),

('rajeev', 'p', '650-723-6045'),

('rajeev', 'p', '650-725-4671'),

('rinard', 'e', 'rinard@lcs.mit.edu'),

('rinard', 'p', '617-253-1221'),

('rinard', 'p', '617-258-6922'),

('serafim', 'e', 'serafim@cs.stanford.edu'),

('serafim', 'p', '650-723-3334'),

('serafim', 'p', '650-725-1449'),

('shoham', 'e', 'shoham@stanford.edu'),

('shoham', 'p', '650-723-3432'),

('shoham', 'p', '650-725-1449'),

('subh', 'e', 'subh@stanford.edu'),

('subh', 'e', 'uma@cs.stanford.edu'),

('subh', 'p', '650-724-1915'),

('subh', 'p', '650-725-3726'),

('subh', 'p', '650-725-6949'),

('thm', 'e', 'pkrokel@stanford.edu'),

('thm', 'p', '650-725-3383'),

('thm', 'p', '650-725-3636'),

('thm', 'p', '650-725-3938'),

('tim', 'p', '650-724-9147'),

('tim', 'p', '650-725-2340'),

('tim', 'p', '650-725-4671'),

('ullman', 'e', 'support@gradiance.com'),

('ullman', 'e', 'ullman@cs.stanford.edu'),

('ullman', 'p', '650-494-8016'),

('ullman', 'p', '650-725-2588'),

('ullman', 'p', '650-725-4802'),

('vladlen', 'e', 'vladlen@stanford.edu'),

('widom', 'e', 'siroker@cs.stanford.edu'),

('widom', 'e', 'widom@cs.stanford.edu'),

('widom', 'p', '650-723-0872'),

('widom', 'p', '650-723-7690'),

('widom', 'p', '650-725-2588'),

('zelenski', 'e', 'zelenski@cs.stanford.edu'),

('zelenski', 'p', '650-723-6092'),

('zelenski', 'p', '650-725-8596'),

('zm', 'e', 'manna@cs.stanford.edu'),

('zm', 'p', '650-723-4364'),

('zm', 'p', '650-725-4671')}

False Positives (0):

False Negatives (0):

set()

Summary: tp=117, fp=0, fn=0

[Done] exited with code=0 in 1.083 seconds

1. Where a target may be for a legitimate business interest or something more nefarious. [↑](#footnote-ref-1)
2. Regex definitions can be found in many locations on the Internet. One such location is <https://www.rexegg.com/regex-quickstart.html>. [↑](#footnote-ref-2)
3. Names for parts of a telephone number come from <https://talkroute.com/area-code-prefix-and-other-parts-of-a-phone-number/>. [↑](#footnote-ref-3)
4. Professor Vladden may want to switch to decaf. [↑](#footnote-ref-4)
5. Section 2.2 describes work done in fulfillment of Option 3 of the assignment. Specifically, additional patterns that did not strictly conform to the two capture group rule were added as preprocessing steps and the internal e-mail pattern generator was revised. [↑](#footnote-ref-5)