2 综述—— ELIZA 从何处开始?

本章内容概述

自然语言处理（NLP）通过建立形式化的数学模型，来分析、处理自然语言，并在计算机上用程序来实现分析和处理的过程，从而达到以机器来模拟人的部分乃至全部语言能力的目的[1]。NLP处理的“原材料”是自然语言，其产生具有无规律性，包括其目的、形式和种类等方面。著名的对话系统ELIZA 从何处开始？从数据开始。因此为了有效处理语言文本、更好地达到任务目的，往往需要对来自于世界不规则的语言文本进行**文本预处理**，而在本章中主要关注其中的**文本规范化**。文本规范化是将文本转换为标准、统一以及方便的形式，通过该步骤可以提高文本数据的质量和一致性，从而改善自然语言处理任务的性能。从一个语料库出发，我们可能需要首先对其进行**句子分割**，然后通过**分词**算法将句子划分为词（word），接着可能还需要进行**词规范化**如词形还原或词干提取等，最后将形成NLP后续处理的基本处理单元——词元（token）。值得注意的是，文本规范化具有明显的任务特异性，不同的NLP任务可能需要不同的规范化方法和步骤。例如，在情感分析任务中，可能需要保留情感词汇和标点符号，而在机器翻译任务中，可能需要更多地关注语法和句法的一致性。不同的任务对文本的要求不同，因此规范化过程也需要根据具体任务进行调整和优化。**正则表达式**是实现文本规范化的重要工具，其使用简单的字符串模式来描述、匹配文中全部匹配指定格式的字符串[2]。正则表达式在文本规范化的整个“生命过程”中无处不在，无论是在句子分割、分词还是词规范化中，都能发挥其强大的字符串处理功能。可以说，正则表达式是文本规范化的“基石”。本章的最后还对**最小编辑距离**进行了介绍，其提供了一种量化文本差异的方法，有助于在文本规范化过程中进行精确的修正和优化。

正则表达式

正则表达式是一种文本模式，包括普通字符（例如，a 到 z 之间的字母）和特殊字符（称为"元字符"），可以用来描述和匹配字符串的特定模式。作为一种用于模式匹配和搜索文本的工具。正则表达式提供了一种灵活且强大的方式来查找、替换、验证和提取文本数据，并在各种编程语言和文本处理工具中如 JavaScript、Python、Java、Perl 等有所应用。例如，下面的正则表达式可以正则表达式可以用于从字符串中提取电子邮件地址：

```
\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b
```

模式

正则表达式中的“模式”即规则，指用来描述和匹配文本中特定字符序列的规则。这些模式由一系列特殊字符和符号组成，可以表示字符类、重复次数、位置锚点等。总结起来包括以下：

- 字面值字符：例如字母、数字、空格等，可以直接匹配它们自身。
- 特殊字符：例如点号 `.`、星号 `*`、加号 `+`、问号 `?` 等，它们具有各自特殊的含义和功能。
- 字符集：用方括号 `[ ]` 包围的字符集合，用于匹配方括号内的任意一个字符。

- 元字符（转义字符）例如 `\d`、`\w`、`\s` 等，用于匹配特定类型的字符，如数字、字母、空白字符等。
- 量词：例如 `{n}`、`{n,}`、`{n,m}` 等，用于指定匹配的次数或范围。
- 边界符号：例如 `^`、`$`、`\b`、`\B` 等，用于匹配字符串的开头、结尾或单词边界位置。

通过组合这些模式，可以灵活地匹配和处理复杂的文本结构，从而使得文本搜索、替换和提取变得高效且精确。

下表对这些模式进行了简要介绍，注意双斜杠 ‘/ /’ 不属于模式本身。

类型	字符	描述
字面值字符	ab1	直接匹配它们自身。例如 ‘ab1’ 匹配字符串中连续出现的字符a、b和1。
量词	*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
	+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
	{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
	{n,}	n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
	{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。
	?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 。? 等价于 {0,1}。当该字符紧跟在任何一个其他限制符 (*, +, ?, {n}, {n,}, {n,m}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串 "oooo"，'o+?' 将匹配单个 "o"，而 'o+' 将匹配所有 'o'。
特殊字符	\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个 向后引用、或一个八进制转义符。例如，'n' 匹配字符 "n"。'\n' 匹配一个换行符。序列 '\\' 匹配 "\" 而 "\(" 则匹配 "("。
	^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
	$	匹配输入字符串的结束位置。如果设置了RegExp 对象的 Multiline 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
	.	匹配除换行符（\n、\r）之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用像"**(.|\n)**"的模式。

