NLP算法入门系列：中文切词算法，基于规则匹配

原创 IT可达鸭 IT可达鸭 5月4日

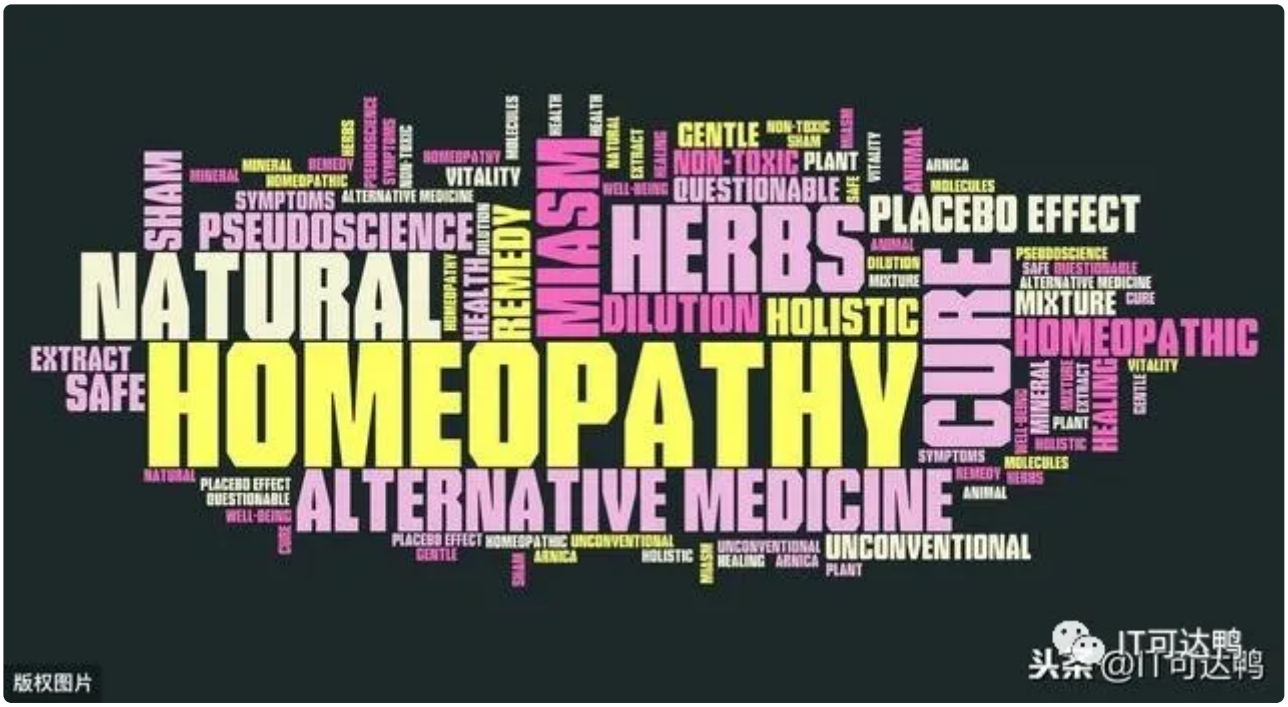
文/IT可达鸭

图/IT可达鸭、网络

- **前言**

英文分词，以空格进行分割就行了。但是，对于中文分词，它就是一个技术难点。因为对于人而言，不同知识背景的人，有时候看待同一个句子，它的分词完全是不一样的。

在进行自然语言理解的第一步，就是将词语确定下来。如果能达到像英文分词那样简单，后续的短语划分、概念抽取、主题分析以及语言理解那就顺理成章了。**因此每个NLP算法工程师的第一个最先掌握的基础算法就是分词技术。**



- **三类分词算法**

1 规则分词

基于规则的分词，它是一种机械分词方法。主要是通过定期维护一个词典（定时记录新词、删除旧的词汇等），在对句子进行切分时，利用句子的每个子串与词典中的词进行逐一匹配切分，未匹配就作为单字切分。

