自然语言处理
第三次编程作业

西安交通大学, 数学与统计学院, 强基数学 002

马煜璇 [a], 吴天阳 [b]

2204220461[a], 2204210460[b]

2022 年 12 月 3 日

目录

1 实验目的

利用最大匹配算法对汉语进行分词操作，并对结果进行分析。我们分别实现下面三种算法：

1. 正向最大匹配算法 (FMM)
2. 反向最大匹配算法 (BMM)
3. 双向最大匹配算法

2 实验原理

2.1 正向最大匹配算法

正向最大匹配算法 (FMM) 匹配的基本思想是：假设词典中最大词条所含的汉字个数为 n 个，取待处理字符串的前 n 个字作为匹配字段，查找分词词典。若词典中含有该词，则匹配成功，分出该词，然后从被比较字符串的 $n+1$ 处开始再取 n 个字组成的字段重新在词典中匹配；如果没有匹配成功，则将这 n 个字组成的字段的最后一位剔除，用剩下的 $n-1$ 个字组成的字段在词典中进行匹配，如此进行下去，直到切分成功为止。

例如，待处理字符串为“汉字多为表意文字”，取字符串“汉语多为表”与词典进行比较，没有与之对应的词，去除“表”字，用字段“汉语多为”进行匹配，直至匹配到“汉语”为至，再取字符串“多为表意”，循环到切分出“文字”一词。

目前，正向最大匹配方法作为一种基本的方法已被肯定下来，但是由于错误比较大，一般不单独使用。如字符串“处理机器发生的故障”，在正向最大匹配方法中会出现歧义切分，该字符串被分为：处理机、发生、故障，但是使用逆向匹配就能得到有效的切分。

2.2 逆向最大匹配算法

逆向最大匹配算法 (RMM) 的分词原理和过程与正向最大匹配相似，区别在于前者从文章或者句子 (字串) 的末尾开始切分，若不成功则减去最前面的一个字。

比如对于字符串“处理机器发生的故障”，第一步，从字串的右边取长度以步长为单位的字段“发生的故障”在词典中进行匹配，匹配不成功，再取字段“生的故障”进行匹配，依次匹配，直到分出“故障”一词，最终使用 RMM 方法切分的结果为：故障、发生、机器、处理。该方法要求配备逆序词典。

一般来说根据汉语词汇构成的特点，从理论上说明了逆向匹配的精确度高于正向匹配，汉语语句的特点一般中心语偏后。有研究数据, 单纯使用正向最大匹配的错误率为 1/169, 单纯使用逆向最大匹配的错误率为 1/245。

2.3 双向最大匹配算法

FMM 和 BMM 两种算法都分词一遍，然后根据大颗粒度词越多越好，非词典词和单字词越少越好的原则，选取其中一种分词结果输出。

3 实验步骤与结果分析

3.1 读入词库

我们从https://github.com/fxsjy/jieba/blob/master/extra_dict/dict.txt.big下载了词库，并读取词库，词库大小为 583280。

```
# 1.词库读取
dic = set()
with open('dict.txt_2.big', encoding='utf-8') as file:
    for s in file.readlines():
        word = s.split()[0]
        dic.add(word)
print('词库大小', len(dic))
```

同时，我们查找了词库中最长的词语与最长词的长度，长度为 16.

```
#词库中的最长词
counter = collections.Counter()
for w in dic:
    counter[w] = len(w)
print(counter.most_common()[:10])
```

词库中最长的词语与长度为

```
[('第九届全国人民代表大会常务委员会', 16), ('侵華日軍南京大屠殺遇難同胞紀念館', 16),
    ('侵华日军南京大屠杀遇难同胞纪念馆', 16), ('第九屆全國人民代表大會常務委員會',
    16), ('劳动和社会保障部职业技能鉴定中心', 16),
    ('八千一百三十七萬七千二百三十六口', 16), ('外交部駐澳門特別行政區特派員公署',
    16), ('外交部駐香港特別行政區特派員公署', 16),
    ('外交部驻香港特别行政区特派员公署', 16), ('八千一百三十七万七千二百三十六口',
    16)]
```

计算文本长度

```
#原文本长度
```

```
length = len(text) # 获取文本长度
print('原文本长度', length, '\n')
```

判断是否为词语

```
def check(word): # 判断是否为词语
return word in dic
```

3.2 正向最大匹配算法

正向最大匹配算法从正向取词，即从左往右取词，取词最大长度为词典中最长词的长度16，每次右边减一个字，直到词典中存在或剩下 1 个单字。

```
pos_div= []
pos_single = 0
i = 0 # 左指针
while i < length:
    for j in range(min(i+16, length), i, -1): # 右指针
        if check(text[i:j]):
            pos_div.append(text[i:j])
            break
    if i + 1 == j: # 记录单个字词个数
        pos_single += 1
    i = j
print('正向最大匹配法长度: ', len(pos_div))
print('单个字个数: ', pos_single)
```

