NLP Contact Finder using Regular Expressions

Alejandro Mora

11/5/2020

# Introduction

Regular expressions are a good way to search through text to find objects to match what you are looking for. Sometimes, functions through programming languages can be added and implemented to help with formatting of those objects. A good example of using Regular expressions are search of emails and phone numbers for identifying individuals. Given the program code in ContactFinder.py with the functions to format emails and base U.S. phone numbers, more regular expressions can be added to help locate the different forms throughout multiple documents of text.

Regular expressions for Emails

The first given regular expressions for emails were as follows, to modify and add to.

epatterns = []  
#orignal line, shows standard email (name@domain.edu)  
epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.edu')   
#original line, adding to original (name @ domain.edu) reads a space before and after the '@' character.  
epatterns.append('([A-Za-z.]+)\s@\s([A-Za-z.]+)\.edu')

Examples:

-Outcome: [ashishg@stanford.edu](mailto:ashishg@stanford.edu)

-Text: ashishg @ stanford.edu

-Outcome: [balaji@stanford.edu](mailto:balaji@stanford.edu)

-Text: [balaji@stanford.edu](mailto:balaji@stanford.edu)

-Outcome: [dabo@cs.stanford.edu](mailto:dabo@cs.stanford.edu)

-Text: dabo @ cs.stanford.edu

I decided to go with the route of adding or appending more expressions to the originals. The reason for this is because throughout the addition of new expressions, I got better results of more True Positives and less False Positives by adding more expressions, rather than adding to the previous expressions and receiving more and more False Positives.

Beginning Results: TP=42, FP=0, FN=75

Expression that reads Upper case letters of ‘EDU’ instead of ‘edu’.

#reads emails as (name@domain.EDU)  
epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.EDU')

Examples include:

-Outcome: [uma@cs.stanford.edu](mailto:uma@cs.stanford.edu)

-Text: [uma@cs.stanford.EDU](mailto:uma@cs.stanford.EDU)

Results: TP=43, FP=0, FN=74

Expression that reads ‘WHERE’ instead of an ‘@’ character and ‘DOM’ instead of a ‘.’ before ‘edu’. This is geared for emails written in a specific sentence form.

#reads emails as (name WHERE domain DOM edu)  
epatterns.append('([A-Za-z]+)\sWHERE\s([A-Za-z]+)\sDOM\sedu')

Examples include:

-Outcome: [engler@stanford.edu](mailto:engler@stanford.edu)

-Text: engler WHERE stanford DOM edu

Results: TP=44, FP=0, FN=73

Expression that reads a name and ‘at’ instead of ‘@’ while keeping the rest of the format the same. This is for partial sentence form emails.

#reads emails as (name at domain.edu)  
epatterns.append('([A-Za-z.]+)\sat\s([A-Za-z.]+)\.edu')

Examples include:

-Outcome: [cheriton@cs.stanford.edu](mailto:cheriton@cs.stanford.edu)

-Text: cheriton at cs.stanford.edu

-Outcome: [lam@cs.stanford.edu](mailto:lam@cs.stanford.edu)

-Text: lam at cs.stanford.edu

Results: TP=45, FP=2, FN=72

Expression that is more specific as there were special characters and letters between the name and the ‘@’. This one reads the possibility of having ‘<del>’ right before the ‘@’ character.

#reads emails as (name<del>@domain.edu)  
epatterns.append('([A-Za-z.]+)<del>@([A-Za-z.]+)\.edu')

Examples include:

-Outcome: [latombe@cs.stanford.edu](mailto:latombe@cs.stanford.edu)

-Text: latombe<del>@cs.stanford.edu

Results: TP=48, FP=2, FN=69

Expression that reads a sentence format with having ‘<at symbol>’ instead of ‘@’, while keeping the original last half the same.

