正则表达式

readme

- 正则表达式在NLP任务中极其重要，学习此文档前需学习前置知识 python字符串处理
- 正则表达式需要重点掌握的内容为 `re.search()` ，`re.sub()` 以及各种 `pattern` 的构建

正则表达式

- re: regular expression，用于匹配字符
- re模块官方文档：https://docs.python.org/zh-cn/3.8/library/re.html

导入库

```
import re
#python自带库，不用pip安装
```

基本规则

单个字符匹配表

字符	功能

.	匹配任意1个字符
*	匹配 `*` 前面的字符0-n次， `ab*` 会匹配 `'a'` ， `'ab'` ，或者 `'a'` 后面跟随任意个 `'b'`
+	匹配 `+` 前面的字符1-n次， `ab+` 会匹配 `'ab'` ，或者 `'a'` 后面跟随任意个 `'b'` ，但不会匹配 `'a'` ，相当于 `{1,}`
?	匹配 `?` 前面的字符0次或1次， `ab?` 会匹配 `'a'` 或者 `'ab'` ，相当于 `{0,1}`
{m}	匹配 `m` 前的字符 `m` 次重复，比如 `a{2}` 将匹配2个 `'a'` ,但是不能是1个或0个
{m,n}	匹配 `{m,n}` 前的字符 `m` 到 `n` 次重复，尽可能取多， `a{3,5}` 将匹配 3 到 5个 `'a'` ， `{,n}` 表示下界为0， `{m,}` 表示上界为无数次
[ ]	匹配 `[]` 中的字符，具体用法为下面的自定义匹配
\d	匹配数字 0-9
\D	匹配非数字
\s	匹配空白字符，即空格、tab等
\S	匹配非空白字符
\w	匹配单词字符，即a-z、A-Z、0-9
\W	匹配非单词字符
\b	`\b` 表示单词边界。正则表达式 `\btest\b` 会匹配字符串中独立的单词 `test` ，而不会匹配像 `contest` 或 `testing` 中的 `test`

自定义匹配

例子	可匹配
[abc2#]	`a` , `b` , `c` , `2` , `#` 其中任意一个
[^awe]	非 `a` , `w` , `e` 的任意字符
[a-g]	`a` ～ `g` 之间的任意一个字符
[^a-g0-3]	匹配 `a` ～ `g` , `0` ~ `3` 之外的任意一个字符

- 下面的例子看不懂可以先看下面的基础函数

```
1 prog = re.compile('[abc][abc]')
2 result = prog.search('qbcd')
```

```
print(result)
```

- output: <re.Match object; span=(1, 3), match='bc'>

re模块中的基础函数

re.compile(pattern,flags = 0)

将正则表达式的样式编译为一个 正则表达式对象（正则对象），可以用于匹配，通过这个对象的方法 `match()`，`search()` 等，如下例中的prog，如果prog使用频繁建建议使用compile函数

```
prog = re.compile(pattern)
result = prog.match(string)

# 等价于
result = re.match(pattern, string)
# 例子在下面
```

re.match(pattern,string,flags = 0)

用于在给定的字符串开头进行匹配

- pattern：要匹配的正则表达式
- string：要匹配的 字符串
- flags：可选参数，用于指定匹配的模式，例如忽略大小写等。

flags 参数表

`re.I`	忽略正则表达式的大小写， `[A-Z]` 也能匹配小写字母
`re.M`	正则表达式的 `^` 可以让每行当作匹配开始
`re.S`	正则表达式中的 `.` 能够匹配所有字符，默认是匹配出了 `\n` 以外的所有字符

匹配对象具有以下常用方法

- group()：返回匹配的字符串

- start()：返回匹配的起始位置的索引
- end()：返回匹配的结束位置的索引。
- span()：返回一个元组，包含匹配的起始和结束位置的索引。

```
pattern = 'abc'
string = 'abc efg'
result = (re.match(pattern, string))
print(result)
# output:<re.Match object; span=(0, 3), match='abc'>

print(result.group())
# output: 'abc'

print(result.start())
# output: 0

print(result.end())
# output: 3

print(result.span())
# output: (0,3)
```

由于match只能在字符开头匹配，那么以下的字符就无法被匹配出来

```
pattern = 'abc'
string = 'efg abc'
result = (re.match(pattern, string))
print(result)
#output: None
```

re.search(pattern,string,flags = 0)

```
pattern = 'abc'
string = 'efg abc'
result = (re.search(pattern, string))
print(result)
# output:<re.Match object; span=(4, 7), match='abc'>
```

```
6
7 print(result.group())
8 # output: 'abc'
```

re.findall(pattern,string,flags = 0)

在给定的字符串中查找所有匹配的模式，并以列表的形式返回匹配结果。

```
1 pattern = 'abc'
2 string = 'abc abc'
3 result = (re.findall(pattern, string))
4 print(result)
5 # output: ['abc', 'abc']
```

