Exracting Information by matching regex, A Search Engine

Data Structures and Algorithms, 2017

INDEX

ABSTRACT

Include a sample input and try to come with output which is matched with the input using concept of regular expressions. The output will contain the strings related to the query searched by the user.

The objective of this project is to study whether information can be extracted on selected web pages, images, word and pdf documents using regular expressions. We evaluate the performance of our algorithm on multiple web pages, word documents, images and pdf documents.

The result we obtained is that we are able to extract information using regular expressions using the algorithm.

INTRODUCTION

REGULAR EXPRESSIONS

Regular expressions are building blocks that you can mix and match, it helps you learn how to write and debug your own patterns but also how to understand patterns written by others. Regular expressions provide a declarative language to match patterns within strings. They are commonly used for string validation, parsing, and transformation. They are available in a variety of dialects (also known as *flavors)* in a number of programming languages and text-processing tools, as well as many specialized application.

Extraction tasks can be accomplished by the use of carefully constructed regular expressions (regexes). Examples of such extractions include email addresses and software names (web collections), credit card numbers and social security numbers (email compliance), and gene and protein names (bioinformatics), etc. These entities share the characteristic that their key representative patterns (features) are expressible in standard constructs of regular expressions.

USING REGULAR EXPRESSION TO EXTRACT INFORMATION

Regular expressions are extremely useful in extracting information from text such as code, log files, spreadsheets, or even documents. And while there is a lot of theory behind formal languages we will try to explore the more practical uses of regular expressions.

The first thing to recognize when using regular expressions is that *everything is essentially a character*, and we are writing patterns to match a specific sequence of characters (also known as a string). Most patterns use normal ASCII, which includes letters, digits, punctuation and other symbols on your keyboard like %#$@!, but unicode characters can also be used to match any type of international text.

Basically, a regular expression is a pattern describing a certain amount of text. Their name comes from the mathematical theory on which they are based. We will usually find the name abbreviated to "regex".

The discussed example is actually a perfectly valid regex. It is the most basic pattern, simply matching the literal text regex. A "match" is the piece of text, or sequence of bytes or characters that pattern was found to correspond to by the regex processing software. Matches are highlighted in blue on this site.

\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b is a more complex pattern. It describes a series of letters, digits, dots, underscores, percentage signs and hyphens, followed by an at sign, followed by another series of letters, digits and hyphens, finally followed by a single dot and two or more letters. In other words: this pattern describes an email address. This also shows the syntax highlighting applied to regular expressions on this site. Word boundaries and quantifiers are blue, character classes are orange, and escaped literals are gray.

With the above regular expression pattern, we can search through a text file to find email addresses, or verify if a given string looks like an email address.

Similarly this project will be based on searching for particular strings which is searched or processed by the user. The term "string" or "character string" is used to indicate a sequence of characters. We can use regular expressions with whatever data we can access using the application or programming language we are working with.

Text Extraction HTML Documents comprise markup, which must be separated from the user-visible content of the page. To accomplish this, we must make the markup understandable to our program. This is done using an HTML parser.

Therefore by creating regular expression for various strings and queries and comparing it against out extracted text , we can assign score to various selected web pages, word and pdf documents and extract out the text file or the web page or in other words the information which has the highest score and thus highest relevance.

Requirement

With a regular expression, we can do powerful string parsing in only a handful lines of code, or maybe even just a single line. A regex is faster to write and easier to debug and maintain than dozens or hundreds of lines of code to achieve the same by hand.

We will use PYTHON for our project which is a popular high-level scripting language with a comprehensive built-in regular expression library

Regular Expressions :

Brackets

[abc] Find any character between the brackets
[^abc] Find any character NOT between the brackets
[0-9] Find any digit between the brackets
[^0-9] Find any digit NOT between the brackets
(x|y) Find any of the alternatives specified

Metacharacters

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd

BACKGROUND WORK

Search Engine

All search engines work using a 3 phase approach to managing , ranking and returning search results. But a lot of people have no idea what is happening behind that search box when they type in their search queries. So just how do Google, Bing and the rest of them work out what is on the web, what is relevant to your general query and which specific websites should be ranked highly?

There are three functions which need to be done:

Web Crawling

This is the means by which search engines can find out what is published out on the World Wide Web. Essentially, crawling is copying what is on web pages and repeatedly checking the multitude of pages to see if they are changed and make a copy of any changes found.

Indexing

Once a spider has crawled a web page, the copy that is made is returned to the search engine and stored in a data center. Data centers are huge, purpose built collections of servers which act as a repository of the all the copies of webpages being made by the crawlers. Google owns dozens of them dotted around the world, which it guards very closely and which are among the most hi-tech buildings in the world.