选择	x|y	匹配 x 或 y。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。
字符集	[xyz]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
	[^xyz]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'、'l'、'i'、'n'。
	[a-z]	字符范围。匹配指定范围内的任意字符。例如，'[a-z]' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符。
	[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，'[^a-z]' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符。
断言	(?:pattern)	匹配 pattern 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符 (|) 来组合一个模式的各个部分是很有用。例如，'industr(?:y|ies) 就是一个比 'industry|industries' 更简略的表达式。
	(?=pattern)	正向肯定预查（look ahead positive assert），在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，"Windows(?=95|98|NT|2000)"能匹配"Windows2000"中的"Windows"，但不能匹配"Windows3.1"中的"Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
	(?!pattern)	正向否定预查(negative assert)，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如"Windows(?!95|98|NT|2000)"能匹配"Windows3.1"中的"Windows"，但不能匹配"Windows2000"中的"Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
	(?<=pattern)	反向(look behind)肯定预查，与正向肯定预查类似，只是方向相反。例如，"`(?<=95|98|NT|2000)Windows`"能匹配"`2000Windows`"中的"`Windows`"，但不能匹配"`3.1Windows`"中的"`Windows`"。
	(?<!pattern)	反向否定预查，与正向否定预查类似，只是方向相反。例如"`(?<!95|98|NT|2000)Windows`"能匹配"`3.1Windows`"中的"`Windows`"，但不能匹配"`2000Windows`"中的"`Windows`"。
获取	(pattern)	匹配 pattern 并获取这一匹配。所获取的匹配可以从产生的 Matches 集合得到，在 VBScript 中使用 SubMatches 集合，在JScript 中则使用 $0…$9 属性。要匹配圆括号字符，请使用 '\(' 或 '\)'。
元字符（转义字符）	\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
	\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。

	\cx	匹配由 x 指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
	\d	匹配一个数字字符。等价于 [0-9]。
	\D	匹配一个非数字字符。等价于 [^0-9]。
	\f	匹配一个换页符。等价于 \x0c 和 \cL。
	\n	匹配一个换行符。等价于 \x0a 和 \cJ。
	\r	匹配一个回车符。等价于 \x0d 和 \cM。
	\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
	\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
	\t	匹配一个制表符。等价于 \x09 和 \cI。
	\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。
	\w	匹配字母、数字、下划线。等价于'[A-Za-z0-9_]'。
	\W	匹配非字母、数字、下划线。等价于 '[^A-Za-z0-9_]'。
	\xn	匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，'\x41' 匹配 "A"。'\x041' 则等价于 '\x04' & "1"。正则表达式中可以使用 ASCII 编码。
	\num	匹配 num，其中 num 是一个正整数。对所获取的匹配的引用。例如，'(.)\1' 匹配两个连续的相同字符。
	\n	标识一个八进制转义值或一个向后引用。如果 \n 之前至少 n 个获取的子表达式，则 n 为向后引用。否则，如果 n 为八进制数字 (0-7)，则 n 为一个八进制转义值。
	\nm	标识一个八进制转义值或一个向后引用。如果 \nm 之前至少有 nm 个获得子表达式，则 nm 为向后引用。如果 \nm 之前至少有 n 个获取，则 n 为一个后跟文字 m 的向后引用。如果前面的条件都不满足，若 n 和 m 均为八进制数字 (0-7)，则 \nm 将匹配八进制转义值 nm。
	\nml	如果 n 为八进制数字 (0-3)，且 m 和 l 均为八进制数字 (0-7)，则匹配八进制转义值 nml。
	\un	匹配 n，其中 n 是一个用四个十六进制数字表示的 Unicode 字符。例如， \u00A9 匹配版权符号 (?)。

优先级

正则表达式与算术表达式非常类似，采用从左到右进行计算的策略，并遵循规定的优先级顺序。下表由高到最低说明了各种正则表达式运算符的优先级顺序：

运算符	描述
\	转义符
(), (?:), (?=), []	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, \任何元字符、任何字符	定位点和序列（即：位置和顺序）
|	替换，"或"操作字符具有高于替换运算符的优先级，使得"m|food"匹配"m"或"food"。若要匹配"mood"或"food"，请使用括号创建子表达式，从而产生"(m|f)ood"。

修饰

修饰符是一种标记，其写在正则表达式外部，用于指定额外的匹配策略。修饰符的基本格式如下：

/pattern/flags

正则表达式中的常用修饰符如下表所示：

修饰符	含义	描述
i	ignore - 不区分大小写	将匹配设置为不区分大小写，搜索时不区分大小写: A 和 a 没有区别。
g	global - 全局匹配	查找所有的匹配项。
m	multi line - 多行匹配	使边界字符 ^ 和 $ 匹配每一行的开头和结尾，记住是多行，而不是整个字符串的开头和结尾。
s	特殊字符圆点 . 中包含换行符 \n	默认情况下的圆点 . 是匹配除换行符 \n 之外的任何字符，加上 s 修饰符之后, . 中包含换行符 \n。

