Extracting information from reports using RegEx

Yogesh H. Kulkarni

Introduction

Many times, it is necessary to extract key information from reports, articles, papers, etc. A few examples are: names of companies-prices from financial reports, names of judges-jurisdiction from court judgments, and as mentioned in my previous article, account numbers from customer complaints, etc. Such extractions are part of Text Mining and are essential in converting unstructured data to structured form, typically for applying analytics/machine-learning later. Such entity extraction uses approaches such as “lookup”, “rules” and “statistical/machine learning”. In “lookup” based approaches, words from input documents are searched against pre-defined data dictionary. In “rules” based approaches pattern searches are made to find key information, whereas in “statistical” approaches supervised-unsupervised methods are used to extract the information. “Regular expression” (RegEx) is one of the “rules” based pattern search method.

Basic syntax

Python supports regular expressions (not fully Perl compatible though) by library called “re”. Instead of regular strings, search patterns are specified using raw strings “r”, so that backslashes and meta characters are not interpreted by python but sent to RegEx directly. Following is the basic syntax:

abc…	Letters		{m}	m Repetitions
123…	Digits		{m,n}	m to n Repetitions
\d	Any Digit		*	Zero or more repetitions
\D	Any Non-digit character		+	One or more repetitions
.	Any Character		?	Optional character
\.	Period		\s	Any Whitespace
[abc]	Only a, b, or c		\S	Any Non-whitespace character
[^abc]	Not a, b, nor c		^…$	Starts and ends
[a-z]	Characters a to z		(…)	Capture Group
[0-9]	Numbers 0 to 9		(a(bc))	Capture Sub-group
\w	Any Alphanumeric character		(.*)	Capture all
\W	Any Non-alphanumeric character		(abc|def)	Matches abc or def

Python sample code showing usage of RegEx is as follows:

```
import re
regex = r"([a-zA-Z]+) (\d+)"
text = "June 24"
match = re.search(regex, text)
if match:
   #   match.group() or match.group(0) always returns the fully matched string i.e. "June 24"
   print "Match: %s" % (match.group(0))
   #    match.group(1) match.group(2), ... will return the capture groups in order
   print "Month: %s" % (match.group(1)) # "June"
   print "Day: %s" % (match.group(2)) # "24"
else:
   # If re.search() does not match, then None is returned
   print "The regex pattern does not match. :("
```

Instead of “re.search”, which returns all the exact matches, “re.findall()” can be used to return all captured groups. “re.sub” is used to substitute another pattern as replacement for the given search pattern. For performance reasons it is recommended to compile the pattern first using “re.compile” and then use the RegEx object for searching, as shown below:

```
regex = re.compile(r"(\w+) Lamb")
text = "Mary had a little Lamb"
result = regex.search(text)
```

More information about RegEx usage in Python can be found at Regex One and in this AV article.

Use cases

Imagine writing code for searching telephone numbers like +91-9890251406 in a document, with multiple variations in format. With validations, the code will typically be surely more than 10 lines (sample here). But with RegEx it’s just about 2/3 of lines of code, and with high customizability.

Following are some of the frequently occurring scenarios where RegEx can offer substantial help. Please note that the examples shown could have alternate ways of getting same results, especially by using meta characters such as “/d” for “[0-9]” representing digits. In most of the examples, expressive and simplistic patterns are used here just for clarity and understandability.

Finding email

"^[a-zA-Z0-9_\-]+@[a-zA-Z0-9_\-]+\.[a-zA-Z0-9_\-]"	x@y.z
"@\w+.\w+"	Domain within email

Finding telephone number

"([0-9]{3}-){2}[0-9]{4}[^0-9]*$"	xxx-xxx-xxxx
"([0-9]{3}.){2}[0-9]{4}[^0-9]*$"	xxx.xxx.xxxx
"[0-9]{10}[^0-9]*$"	xxxxxxxxxx
"\\([0-9]{3}\\)[0-9]{3}-[0-9]{4}[^0-9]*$"	(xxx)xxx-xxxx

The first two cases differ only in "-"or "." and thus can combined using "(-|\.)"