#reads emails as (name <at symbol> domain.edu)  
epatterns.append('([A-Za-z.]+)\s<at symbol>\s([A-Za-z.]+)\.edu')

Examples include:

-Outcome: [manning@cs.stanford.edu](mailto:manning@cs.stanford.edu)

-Text: manning<at symbol> cs.stanford.edu

Results: TP=50, FP=2, FN=67

Expression that is another more specific one. This one reads ’ (followed by “@domain.edu”)’. It is one of the more specific cases, however if there would to be more examples, it is possible it is not an unlikely case.

#reads emails as (name (followed by "@domain.edu"))  
epatterns.append('([A-Za-z.]+)\s\(followed\sby\s\"@?(@?[A-Za-z.]+)\.edu')

Examples include:

-Outcome: [teresa.lynn@stanford.edu](mailto:teresa.lynn@stanford.edu)

-Text: teresa.lynn (followed by “@stanford.edu”)

Results: TP=51, FP=2, FN=66

Providing all these regular expressions for emails, there were a total of; 51 True Positives, 2 False Positives, and 66 False Negatives. These were all the working expression I could account for with the most, True Positives and the least False Positives.

# Regular Expressions for Phone Numbers

The program gives one base regular expression to start identifying phone numbers in common form.

ppatterns = []  
#original line  
ppatterns.append('(\d{3})-(\d{3})-(\d{4})')

Examples:

-Outcome: 650-723-1131

-Text: 650-723-1131

This expression reads the phone number as (3 numbers)-(3 numbers)-(4 numbers). I was able to find three more regular expressions to add to the original that found and correctly identified all the phone numbers in the given files with no False Positives or False Negatives for phone numbers.

Beginning Results: TP=51, FP=2, FN=66

This first expression is a modification of the original phone number expression. It adds the possibility of having a whitespace in between the first group of numbers and the ‘-’ character. It also adds a possible ‘-’ character after that space and required space after a ‘-’ character. After the second group, it states there should be ’ - ’ meaning a whitespace before and after the ‘-’ character.

#can read phone number as (### - ### - ###)  
# or (### ### ###)  
# or (###- ### - ###)  
ppatterns.append('(\d{3})\s?\-?\s(\d{3})\s\-?\s(\d{4})')

This Increased my True Positives from 51 to 56 and decreased my False Positives from 32 to 6 . This was more of a fix to original function giving the opportunity of spacing. It decreased the False Positives because before it was reading many lines of consecutive numbers and counting them as phone numbers.

Results: TP=56, FP=6, FN=61

This expression can read possible parenthesis or brackets around the first group as a lot of phone numbers are written and registers a possible ‘-’ character right after. This expression also looks at the possible space right after the parenthesis.

# reads phone number as ((###) -###-####)  
# or ((###)- ###-####)  
# or ((###) ###-####)  
# or ((###)###-####)  
# or ([###]-###-####)  
# or ([###]- ###-####)  
# or ([###] ###-####)  
# or ([###]###-####)  
ppatterns.append('\W(\d{3})\W\s?-?(\d{3})-(\d{4})')

This parenthesis’ possibility made a good connection to make the True Positives go from 54 to 64, but while adding the possible whitespace after the parenthesis, it adds to the total True Positives from 64 to 103.

Examples:

-Outcome: 650-814-1478

-Text: (650)814-1478

-Outcome: 650-724-6354

-Text: (650) 724-6354

-Outcome: 650-725-3897

-Text: (650) 725-3897

-Outcome: 650-723-5499

-Text: [650] 723-5499

Results: TP=103, FP=5, FN=14

This expression is more specific to phone numbers looking for the +1 extension before the numbers. Adding the “+1” matches the “+1” before a phone number just in case.

# reads a phone number as (+1 (###) ### ####)  
# or (+1 ### ### ####)  
# or (+1 ###########)  
ppatterns.append('\+1\s\(?(\d{3})\)?\s?(\d{3})\s?(\d{4})')

This expression only gets a couple of examples, but it shows that they are possible cases. Examples:

-Outcome: 650-723-5666

-Text: +1 650 723 5666

-Outcome: 650-725-3358

-Text: +1 (650) 725-3358

-Outcome: 650-725-9046

-Text: +1 650 725 9046

Results: TP=104, FP=2, FN=13

Providing all these regular expressions for phone numbers, after emails, there were a total of; 104 True Positives, 2 False Positives, and 13 False Negatives. The number of False positives went down as there were less falsely matched numbers in some files. These were all the working expression I could account for with the most, True Positives and the least False Positives.

