IST664 Natural Language Processing: Homework 2

2019-05-10

1. Write 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. For each regular expression that you write or extend, list:

* Example(s) of email or phone numbers that match that pattern.
* Include example(s) of the obfuscated text that was matched from the file.
* A short English description of the expressions the pattern matches, demonstrating your understanding of regular expressions.
* The results (TP, FP and FN) before and after you added the expression.

For this exercise, I used 10 different regex patterns to identify obfuscated emails and phone numbers. Seven regex expressions were used for the former, three for the latter.

For the email regular expressions, these were separated into two capturing groups that contain both the user and the domain of the email address. The at symbol (@) and ending domain (i.e edu) were appended outside of these. The expressions I used to identify emails were the following:

* ([a-z]+)&#x40;(graphics.stanford).edu: the first capturing group matches on or more single character letters. After this first group, the regex expression asks to match exactly the ‘&#x40;’ characters, followed by a second group containing ‘graphics.stanford’. The final part asks to search for any address that ends in .edu. Email addresses matched by this expression include ada&#x40;graphics.stanford.edu.
* ([A-Za-z.]+)\s+@\s+([A-Za-z.]+)\.[A-Za-z]+: This second expression asks to match one or more uppercase or lowercase letters and special characters. The expression then indicates that the at sign should follow, surrounded by one or more special characters, in case there are whitespaces. The second group then asks for the regex to identify the same letters and characters as the first group. This expression is capable of matching email addresses such as ullman @ cs.stanford.edu
* ([a-z.]+)\b[<\>|(followed by &ldquo;][+.?@([a-z.]+).edu](mailto:+.?@([a-z.]+).edu): This expression starts asking the first group to contain one or more lowercase letters and special characters. It then asks for the middle section to be followed by either and open triangle or the expression ‘followed by &ldquo;’, any other special characters, and the at sign. The second group is the same as the first group, followed by ‘.edu’. Emails matched by this expression are: asandra<del>@cs.stanford.edu, liliana<del>@cs.stanford.edu.
* ([A-Za-z.]+)@([A-Za-z.]+)\.[A-Za-z]+: The fourth regex expression is similar to the second regex expression. However, the middle part of the expression changes, stating that the at sign should be alone, without any surrounding special characters. This expression catches all non-obfuscated addresses like: balaji@stanford.edu.edu
* ^([a-z]+).?\bat\b\s(\W.+).edu+: This expression matches any lowercase character, starting at the beginning of the sentence. It then indicates that the middle section could contain either a special character, the word ‘at’, or a special character. The second group contains any word or special character and ends with the domain name ‘.edu’. Email addresses matched by this expression include: vladlen at <!-- die!--> stanford <!-- spam pigs!--> dot <!-- die!--> edu
* (\w+)\b.[A-Z].\*\b(stanford).[A-Za-z]+.edu: The first group of this expression is a single word, followed by uppercase letters and special characters. It then has the word ‘stanford’, followed by any additional words and the domain name ‘edu’. This expression identifies emails like: engler WHERE stanford DOM edu.
* ([a-z]+).at <!-----.+>.(Stanford).+edu: the last email regular expression looks for any email that starts with lowercase letters, followed by the very specific expression <!---.+>, which asks to return the symbols <!-> and any additional matching special characters. The second group specifically asks for .Stanford, before ending with the expression ‘.edu’.

As for phone numbers, the regex expressions try to capture three groups instead of two. For these cases, the following three regular expressions were used:

* .+(\d{3}).[^0-9](\d{3{})[^0-9](\d{4}): Before the first group, the expression looks to match any character that is not a line terminator. Then, the first group matches any three successive digits. It then asks to match a single non-digit character, before repeating the first group. Then, it asks to once again match a single non-numeric character, before finally asking for four consecutive numbers.
* .?(d{3})[^0-9](\d{3})[^0-9](\d{4}): This second expression is similar to the prior one. However, the initial match at the beginning of the expression is optional, regarding special characters.
* (\d{3})-(\d{3})-(\d{4}): This final expression asks to match specific phone numbers that have three numbers, followed by three numbers, and ending in four numbers. However, between these numbers, there must be a dash sign.

Using these ten expressions, we get a result of 105 true positive results, no false positive results, and 12 false negative results.

1. Exercise 2
   1. With the regular expressions listed in exercise 1, the following list of regular expressions could not be matched:

* [d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u](mailto:d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u)
* hager at cs dot jhu dot edu
* jks at robotics;stanford;edu
* obfuscate(‘stanford.edu’, ‘jurafsky’)
* lam at cs.stanford.edu
* dbarros <at symbol> cs.stanford.edu
* manning <at symbol> cs.stanford.edu
* pal at cs stanford edu
* support at gradiance dt com

These regular expressions, though matched in the testing phase, could not be extracted from the documents because there were additional whitespace characters intertwined with the expressions that were not contemplated during the test phase.

In cases like ‘jks at robotics;stanford;edu’, the reason this expression couldn’t be matched was because of the semicolons that obfuscated the regular expression within the test phase. Similarly, the [d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u](mailto:d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u) email could not be retrieved because of the dashes between the letters. Though these were able to be identified individually, when the groups were constructed, I could not find a way of ignoring the dashes.

Finally, the email address ‘obfuscate(‘stanford.edu’, ‘jurafsky’) was built in a way not expected, as there were no ways of two groups with a middle at symbol to identify the expression. Thus, a different expression needed to be created to extract this email.

* 1. From what research I could find, the consensus is that the best way to obfuscate email addresses is by using HTML tags around and within the email address. Similarly, there are many websites that provide email obfuscation by using methods, such as encryption and appending different, non-relevant characters around and within the email address.

Finally, one last advice I’ve seen about obfuscation is to hide the addresses within Python and JavaScript code, which makes it hard for web crawlers to separate the code from the email address, as they intertwine seamlessly.

For example, emails could be created as:

* <href>[mailto:martin @ <domain>example</domain>.edu</href>endmail](mailto:martin%20@%20%3cdomain%3eexample%3c/domain%3e.edu%3c/href%3eendmail).
* <a href="javascript:window.location.href=atob('<?= base64\_encode("mailto:email@example.com") ?>')" style="unicode-bidi: bidi-override; direction: rtl;"><?= strrev("email@example.com") ?></a>
* <h3 id="email">[hello@gmail.com</h3](mailto:hello@gmail.com%3c/h3)>