常用方法介绍

一个简单示例

- `|` ：希望匹配许多表达式中的一个时可以使用，比如下面的例子中希望能够匹配到星期几和几天的其中一个
- `\d`：匹配 `0-9` 中的一个
- `.`：匹配任意字符

```
1 pattern = '星期.|\d天'
2 string = '今天星期一，距离星期天还有6天'
3 result = (re.search(pattern, string))
4 print(result.group())
5 # output: 星期一
6
7 result = (re.findall(pattern, string))
8 print(result)
9 # output: ['星期一', '星期天', '6天']
```

?：可选匹配，出现0或1次

```
pattern = '星期(四)?'
string = '这个星期要调休'
result = (re.search(pattern, string))
print(result.group())
# output: 星期

pattern = '星期(.)?'
string = '星期五要调休'
result = (re.search(pattern, string))
print(result.group())
# output: 星期五

pattern = '星期(.)?'
string = '这个星期要调休'
result = (re.search(pattern, string))
print(result.group())
# output: 星期要
```

组合使用例子

- 切记：
- `(word1|word2)` 用于识别词语 `word1` 或 `word2`
- `[ab]` 用于识别字符 `a` 或 `b`

例1：re.search('\d+\.?\d*', '12.5')

```
string = '12.5'
result = re.match('\d+\.?\d*', string)
```

- 拆解：
- `\d` 表示一个数字， `+` 表示匹配至少1次
- `\d+` 表示匹配至少一个数字
- `\.` 表示 `.` ， `?` 表示出现0次或1次
- `\.?` 表示 `.` 出现0次或1次
- `*` 表示匹配任意次， `\d*` 表示匹配任意次数字，即数字可有可无
- 可以被匹配的string如 `12.5` ， `12.` ， `5.3` ， `13`

- 不能被匹配的string如 `.3`

例2：复杂例子

- 我有一句话，我遇到这个开头我就需要把这句话提取，这个开头是这几种：
- `您提到的关于喜剧话题的主题，`
- `您提到在关于悲剧话题的主题，`这里与上句逗号不同，同时这里也可能是冒号 `：`或`:` 或句号 `。`
- `关于电影话题的案例`

- 这里该怎么分析呢?

1. 因为是要以这个开头，所以必须是 `^` 开头
2. `您提到` 可有可无，所以可以用 `(您提到)?`
3. 第一句后面是 `的` 字，第二句后面是 `在` 字，但是第三句中两个字都没有，那么也是可有可无，变成了 `的?在?`
4. `关于` 必有，所以可以要有 `关于`
5. `话题` 前的字是变化的，可以是 `.*话题` ， `.*` 代表任意个字符
6. `的` 字必有
7. `主题` 和 `案例` 二选一，所以是 `(主题|案例)`
8. 可能会出现各种标点符号 `[,, :: 。]` ，也可能没有标点符号，所以是 `[,, :: 。]?`
9. 最后合起来就是'`^(您提到)?的?在?关于.*话题的(主题|案例)[,, :: 。]?`'

分割函数、替换函数

re.split(pattern,string,maxsplit,flags)

- `maxsplit` 为限定切分多最大次数

```python
string = 'abc1def23gh4i'
result = re.split('\d+',string)
print(result)
# output: ['abc', 'def', 'gh', 'i']

result = re.split('\d+',string,maxsplit = 2)
print(result)
# output: ['abc', 'def', 'gh4i']
```

re.sub(pattern,repl,string,count,flags)

- `repl` 代表要替换成的字符串
- `count` 表示匹配后替换的最大次数

```
1 string = '我喜欢吃健康的水果，所以我长得高'
2 re.sub('我喜欢.*的水果','我不喜欢水果',string)
```

- output：'我不喜欢水果，所以我长得高'

常用例子

- 写prompt的时候提示大模型：Please output in the following format: <lable>
- 输出类似于：For the word "gam / leisure", one possible label could be <leisure>.
- 提取出<>的内容得到：
- leisure

```
1 result = 'For the word "gam / leisure", one possible label could be <leisure>.'
2 match = re.search(r'<(.*?)>', result)
3 result = match.group(1)
```

leetcode10.正则表达式

- 链接：正则表达式匹配
- 题目描述

10. 正则表达式匹配

已解答

困难 相关标签 相关企业 A文

给你一个字符串 `s` 和一个字符规律 `p`，请你来实现一个支持 `'.'` 和 `'*'` 的正则表达式匹配。

- `'.'` 匹配任意单个字符
- `'*'` 匹配零个或多个前面的那一个元素

所谓匹配，是要涵盖 **整个** 字符串 `s` 的，而不是部分字符串。

示例 1：

> **输入：** s = "aa", p = "a"
> **输出：** false
> **解释：** "a" 无法匹配 "aa" 整个字符串。

示例 2：

> **输入：** s = "aa", p = "a*"
> **输出：** true
> **解释：** 因为 '*' 代表可以匹配零个或多个前面的那一个元素，在这里前面的元素就是 'a'。因此，字符串 "aa" 可被视为 'a' 重复了一次。

示例 3：

> **输入：** s = "ab", p = ".*"
> **输出：** true
> **解释：** ".*" 表示可匹配零个或多个（'*'）任意字符（'.'）。

```
# 答案仅供参考，几乎没任何实际价值，还需自行到lc上查看答案
# 有待补充
```

结语

- 学完请移步至 英文文档评测数据预处理