优点：简单高效；

缺点：对新词很难处理。

2 统计分词

根据统计学、机器学习技术，利用提前准备好的文章语料，进行统计分析。分词的好坏依赖机器学习算法的参数、语料的大小和质量优劣。

优点：

1.能够较好的应对新词的发现、

2.不同领域的文章分词算法，可以通过不同训练不同语料库获取的模型进行分词

缺点：太过于依赖语料的质量

3 规则分词+统计分词

结合规则分词和统计分词的优劣，实践中多数是采用以上两种方法的结合，或是不同场景不同业务下采用不同的方法。即混合分词。

规则切词详解

按照规则切词的方式，主要有**正向最大匹配算法**、**逆向最大匹配算法**以及**双向最大匹配算法**。

- **基于规则的三种算法: 正向最大匹配算法**

算法描述：

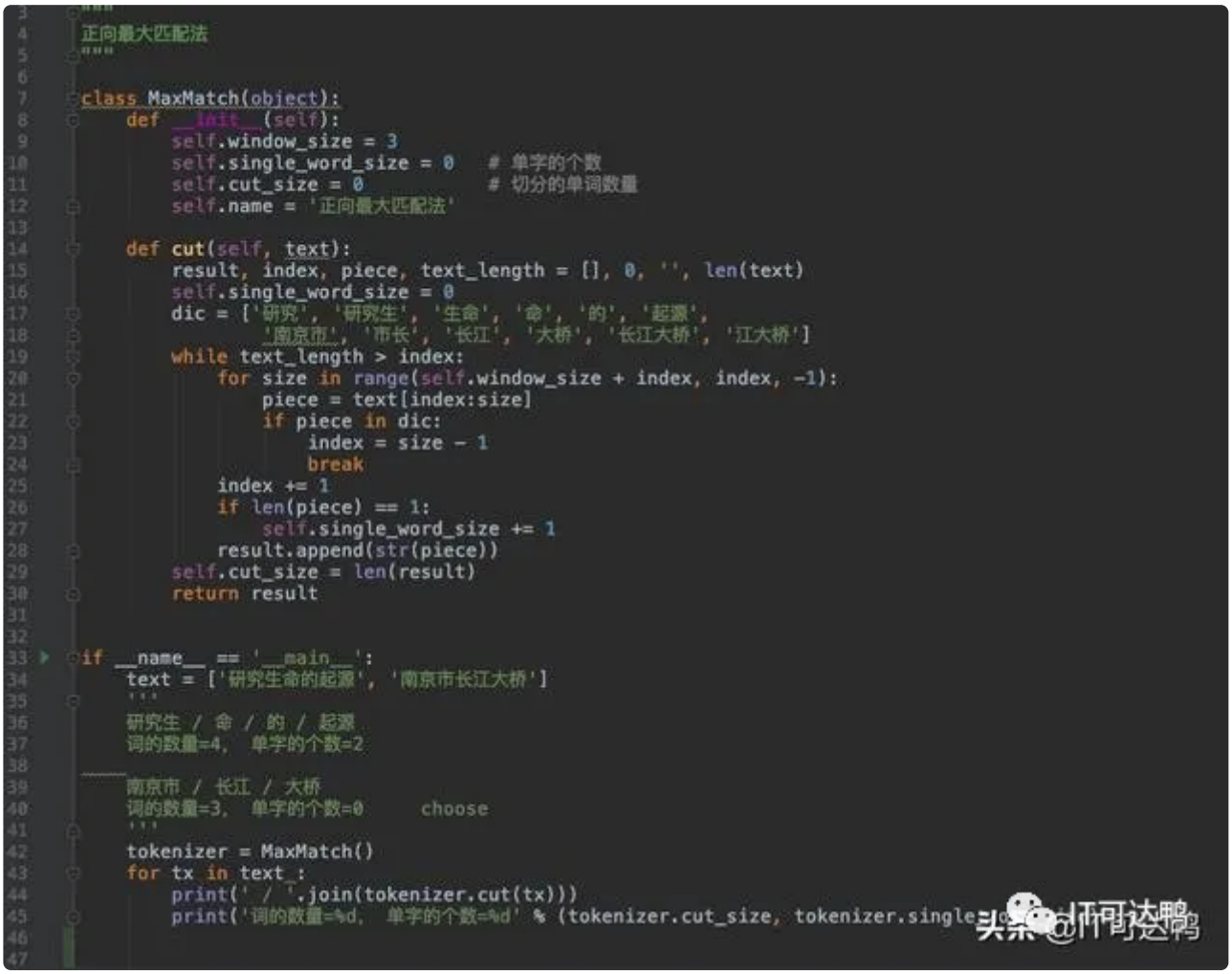
1）从左向右可重叠地取语句的m个字符作为匹配字符子串，其中，m为机器词典中最长词语的字符数；