The repository of web pages is referred to as the 'Index', and it is this data store which is organized and used to provide the search results you see on the search engine. Indexing is the process of organizing the masses of data and pages so they can be searched quickly for relevant results to your search query.

The Algorithm

Finally, we have a huge collection of web page copies, word documents, pdf documents and images which are being constantly updated and organized so we can quickly find what you

are looking for. But we need a means by which they can be ranked in order of relevance to your search term – this is where the Algorithm comes into play.

The algorithm is a very complex and lengthy equation which calculates a value for any given site in relation to a search term. We don't know what the algorithm actually is, because search engines tend to keep this a closely guarded secret from competitors and from people looking to game the search engine to get to the top spots.

We have made a much simpler version of the search engine which is capable of searching for the queries searched by the user in the basic html pages. Our algorithm includes three basic functions which refine our search results and sort the pages accordingly.

Search Algorithm for regular expressions :

Each document is assigned a score, which is evaluated using the algorithm.

- The initial score is set to zero.

The score is incremented according to the conditions discussed in the following.

- Content contained in different tags carries different weightage.

CONTAINER WEIGHT

Heading **3**

Title **2**

Paragraph **1**

- Score increment = number of encounters * weight of container

The Document Frequency of a word is defined as the number of documents in which the searched word appears.

Inverse Document Frequency (IDF) is the inverse of Document Frequency. Mathematically, we calculate IDF as:

- IDF = log((total number of searchable documents+1)/number of documents containing keyword)

The score is, therefore, incremented by:

Score = Number of occurrences * weight of enclosing tag * IDF

The total length of a document also affects its score.

A document which simply contains more words is also more likely to contain the supplied keywords.

In order to counter this, we multiply the final score with the ratio of the number of relevant terms to the total number of terms.

- Score = Number of occurrences * weight of enclosing tag * IDF * (number of relevant terms / total number of terms)

If a regex encounters a web page/ word document/pdf document/image and a query is matched the score is increamented by the following algorithm.

Overview

Problem description:

The prime objective of using a search engine is to filter out the information which we need. The more exact we can be with the types of keywords/images used, the better your results will be.

We get a lot of traffic in the different documents, we need to extract and filter out the web pages/images/word and pdf documents which are relevant to us or in other words, extracting the web pages/documents/images which match our search.

Working model

Our algorithm is a working model for taking out relevant information from the several web pages, word and pdf documents by matching the search supplied by the user.

Specifically, we have a number of web pages, a few images and a word and pdf document, which we parse through our algorithm to extract the vital information which a user generally wants.

This model displays the information with the use of regular expressions which we made with the regular encryption which is generally followed when regular expressions are used.

This ensures that extra texts which is present in the documents or web pages are separated out. The score is sorted according to the hit ratio with respect to our supplied keywords and output is displayed accordingly.

Design description

Our search engine has three main function it uses to assign score to various documents.

Importance of keywords in different tags of a HTML document: The search engine runs through the entire document and separates different tags present in the HTML document and assign scores according to the matched keywords in title, heading or paragraph

IDF (Inverse Document Frequency): There are certain words like conjunctions and helping verbs which are present in the word documents, pdf documents, web pages as well as in our keyword. This hampers the search results because there might be less relevance of the actual words or noun and verbs we search for in certain documents but their score increases significantly due to these conjunctions and helping verbs.
Ratio of matched occurrences and total number of words present in the document.

After this refining of the documents, our regular expressions match the words which are present in the document, is then displayed in the output.

IMPLEMENTATION

Description of Programs:

Python: Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

SOURCE CODE:

Python Code:

```
#16BCE0194 #16BCE0765 #16BCE0768 #16BCE2308

import re
import math
import string
import textract
import os.path
from bs4 import BeautifulSoup

def readFile(filename):
	extension = os.path.splitext(filename)[1]
	if extension == ".html":
		f = open(filename)
		text = f.read()
		f.close()
		return text
	if extension in ['.pdf', '.docx', '.jpg', '.jpeg', '.png', '.gif']:
		text = textract.process(filename)
		return text


def extract_flight(filename):
	text = readFile(filename)
	#pnr
	print re.search(r'[A-Z0-9]{6}', text).group()
	#passenger name
	print re.search(r'MR\. (\w*) (\w*)', text).group()
	#flight number
	print re.search(r'[A-Z0-9][A-Z0-9] \d\d\d', text).group()
	#date
	print re.search(r'[A-Z][a-z]{2} [0-9]{2} [A-Z][a-z]{2}, 20[0-9]{2}',
text).group()
	#arr, dep time
	time_tuples = re.findall(r'\d+:\d\d [AP]M', text)
	for row in time_tuples:
```

```
                dept_time = time_tuples[0]
                arr_time = time_tuples[1]
        print dept_time, '->', arr_time

def extract_train(filename):
        text = readFile(filename)
        #pnr
        print re.search(r'<span>([0-9]{10})</span>', text).group(1)
        #passenger name
        name_tuples = re.findall(r'<td(.*)>([A-Z]+ [A-Z]+)</td>', text)
        for row in name_tuples:
                (attb, name) = row
                print name
        #train number, name
        print re.search(r'<span>([0-9]{5} / [A-Z\s]+)</span>', text).group(1)
        #scheduled departure
        print re.search(r'[0-9]+-[A-Z][a-z]{2}-20[0-9]{2}', text).group(), ' ',
re.search(r'[0-9]{2}:[0-9]{2}', text).group()

def idf(keyword):
        totalDocs = float(len(files))
        relDocsCount = 0.0
        for i in range(len(soup)):
                title = [word.strip(string.punctuation).lower() for word in
soup[i].title.string.split()]
                heading = [word.strip(string.punctuation).lower() for word in
soup[i].h2.string.split()]
                paragraph = [word.strip(string.punctuation).lower() for word in
soup[i].p.string.split()]
                for word in title:
                        if word == keyword:
                                relDocsCount+=1
                                break
                else:
                        for word in heading:
                                if word == keyword:
                                        relDocsCount+=1
                                        break
                        else:
                                for word in paragraph:
                                        if word == keyword:
                                                relDocsCount+=1
                                                break
        if relDocsCount == 0:
                return 0
        else:
                return math.log((totalDocs+1)/(relDocsCount))

def scorePage(hotSoup, keywords):               #keywords: list of keywords

        title = [word.strip(string.punctuation).lower() for word in
hotSoup.title.string.split()]
        heading = [word.strip(string.punctuation).lower() for word in
hotSoup.h2.string.split()]
        paragraph = [word.strip(string.punctuation).lower() for word in
hotSoup.p.string.split()]
        keywords = [word.strip(string.punctuation).lower() for word in keywords]

        tkt = 'n'                       #no ticket
        if ('flight' in heading):
                tkt = 'fl'              #flight
        elif ('train' in heading):
                tkt = 'tr'              #train

        score = 0.0
        relevance = 0.0

        length = len(title) + len(heading) + len(paragraph)
```

```
	for keyword in keywords:
		for word in title:
			if word == keyword:
				relevance+=1
				score+=2
		for word in heading:
			if word == keyword:
				relevance+=1
				score+=3
		for word in paragraph:
			if word == keyword:
				relevance+=1
				score+=1
		#score*=(idf(keyword))

	score *= relevance/length
	return [score, tkt]		#return score and ticket status

def scoreDoc(text, keywords):
		paragraph = [word.strip(string.punctuation).lower() for word in
text.split()]
		keywords = [word.strip(string.punctuation).lower() for word in
keywords]

		tkt = 'n'			#no ticket
		if ('flight' in paragraph):
			tkt = 'fl'		#flight
		elif ('train' in paragraph):
			tkt = 'tr'		#train

		score = 0.0
		relevance = 0.0

		length = len(paragraph)

		for keyword in keywords:
			for word in paragraph:
				if word == keyword:
					relevance+=1
					score+=1

		score *= relevance/length
		return [score, tkt]		#return score and ticket status


#main
#TODO: ADD INDIVIDUAL NAMES
files = ['doc0.docx', 'doc1.docx', 'doc2.docx', 'page0.html', 'page1.html',
'page2.html', 'doc3.docx', 'page3.html', 'page4.html', 'page5.html','page6.html',
'page7.html', 'page8.html', 'page9.html', 'try.pdf', 'picture.png', 'picture2.png',
'ppt1.pptx']
			#list of files`
soup = []
			#list of BeautifulSoup objects

#parse [files] and convert to BeautifulSoup objects
for i in range(len(files)):
	soup.append(BeautifulSoup(open(files[i]), 'html.parser'))

keywords = [word.strip(string.punctuation).lower() for word in raw_input('Search
here: ').split()]		#list of keywords

#is user looking for trains or flights?
wantFlight = False
wantTrain = False
if 'flights' in keywords or 'flight' in keywords or 'plane' in keywords:
	wantFlight = True
```

```
if 'trains' in keywords or 'train' in keywords or 'irctc' in keywords:
	wantTrain = True
if 'trips' in keywords:
	wantFlight = True
	wantTrain = True
scores = {}                                     #stores file name against score of
page
tickets = {}                                #stores file name against ticket status
l = []                                          #stores result of scorePage function

#assign scores according to algorithm to each file
for i in range(len(files)):
	extension = os.path.splitext(files[i])[1]
	if extension == ".html":
		hotSoup = BeautifulSoup(open(files[i]), 'html.parser')
		l = scorePage(hotSoup,keywords)
		scores[files[i]] = l[0]
		tickets[files[i]] = l[1]
	elif extension in ['.pdf', '.docx', '.jpg', '.jpeg', '.png', '.gif']:
		l=scoreDoc(readFile(files[i]), keywords)
		scores[files[i]] = l[0]
		tickets[files[i]] = l[1]


#display files and scores, sorted by score
border = '********************'
for file in sorted(scores, key=scores.get, reverse=True):
	if wantFlight == True and tickets[file] == 'fl':
		print border + '\n', file
		extract_flight(file)
		print border
	elif wantTrain == True and tickets[file] == 'tr':
		print border + '\n', file
		extract_train(file)
		print border
	else:
		print file, scores[file]
```