分词

分词将连续的句子分割为词元（token），用于后续任务。从分词启动角度可一般地分为自上而下的分词算法和自下而上的分词算法。

- 自上而下的分词算法

自上而下的分词方法从整个句子或文本开始，逐步将其分解为较小的单位。这种方法通常基于语法规则或模式匹配，从全局视角出发，逐步细化分词结果。

示例：

假设有句子“我爱北京天安门”，自上而下的分词可能会先识别出“我爱北京”和“天安门”，然后进一步分解为“我”、“爱”、“北京”、“天安门”。

- 自下而上的分词算法

自下而上的分词方法从最小的单位（如字符或词素）开始，逐步组合成较大的单位。这种方法通常基于统计模型或词典匹配，从局部视角出发，逐步扩展分词结果。

示例：

同样是句子“我爱北京天安门”，自下而上的分词可能会先识别出“我”、“爱”、“北京”、“天安门”，然后逐步组合成“我爱北京”和“天安门”。

用一张表总结一下：

	视角	驱动方式	使用一般场景	典型算法
自上而下	全局视角：从整体出发，逐步细化。	规则驱动：常基于语法规则或模式匹配。	适用于结构化文本或有明确语法规则的文本。	正则表达式分词、语法分析分词、模板匹配分词、隐马尔可夫模型分词
自下而上	局部视角：从细节出发，逐步扩展。	统计驱动：常基于统计模型或词典匹配。	适用于非结构化文本或词汇丰富的文本。	双向匹配分词、N-gram分词、条件随机场分词、BiLSTM-CRF分词、BERT分词

词规范化

词规范化将词或词元转化为标准格式，以便于后续的文本处理和分析。词规范化可以减少词汇的多样性，提高文本数据的一致性和质量，有助于任务实现过程中的统一表达。常见的词规范化操作有大小写转换、词干提取（将词语还原为其基本形式或词根）、词形还原（将词语还原为其基本形式或词典形式）以及去除停用词（去除常见但对文本意义贡献较小的词语）等。在词规范化这一处理过程中，尤其要关注将要进行的任务。

最小编辑距离

编辑距离是计算机科学中的经典问题，即给定两个字符串a和b，编辑距离是将a转换为b所需的最小替换、插入和删除操作的数量[3]。最小编辑距离即所需上述操作的最小数量。

一般使用动态规划算法来计算最小编辑距离，其计算步骤如下：

1. 初始化一个二维表格 `dp` ，其中 `dp[i][j]` 表示将字符串 `s1` 的前 `i` 个字符转换为字符串 `s2` 的前 `j` 个字符所需的最少编辑操作次数。

2. 初始化表格的第一行和第一列，分别表示将空字符串转换为字符串 `s2` 的前 `j` 个字符和将字符串 `s1` 的前 `i` 个字符转换为空字符串所需的操作次数。
3. 填充表格，根据以下规则更新 `dp[i][j]`：
   - 如果 `s1[i-1] == s2[j-1]`，则 `dp[i][j] = dp[i-1][j-1]`。
   - 否则，`dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]) + 1`。
4. 最终的编辑距离为 `dp[m][n]`，其中 `m` 和 `n` 分别是字符串 `s1` 和 `s2` 的长度。

这里给出一个简单的 Python 实现：

```python
def min_edit_distance(s1, s2):
    # 获取两个字符串的长度
    m, n = len(s1), len(s2)

    # 初始化一个二维表格dp，其中dp[i][j]表示将s1的前i个字符转换为s2的前j个字符所需的最少编辑操作次数
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # 初始化表格的第一行，表示将空字符串转换为s2的前j个字符所需的操作次数
    for j in range(n + 1):
        dp[0][j] = j

    # 初始化表格的第一列，表示将s1的前i个字符转换为空字符串所需的操作次数
    for i in range(m + 1):
        dp[i][0] = i

    # 填充表格，根据以下规则更新dp[i][j]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            # 如果s1[i-1] == s2[j-1]，则不需要额外的编辑操作
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                # 否则，取插入、删除和替换操作中的最小值，并加1
                dp[i][j] = min(dp[i - 1][j],     # 删除操作
                               dp[i][j - 1],     # 插入操作
                               dp[i - 1][j - 1]) + 1  # 替换操作

    # 最终的编辑距离为dp[m][n]
    return dp[m][n]

# 示例
s1 = "kitten"
s2 = "sitting"
print(min_edit_distance(s1, s2))  # 输出: 3
```

```
35
```

总结

"巧妇难为无米之炊"，没有数据“原材料”这一原材料我们无法进行自然语言处理任务，也无法对计算语言学进行研究。更进一步的是，“好”的数据往往可以让我们在任务上的性能更好，因此需要对语料库中的原始数据进行文本预处理或文本规范化，这就涉及到了句子分割、分词、词规范化等具体技术。需要认识到的是，对数据的处理具有任务针对性，需要根据任务本身的特点以及需求采取不同的文本规范化标准和手段，才能在各个任务上表现出好的效果。

本章问题

参考文献

[1] 宋丽珏著,法律翻译新视野,重庆大学出版社,2016.06,第60页

[2] https://zh.wikipedia.org/wiki/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F

[3] https://arxiv.org/pdf/2204.09535v1

[4] https://web.stanford.edu/~jurafsky/slp3/2.pdf