2）当原句中m个字符的子串与词典的所有词进行匹配，若匹配成功，则将这个匹配字符串作为一个词语；

3）若匹配不成功，则将m个字符的最后一个字符去掉，用m-1个字符作为新的匹配字段。即

m=m-1 （ m>1 ）， 重复 1 ~ 3步骤，直到切分出所有的词为止。

代码截图：



- **基于规则的三种算法: 逆向最大匹配算法**

算法描述：

1 ）从右到左可重叠地取语句的m个字符作为匹配字符子串，其中，m为机器词典中最长词语的字符数；

2 ）当原句中m个字符的子串与词典的所有词进行匹配，若匹配成功，则将这个匹配字符串作为一个词语；

3 ）若匹配不成功，则将m个字符的最后一个字符去掉，用m-1个字符作为新的匹配字段。即

m=m-1 （ m>1 ）， 重复 1 ~ 3步骤，直到切分出所有的词为止。

代码截图：

```
"""
逆向最大匹配法
"""

class ReverseMaxMatch(object):
    def __init__(self):
        self.window_size = 3
        self.single_word_size = 0  # 单字的个数
        self.cut_size = 0  # 切分的单词数量
        self.name = '逆向最大匹配法'

    def cut(self, text):
        result, index, piece = [], len(text), ''
        self.single_word_size = 0
        dic = ['研究', '研究生', '生命', '命', '的', '起源',
               '南京市', '市长', '长江', '大桥', '长江大桥', '江大桥']
        while index > 0:
            for size in range(index - self.window_size, index):
                piece = text[size:index]
                if piece in dic:
                    index = size + 1
                    break
            index -= 1
            if len(piece) == 1:
                self.single_word_size += 1
            result.append(str(piece))
        result.reverse() # 结果翻转一下
        self.cut_size = len(result)
        return result

if __name__ == '__main__':
    text = ['研究生命的起源','南京市长江大桥']
    '''
    研究 / 生命 / 的 / 起源
    词的数量=4,  单字的个数=1      choose

    南 / 京 / 市长 / 江大桥
    词的数量=4,  单字的个数=2
    '''
    tokenizer = ReverseMaxMatch()
    for tx in text:
        print(' / '.join(tokenizer.cut(tx)))
        print('词的数量=%d,  单字的个数=%d' % (tokenizer.cut_size, tokenizer.single_word_
```

- **基于规则的三种算法: 双向最大匹配算法**

算法描述：

1 ） 结合正向最大匹配算法和逆向最大匹配算法；

2 ）如果正向逆向分词结果的词语数量不同，则取分词数量较少的结果；

3 ）如果分词结果的词语数量相同，但是分词结果不同，就返回分词结果中单字较少的结果。否则就返回逆向最大匹配算法的分词结果（据实验数据统计，逆向最大匹配算法的分词结果准确的概率比正向最大匹配算法分词结果准确的概率要高。）

代码截图：

```
"""
双向最大匹配
"""
from 中文分词技术.algorithm.MaxMatch import MaxMatch
from 中文分词技术.algorithm.ReverseMaxMatch import ReverseMaxMatch

class DoubleDirMatch(object):
    def __init__(self):
        # 正向最大匹配和逆向最大匹配
        self.max_match = MaxMatch()
        self.reverse_max_match = ReverseMaxMatch()
        self.name = '双向最大匹配'
        self.choose_match = ''

    def cut(self, text):
        rnt_max_match = self.max_match.cut(text)
        rnt_reverse_max_match = self.reverse_max_match.cut(text)

        if self.max_match.cut_size != self.reverse_max_match.cut_size:
            # 如果正反向分词结果，词数不同， 返回次数较少的分词结果
            if self.max_match.cut_size < self.reverse_max_match.cut_size:
                self.choose_match = self.max_match.name
                return rnt_max_match
            else:
                self.choose_match = self.reverse_max_match.name
                return rnt_reverse_max_match
        else:
            # 如果正反向分词结果，词数相同，则返回单字较少的一个
            if self.max_match.single_word_size < self.reverse_max_match.single_word_size:
                self.choose_match = self.max_match.name
                return rnt_max_match
            else:
                # 若单字数也相同，就返回最大逆向匹配（实验表明，最大逆向结果准确性比较高）
                self.choose_match = self.reverse_max_match.name
                return rnt_reverse_max_match

if __name__ == '__main__':
    text = ['研究生命的起源', '南京市长江大桥']
    tokenizer = DoubleDirMatch()
    for tx in text:
        print(' / '.join(tokenizer.cut(tx)))
        print('选择的分词算法: %s' % tokenizer.choose_match)
```

- **结语**

基于规则的分词，一般都较为简单和高效，但是词典的维护是一个非常巨大的工程。在网络发达的今天，网络新词层出不穷，很难通过词典覆盖到所有词。

但是，幸运的是，在一些指定的业务场景或者领域中，词典很容易被穷尽，而且存储不大。例如：法律领域的文档分词、医学领域的文档分词，它的所有词是固定的，一般很少变动，不像自媒体，经常有新的网络词汇出现。所以在这种业务场景下，使用规则分词，是最好的选择，只需要不同业务场景，维护不同的词典即可。

由于小编技术有限，文中难免有不对的地方，欢迎大家指正。如果大家对规则分词有其他的想法，欢迎在下方评论讨论。

如果有疑问想获取源码，可以关注后，在后台私信我，回复：python规则分词。我把源码发你。持续关注"IT可达鸭"，每天除了分享有趣Python源码，还会介绍NLP算法。最后，感谢大家的阅读，祝大家工作生活愉快！