3.3 逆向最大匹配算法

反向即从右往左取词，其他逻辑和正向相同。

```
neg_div= []
neg_single = 0
j = length # 右指针
while j > 0:
    for i in range(max(j-16, 0), j): # 左指针
        if check(text[i:j]):
            neg_div.append(text[i:j])
            break
```

```
    if i + 1 == j: # 记录单个字词个数
        neg_single += 1
    j = i
neg_div = neg_div[::-1]
print('逆向最大匹配法长度: ', len(neg_div))
print('单个字个数: ', neg_single,'\n')
```

3.4 双向最大匹配算法

FMM 和 BMM 两种算法都分词一遍，选取其中一种分词结果输出。选择标准：

1. 首先看两种方法结果的分词数，分词数越少越好；
2. 分词数相同的情况下，看单个词的数量，越少越好.

```
best_div = None
info = [len(pos_div)-len(neg_div), pos_single-neg_single]
# 优先选取分词个数少的，再选取单个词少的
if info[0] < 0 or (info[0] == 0 and info[1] < 0):
    best_div = pos_div
    print('选择正向')
elif info[0] == info[1] == 0: # 两个数目都相同
    best_div = pos_div
    print('两者相同')
else:
    print('选择逆向')
    best_div = neg_div
print(best_div)
```

4 结论与讨论

我们对朱自清先生的文章《荷塘月色》片段进行分词，文本长度为 787，分别利用正向最大匹配算法和逆向最大匹配算法对其进行分词，得到分词结果如下：

```
正向最大匹配法长度： 448
单个字个数： 355
逆向最大匹配法长度： 447
单个字个数： 354
```

根据双向最大匹配算法的选择标准，选择逆向，原文分词结果为

[' 路上', ' 只', ' 我', ' 一', ' 个人', ' 背着手', ' 踱', ' 着', ' 这', ' 一片', ' 天地', ' 好', ' 像是', ' 我', ' 的', ' 我', ' 也', ' 像', ' 超出', ' 了', ' 平常', ' 旳', ' 自己', ' 到', ' 了', ' 另', ' 一', ' 世界', ' 里', ' 我', ' 爱', ' 热闹', ' 也', ' 爱', ' 冷静', ' 爱', ' 群居', ' 也', ' 爱', ' 独处', ' 像', ' 今', ' 晚上', ' 一', ' 个人', ' 在', ' 这', ' 苍茫', ' 旳', ' 月', ' 下', ' 什么', ' 都', ' 可以', ' 想', ' 什么', ' 都', ' 可以', ' 不想', ' 便', ' 觉', ' 是', ' 个', ' 自由', ' 的', ' 人', ' 白天', ' 里', ' 一', ' 定要', ' 做', ' 的', ' 事', ' 一定', ' 要说', ' 的话', ' 现在', ' 都', ' 可', ' 不理', ' 这', ' 是', ' 独处', ' 的', ' 妙处', ' 我', ' 且', ' 受用', ' 这', ' 无边', ' 的', ' 荷香', ' 月色', ' 好', ' 了', ' 曲曲折折', ' 的', ' 荷塘', ' 上面', ' 弥望', ' 旳', ' 是', ' 田', ' 田', ' 的', ' 叶子', ' 叶子', ' 出水', ' 很', ' 高', ' 像', ' 亭亭', ' 旳', ' 舞女', ' 旳', ' 裙', ' 层层', ' 的', ' 叶子', ' 中间', ' 零星', ' 地', ' 点缀着', ' 些', ' 白花', ' 有', ' 袅娜', ' 地', ' 开', ' 着', ' 旳', ' 有', ' 羞涩', ' 地', ' 打着', ' 朵儿', ' 旳', ' 正', ' 如一', ' 粒粒', ' 的', ' 明珠', ' 又', ' 如', ' 碧', ' 天', ' 里', ' 的', ' 星星', ' 又', ' 如', ' 刚', ' 出浴', ' 的', ' 美', ' 人', ' 微风', ' 过', ' 处', ' 送来', ' 缕缕', ' 清香', ' 仿佛', ' 远处', ' 高', ' 楼上', ' 渺茫', ' 的', ' 歌声', ' 似的', ' 这时候', ' 叶子', ' 与', ' 花', ' 也', ' 有', ' 一丝', ' 的', ' 颤动', ' 像', ' 闪电般', ' 霎时', ' 传过', ' 荷塘', ' 的', ' 那边', ' 去', ' 了', ' 叶子', ' 本', ' 是', ' 肩并肩', ' 密密', ' 地', ' 挨着', ' 这', ' 便', ' 宛然', ' 有', ' 了', ' 一道', ' 凝', ' 碧', ' 的', ' 波痕', ' 叶子', ' 底下', ' 是', ' 脉脉', ' 的', ' 流水', ' 遮住', ' 了', ' 不', ' 能见', ' 一些', ' 颜色', ' 而', ' 叶子', ' 却', ' 更', ' 见', ' 风致', ' 了', ' 月光', ' 如', ' 流水', ' 一般', ' 静静地', ' 泻', ' 在', ' 这', ' 一片', ' 叶子', ' 和', ' 花', ' 上', ' 薄薄的', ' 青', ' 雾', ' 浮起', ' 在', ' 荷塘', ' 里', ' 叶子', ' 和', ' 花', ' 仿佛', ' 在', ' 牛乳', ' 中', ' 洗过', ' 一样', ' 又', ' 像', ' 笼', ' 着', ' 轻纱', ' 的', ' 梦', ' 虽然', ' 是', ' 满月', ' 天上', ' 却', ' 有', ' 一层', ' 淡淡的', ' 云', ' 所以', ' 不能', ' 朗', ' 照', ' 但', ' 我', ' 以为', ' 这', ' 恰', ' 是', ' 到', ' 了', ' 好处', ' 酣眠', ' 固', ' 不可', ' 少', ' 小睡', ' 也', ' 别有风味', ' 的', ' 月', ' 光是', ' 隔', ' 了', ' 树', ' 照过来', ' 的', ' 高处', ' 丛生', ' 的', ' 灌木', ' 落下', ' 参差', ' 的', ' 斑驳', ' 的', ' 黑影', ' 峭', ' 楞楞', ' 如', ' 鬼', ' 一般', ' 弯弯的', ' 杨柳', ' 的', ' 稀疏', ' 的', ' 倩影', ' 却', ' 又', ' 像是', ' 画', ' 在', ' 荷叶', ' 上', ' 塘', ' 中', ' 的', ' 月色', ' 并', ' 不', ' 均匀', ' 但', ' 光与影', ' 有着', ' 和谐', ' 的', ' 旋律', ' 如', ' 梵', ' 婀', ' 玲', ' 英语', ' 小提琴', ' 的', ' 译音', ' 上奏', ' 着', ' 的', ' 名曲', ' 荷塘', ' 的', ' 四面', ' 远远近近', ' 高高低低', ' 都', ' 是', ' 树', ' 而', ' 杨柳', ' 最多', ' 这些', ' 树', ' 将', ' 一片', ' 荷塘', ' 重重围住', ' 只', ' 在', ' 小路', ' 一旁', ' 漏', ' 着', ' 几段', ' 空隙', ' 像是', ' 特为', ' 月光', ' 留下', ' 的', ' 树', ' 色', ' 一例', ' 是', ' 阴阴的', ' 乍看', ' 像', ' 一团', ' 烟雾', ' 但', ' 杨柳', ' 的', ' 丰姿', ' 便', ' 在', ' 烟', ' 雾里', ' 也', ' 辨', ' 得出', ' 树梢', ' 上', ' 隐隐约约', ' 的', ' 是', ' 一带', ' 远山', ' 只', ' 有些', ' 大意', ' 罢了', ' 树', ' 缝里', ' 也', ' 漏', ' 着', ' 一两点', ' 路', ' 灯光', ' 没精打采', ' 的', ' 是', ' 渴睡', ' 人', ' 的', ' 眼', ' 这时候', ' 最', ' 热闹', ' 的', ' 要数', ' 树上', ' 的', ' 蝉声', ' 与', ' 水里', ' 的', ' 蛙声', ' 但', ' 热闹', ' 是', ' 它们', ' 的', ' 我', ' 什么', ' 也', ' 没有']

