STAT243 Lecture 5.1 Text manipulation, string processing and regular expressions (regex)

Logic

- Python中的文本处理与UNIX, R, Perl等语言有许多相似之处, 许多概念和工具源自UNIX系统.
- 本文中提到的 "string" 指的是由 character 组成的序列, 可包含数字, 空白符(包括换行符)和特殊字符, 通常存储为 `str` 类对象.
- 我们将重点介绍Python中处理字符串的功能, 特别是使用 `re` 包进行正则表达式操作.

Remark

`re` 包提供的是Perl风格的正则表达式, 但不支持 named character classes (如 `[:digit:]`), 应该使用 `\d` 或 `[0-9]` 等字符类.

1 Finding Patterns 模式查找

1.1 查找单个匹配项

在 Python 中, 可以使用 matching function `re.search()` 来查询结果以获取匹配内容和位置信息.

Python

```python
import re
text = "Here's my number: 919-543-3300."
m = re.search("\\d+", text)     # 也可以使用 m = re.search(r"\d+", text)
print(m)          # <re.Match object; span=(18, 21), match='919'>
print(m.group())  # '919'
print(m.start())  # 18
print(m.end())    # 21
print(m.span())   # (18, 21)
```

Remark

1. 注意: `re.search()` 只返回第一个匹配项
2. Discussion of special characters 解释了为什么我们使用 `\\d` 而不是 `\d`.
   大致原因是: 因为有 `\n` 这类特殊字符的存在, `\d` 并不会被直接传入正则引擎, 而是会先被误编译成特殊字符.
   若使用 `\\d`, 则 Python 解释器会先将 `\\d` 编译为 `\d`, 随后再传入正则引擎匹配

1.2 查找所有匹配项

若要获取所有匹配项, 可使用 `re.findall()`:

Python

```python
re.findall("\\d+", text)  # ['919', '543', '3300']
                          # 也可以使用 re.findall(r"\d+", text)
```

等价于先使用 `re.compile()` 进行编译再使用 `re.findall()`:

Python

```python
pattern = re.compile("\\d+")
```

```
re.findall(pattern, text)
```

> ⚠ Remark

- 显式编译正则表达式(`re.compile`)在复杂模式或重复使用时更高效
- 正则表达式是一种可编译的独立语言.

1.3 正则表达式的 flags

忽略大小写匹配:

Python

```python
text = "That cat in the Hat"
re.findall("hat", text, re.IGNORECASE)  # ['hat', 'Hat']
```

其他有用的正则表达式标志包括

- `re.VERBOSE` (允许编写带注释的正则表达式)
- `re.MULTILINE` (多行匹配)

1.4 使用 list comprehension 处理多个字符串

可使用 list comprehension 处理多个字符串:

Python

```python
def return_group(pattern, txt):
    m = re.search(pattern, txt)
    return m.group() if m else None

texts = [
    "Here's my number: 919-543-3300.",
    "hi John, good to meet you",
    "They bought 731 bananas",
    "Please call 1.919.554.3800"
]
[return_group(r"\d+", s) for s in texts]  # ['919', None, '731', '1']
```

> ⚠ Remark

需注意处理未匹配的情况, 否则会报错

1.5 位置匹配与重复匹配

可使用`^`和`$`匹配字符串的开头或结尾:

Python

```python
text = "hats are all that are important to a hatter."
re.findall(r"^hat\w+", text)  # ['hats']
```

使用重复符号`{}`匹配多次出现:

Python

```python
text = "Here's my number: 919-543-3300. They bought 731 hats. Please call 1.919.554.3800."
```

```python
re.findall(r"\d{3}[-.]\d{3}[-.]\d{4}", text)  # ['919-543-3300', '919.554.3800']
```

更通用的电话号码匹配(允许前缀"1-"或"1."):

Python

```python
re.findall(r"((1[-.])?(\d{3}[-.]){1,2}\d{4})", text)
# [('919-543-3300', '', '543-'), ('1.919.554.3800', '1.', '554.')]
```

> ⚠ **Remark**
>
> - 上述正则表达式可能匹配无效电话号码, 需谨慎设计
> - 上述正则表达式使用了 capturing group `()`, 因此返回了一个 list of tuple, capturing group 会在下文中详细解释

1.6 使用 `finditer` 进行迭代 (惰性) 匹配

对于大文本, 建议使用`finditer`进行惰性匹配:

Python

```python
it = re.finditer(r"(http|ftp)://", text)
# 或 it = re.finditer("(http|ftp):\\/\\/", text)

for match in it:
    print(match.span())
```

这种方法类似于 `pandas.read_csv(chunksize=n)`, 会返回一个 **iterator**, 逐次返回匹配结果

2 Replacing Patterns 模式替换

2.1 替换匹配项

使用`re.sub()`替换匹配的子字符串:

Python

```python
text = "Here's my number: 919-543-3300."
re.sub(r"\d", "Z", text)  # "Here's my number: ZZZ-ZZZ-ZZZZ."
```

2.2 分组与捕获

使用`()`进行分组, 可控制返回的匹配内容:

Python

```python
text = "At the site http://www.ibm.com. Some other text. ftp://ibm.com"
re.search(r"(http|ftp)://", text).group()  # 'http://'
re.search(r"(http|ftp)://", text).group(0)  # 'http://'
re.search(r"(http|ftp)://", text).group(1)  # 'http'
```

使用 `findall` 时, 若有 grouping operator, 则**只返回 "captured groups"**, 我们可以通过在最外层添加一个额外的 grouping operator 来捕获完整的 pattern:

Python

```python
re.findall(r"(http|ftp)://", text)  # ['http', 'ftp']
re.findall(r"((http|ftp)://)", text)  # [('http://', 'http'), ('ftp://', 'ftp')]
```

使用 `(?:...)` (non-capturing) 来匹配完整的 pattern 而不是 inner group:

Python

```python
re.findall(r"((?:http|ftp)://)", text)  # ['http://', 'ftp://']
```

2.3 对 captured group 进行替换

可在替换字符串中引用捕获组(如 `\1`):

Example

在数字前后添加下划线:

Python

```python
text = "Here's my number: 919-543-3300. They bought 731 bananas. Please call 919.554.3800."
re.sub("([0-9]+)", "_\\1_", text)
# "Here's my number: _919_-_543_-_3300_. They bought _731_ bananas. Please call _919_-._554_-._3800_."
```

Example

移除非字段分隔符的逗号:

Python

```python
text = '"H4NY07011","ACKERMAN, GARY L.","H","$13,242",,,'
re.sub(r'([^",]),', r'\1', text)
# '"H4NY07011","ACKERMAN GARY L.","H","$13242",,,'
```

Remark: 代码说明

- `[^",]` 匹配非引号, 非逗号的字符, `\1` 引用该字符, 替换时去掉逗号.
- 当 `^` 出现在 `[]` 内部时, 表示取反; 否则表示句首

可以使用 `\2`, `\3` 等来指代多个 captured groups

Python

```python
text = "Here's my number: 919-543-3300. They bought 731 bananas. Please call 919.554.3800."
re.sub("([0-9]{3})[-\.]([0-9]{3})[-\.]([0-9]{4})", "area code \\1, number \\2-\\3", text)
# "Here's my number: area code 919, number 543-3300. They bought 731 bananas. Please call area code 919, number 554-3800."
```

Remark

以下功能未详细展开, 但值得了解:

- **命名分组**: 可为分组命名, 便于引用.
- **回调函数替换**: `re.sub` 可接受一个函数作为替换逻辑.
- **模式内引用**: 使用 `\1` 等语法在模式中引用之前捕获的组.

3 Greedy Matching 贪婪匹配

3.1 贪婪匹配

默认情况下, 正则表达式会匹配尽可能多的字符, 称为"贪婪匹配":

Python

```python
text = "See the 998 balloons."
re.findall(r"\d+", text)  # ['998'] 而不是仅返回第一个 9
```

3.2 非贪婪匹配

在某些情况下, 贪婪匹配不符合预期, 例如去除HTML标签:

Python

```python
text = "Do an internship <b> in place </b> of <b> one </b> course."
re.sub(r"<.*>", "", text)  # 'Do an internship  course.'(过度匹配)
```

可以通过在 repetition specifier 之后使用 `?` 进行非贪婪匹配:

Python

```python
re.sub(r"<.*?>", "", text)  # 'Do an internship  in place  of  one  course.'
```

Remark

- `?` 在此处表示"尽可能少地匹配", 而非"0或1次"
- 也可改用更精确的表达式, 如:

Python

```python
re.sub(r"<[^>]*>", "", text)
```

4 Python中的特殊字符

Logic

在正则表达式中, 特殊字符需转义才能作为字面量使用. Python中反斜杠 `\` 也用于表示特殊字符(如 `\n`, `\t`), 因此需特别注意.

4.1 转义示例

Python

```python
# 匹配数字
re.search("\\d+", "a93b")  # <re.Match object; span=(1, 3), match='93'>
re.search(r"\d+", "a93b")  # <re.Match object; span=(1, 3), match='93'>

# 使用双反斜杠表示字面反斜杠
tmp = "something \\ other\n"
re.search("\\\\", tmp)  # <re.Match object; span=(10, 11), match='\\'>
```

4.2 使用原始字符串(raw string)

建议使用`r"..."`避免转义混乱:

Python

```python
print(r"My Windows path is: C:\Users\nadal.")  # 原样输出
re.search(r"\\", tmp)  # 匹配一个反斜杠
```

Remark

原始字符串不处理Python转义, 但正则表达式引擎仍会解析其中的转义符.