# Part 2: Wrong Expressions

After working through many different expressions, there were about 4 expressions that I created for email searches, that did not match with the correct outcome and/or did not match at all. All the expressions written for phone numbers matches all possible numbers, but the last 13 False Negatives were emails I could not match. Inputting the expressions into regex101.com, they would match the desired email but did not match in the program. There were many emails I could not come up with a solution for, but I can show the ones that should work.

This expression is the same as the original function provided, however it has a ‘.com’ at the end to recognize emails with ‘.com’ domains. It also detects a ‘dt’ instead of a ‘.’ character and an ‘at’ instead of ‘@’.

#reads emails as (name@domain.com)  
#epatterns.append('([A-Za-z.]+)\sat\s([A-Za-z.]+)dtcom')

The reason this does not work is because in the functions’ code itself, the format for emails is set to ‘.edu’. The email will be recognized but it will result into a ‘.edu’ domain when it is not.

Example:

-Correct Outcome: [support@gradiance.com](mailto:support@gradiance.com)’

-Outcome Created: [support@gradiance.edu](mailto:support@gradiance.edu)’

-Text: support at gradiance dt com

In this expression, it matches with emails having a ‘-’ character before each letter. The function in the program recognizes two groups to an email, one before the ‘@’ and one after. By inserting a ‘-’ character in the middle of the groups, it creates an outcome including the character before every character.

#desired email match (-n-a-m-e@-d-o-m-a-i-n.edu)  
#epatterns.append('([-A-Za-z]+?)\-+@\-+([-A-Za-z]+?)\.[-edu]+?')

Example:

-Correct Outcome: [dlwh@stanford.edu](mailto:dlwh@stanford.edu)

-Outcome Created: [d-l-w-h@s-t-a-n-f-o-r-d-.edu](mailto:d-l-w-h@s-t-a-n-f-o-r-d-.edu)

-Text: [d-l-w-h-@-s-t-a-n-f-o-r-d](mailto:d-l-w-h-@-s-t-a-n-f-o-r-d)-.-e-d-u

This expression takes the ‘@’ and ‘.’ characters and replaces it with ‘at’ and ‘dot’. It will match in Regex but in the function it is accepting the first instance of ‘dot’ and ending the expression after that. It is recording the outcome as its second group and not outputting the correct form.

#Desired match (name at domain dot edu)  
#epatterns.append('([A-Za-z.]+)\sat\s([A-Za-z\sdot]+)\sedu')

Example:

-Correct Outcome: [serafim@cs.stanford.edu](mailto:serafim@cs.stanford.edu)

-Outcome Created: [serafim@cs](mailto:serafim@cs) dot stanford dot.edu

-Text: serafim at cs dot stanford dot edu

This expression produces the same outcome for the one above, just uppercase factors of ‘AT’ and ‘DOT’.

#desired match (name AT domain DOT edu)  
#epatterns.append('([A-Za-z.]+)\sAT\s([A-Za-z\sDOT]+)\sedu')

Example:

-Correct Outcome: [subh@stanford.edu](mailto:subh@stanford.edu)

-Outcome Created: [subh@stanford](mailto:subh@stanford) dot.edu

-Text: subh AT stanford DOT edu

One example of an obscure email address would be an email reporting in hexadecimal format. There could be a set format to accept ‘.edu’ or ‘.com’, the only problem would be the rest is in hexadecimal format. Finding patterns would be difficult and/or hard to write because of the many possible outcomes available; maybe lots of patterns. The other thing to consider would be the translation into an email format once you have gotten the correct pattern. Translating the letters or numbers would have to be inputted through functions to call into the email structure. After searching the web, I have found some small solutions to this and have not found one in the large scale. The challenges are looking through the files and being able to know where to start and end searches to make sure you are not reading random letters and characters.