附录

```
import collections

def solve(text):
# 1.词库读取
dic = set()
with open('dict.txt_2.big', encoding='utf-8') as file:
for s in file.readlines():
word = s.split()[0]
dic.add(word)
print('词库大小', len(dic))

#词库中的最长词
counter = collections.Counter()
for w in dic:
counter[w] = len(w)
print(counter.most_common()[:10])

#原文本长度
length = len(text) # 获取文本长度
print('原文本长度', length, '\n')

def check(word): # 判断是否为词语
return word in dic

# 2.正向最大匹配法
pos_div= []
pos_single = 0
i = 0 # 左指针
while i < length:
for j in range(min(i+16, length), i, -1): # 右指针
if check(text[i:j]):
pos_div.append(text[i:j])
break
if i + 1 == j: # 记录单个字词个数
pos_single += 1
i = j
```

```
print('正向最大匹配法长度: ', len(pos_div))
print('单个字个数: ', pos_single)

# 3.逆向最大匹配法
neg_div= []
neg_single = 0
j = length # 右指针
while j > 0:
for i in range(max(j-16, 0), j): # 左指针
if check(text[i:j]):
neg_div.append(text[i:j])
break
if i + 1 == j: # 记录单个字词个数
neg_single += 1
j = i
neg_div = neg_div[::-1]
print('逆向最大匹配法长度: ', len(neg_div))
print('单个字个数: ', neg_single,'\n')

# 4.双向最大匹配法
best_div = None
info = [len(pos_div)-len(neg_div), pos_single-neg_single]
# 优先选取分词个数少的，再选取单个词少的
if info[0] < 0 or (info[0] == 0 and info[1] < 0):
best_div = pos_div
print('选择正向')
elif info[0] == info[1] == 0: # 两个数目都相同
best_div = pos_div
print('两者相同')
else:
print('选择逆向')
best_div = neg_div
print(best_div)

if __name__ == '__main__':
# with open('text.txt', encoding='utf-8') as file:
#     long_text = file.read()
text = '路上只我一个人，背着手踱着。这一片天地好像是我的...' # 设定目标文本
```

```
75 solve(text)
```