A sample code for more elaborate phone number is as follows:

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # area code
    (\s|-|\.)?                    # separator
    (\d{3})                       # first 3 digits
    (\s|-|\.)                     # separator
    (\d{4})                       # last 4 digits
    (\s*(ext|x|ext.)\s*(\d{2,5}))? # extension
    )''', re.VERBOSE)
```

Finding date

"\d{2}-\d{2}-\d{4}"	xx-xx-xxxx without checking digits
"([2-9]|1[0-2]?)-[0-3][1-9]-[1-2][9|0][0-9]{2}[^0-9]*$"	xx-xx-xxxx with stricter usage
"([2-9]|1[0-2]?)/[0-3][1-9]/[1-2][9|0][0-9]{2}"	xx/xx/xxxx
"[1-2][9|0][0-9]{2}/([2-9]|1[0-2]?)/[0-3][1-9]"	xxxx/xx/xx

The date pattern used above are only numeric. There are other usages such as "27-Mar-1973" or "27 March 1973". Would let the reader to think about their RegEx patterns!!

Finding account /credit card number

"([0-9]{4}-){3}[0-9]{4}"	xxxx-xxxx-xxxx-xxxx
"[0-9]{16}"	xxxxxxxxxxxxxxxx

Adding linked Information

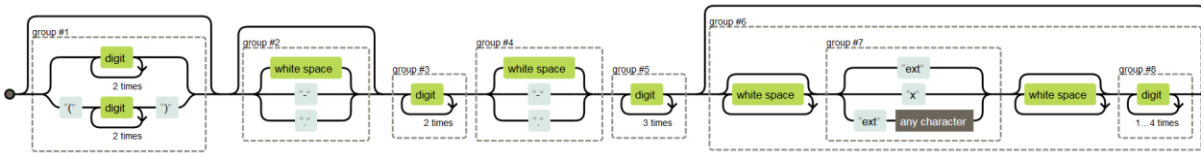
Citations in search papers, or judgements have pre-defined formats and they refer to an external document. It is possible to append the hyperlink information by replacing the citation text. For example, US legal judgements citation is like "17 U.S.C. § 107". The pattern is: text "U.S.C.", another space, a § mark, another space, a set of numbers, and optionally, a year inside a parenthetical. It can be replaced with a "<a href …/a>" hyper link to actual judgement it refers to.

Tools for development, testing and debugging

Although RegEx is powerful but it can get complicated for non-trivial tasks. More challenging it would be if you must understand (and debug) RegEx by someone else! There are quite a few friendly utilities which help in development and testing of RegEx. Try Regex 101. It gives facility to put your own text and try RegEx pattern. Here is one sample text to try on Phone number, account number, date patterns, mentioned before:

+919890251406 01110100 555.867.5309 01101000 9890251406 1.4142135623 01101001 27/3/1973 987-01-6661 01110011 202.555.9355 00100000 01101001 91-020-25898963 912025898963 3.14159265358979323846264338327 95 666-12-4895 01100001 202-555-9355 27-03-1973 00100000 01101000 (555) 867-5309 27-Mar-1973 2.718281828459 555-867-5309 01100101 01110011 01110011 555/867-5309

Sites like RegExper give visual representation of the RegEx search patter for better understanding. PhoneRegex search patter mentioned earlier, for phone number, gets visualized as below:



Once RegEx gives acceptable matches, the pattern can be used in programs. After good enough practice one can directly code the search patterns in the program itself.

All the RegEx patterns used here, with some minor modifications, can be used in programming language like Python, Perl, Java, etc. and in some of the popular text editors for Find-Replace functionality, like Microsoft Word (keep “Use Wildcard” option ON), OpenOffice and in IDEs like PyCharm. Comprehensive information about RegEx is here.

End note

RegEx is a versatile, portable and powerful way of extracting key information from textual data. Mastery over it can help automate many mundane tasks. Although, at times, it can get complicated and hard to develop-debug but owing to its immense capabilities it has become a must weapon in every programmer’s armour, especially for text analytics data scientists.

Let me conclude by giving a food for thought: Can RegEx be used to solve a crossword puzzle?