Test Cases and Snapshots:

```
Search here: india
page7.html 0.275862068966
page0.html 0.182767624021
page1.html 0.150170648464
page4.html 0.0859649122807
doc0.docx 0.0844827586207
picture2.png 0.0430107526882
try.pdf 0.0253164556962
doc2.docx 0.0252986647927
doc3.docx 0.0017793594306
doc1.docx 0.0
page6.html 0.0
picture.png 0.0
page2.html 0.0
page3.html 0.0
page5.html 0.0
page8.html 0.0
page9.html 0.0
Salonees-MacBook-Pro:TOC Project saloneegupta$
```

```
Salonees-MacBook-Pro:~ saloneegupta$ ls
Applications            Downloads               Music
Creative Cloud Files    Dropbox                 Pictures
Desktop                 Library                 Public
Documents               Movies                  VirtualBox VMs
Salonees-MacBook-Pro:~ saloneegupta$ cd Desktop
Salonees-MacBook-Pro:Desktop saloneegupta$ cd 'TOC Project'
Salonees-MacBook-Pro:TOC Project saloneegupta$ python FINAL.py
No handlers could be found for logger "bs4.dammit"
Search here: Jaisankar
doc1.docx 0.0
page0.html 0.0
doc0.docx 0.0
page6.html 0.0
picture.png 0.0
picture2.png 0.0
page2.html 0.0
doc3.docx 0.0
page3.html 0.0
try.pdf 0.0
doc2.docx 0.0
page4.html 0.0
page5.html 0.0
page8.html 0.0
page7.html 0.0
page1.html 0.0
page9.html 0.0
Salonees-MacBook-Pro:TOC Project saloneegupta$
```

```
Salonees-MacBook-Pro:~ saloneegupta$ cd Desktop
Salonees-MacBook-Pro:Desktop saloneegupta$ cd 'TOC Project'
Salonees-MacBook-Pro:TOC Project saloneegupta$ python FINAL.py
No handlers could be found for logger "bs4.dammit"
Search here: indian spy
doc0.docx 0.0620689655172
page1.html 0.061433447099
picture2.png 0.0430107526882
page4.html 0.00175438596491
doc1.docx 0.0
page0.html 0.0
page6.html 0.0
picture.png 0.0
page2.html 0.0
doc3.docx 0.0
page3.html 0.0
try.pdf 0.0
doc2.docx 0.0
page5.html 0.0
page8.html 0.0
page7.html 0.0
page9.html 0.0
Salonees-MacBook-Pro:TOC Project saloneegupta$
```

```
Salonees-MacBook-Pro:TOC Project saloneegupta$ python FINAL.py
No handlers could be found for logger "bs4.dammit"
Search here: queue jumpers
doc1.docx 0.00813008130081
page0.html 0.0
doc0.docx 0.0
page6.html 0.0
picture.png 0.0
picture2.png 0.0
page2.html 0.0
doc3.docx 0.0
page3.html 0.0
try.pdf 0.0
doc2.docx 0.0
page4.html 0.0
page5.html 0.0
page8.html 0.0
page7.html 0.0
page1.html 0.0
page9.html 0.0
Salonees-MacBook-Pro:TOC Project saloneegupta$
```

Conclusion

Successfully implemented our algorithm where a program searches document for specified keywords and returns a list of document where the keywords were found.

Using the regular expressions in the program, the relevant information is extracted.

REFERENCES

Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, "Regular Expression Learning for Information Extraction", IBM Almaden Research Center San Jose, CA 95120

"The Premier website about Regular Expressions", Copyright © 2003-2016 Jan Goyvaerts.

Arnaud Rosier, MD,[1] Anita Burgun, MD, PhD,[1] and Philippe Mabo, MD, PhD[2] ," Using regular expressions to extract information", American Medical Informatics Association Published online 2008.