

华东师范大学数据学院上机实践报告

课程名称：信息检索

年级：2018

上机实践成绩：

指导教师：张蓉

姓名：孙秋实

上机实践名称：汉语分词：最大匹配方法

学号：10185501402

上机实践日期：2021/9/16

Part 1

实验目的

- (1) 编写程序，完成基于最大匹配方法的汉语分词
- (2) 评价分词结果

Part 2

实验任务

- (1) 完成基于最大匹配方法的汉语分词程序编写。
- (2) 使用 Precision/Recall/F1-Score 来评价分词结果
- (3) 案例分析与可视化

Part 3

使用环境

- (1) Google Colab
- (2) Python 3.7

Part 4

实验过程

Section 1

最大匹配方法简介

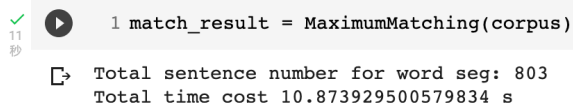
中文分词的目的是将汉字序列切分为词序列，而最大匹配算法则是一种基于词典的贪心算法，我们拥有一个词典，其中有所有可能的词语，即除了单字，分词结果中的每个词均要在词典中出现。从句首开始，向右截取最大长度，组成当前词和字典中的词逐一进行匹配。若匹配成功，则移向下一位置，若失败，则缩短 1 个字的长度，重复该匹配动作，直到匹配成功或是只剩单字。

Section 2

中文分词：最大匹配算法的实现

实现最大分词算法，其思想比较简单，已在简介中阐述，算法的核心函数如下
(其他如读写文件等次要代码在报告中已省略，详情请见 Jupyter 记事本)

```
def MaximumMatching(corpus):
    time_start=time.time()
    match_answer = []
    sentence_num = 0
    for line in corpus:
        per_sentence_answer = []
        pos = 0
        while pos<len(line): # 终止条件为pos到了当前这个句子的最大长度
            current_line_len = len(line)
            start_len = min(max_word_len, current_line_len-pos-1) # start_len指初始状态的最长匹配长度
            # 此处需要确定最大单词长，可能是10，也可能是当前匹配后剩下的部分
            for current_len in range(start_len,0,-1): # 从最大长度开始减，每次失败就-1
                if (line[pos:pos+current_len] in dict_array):
                    per_sentence_answer.append(line[pos:pos+current_len]) #
                    # 往单句匹配结果加入当前匹配好的词
                    pos=pos+current_len-1
                    # position移动到匹配完单词后的下一个位置
                    break;
                elif current_len == 1: # 只有单字可匹配则直接加入匹配结果中
                    per_sentence_answer.append(line[pos:pos+current_len])
            pos+=1
        # 将当前句子的分词结果以列表形式append到答案中
        match_answer.append(per_sentence_answer)
        sentence_num+=1
    time_end=time.time()
    print("Total sentence number for word seg:",sentence_num)
    print('Total time cost',time_end-time_start,'s')
    return match_answer
```



```
11 1 match_result = MaximumMatching(corpus)
Total sentence number for word seg: 803
Total time cost 10.873929500579834 s
```

图 1: 算法执行

如图 1 所示总共为 803 个句子进行分词，时间开销 10.87 秒。

分词结果存储在 corpus.out.txt，如图 2 所示

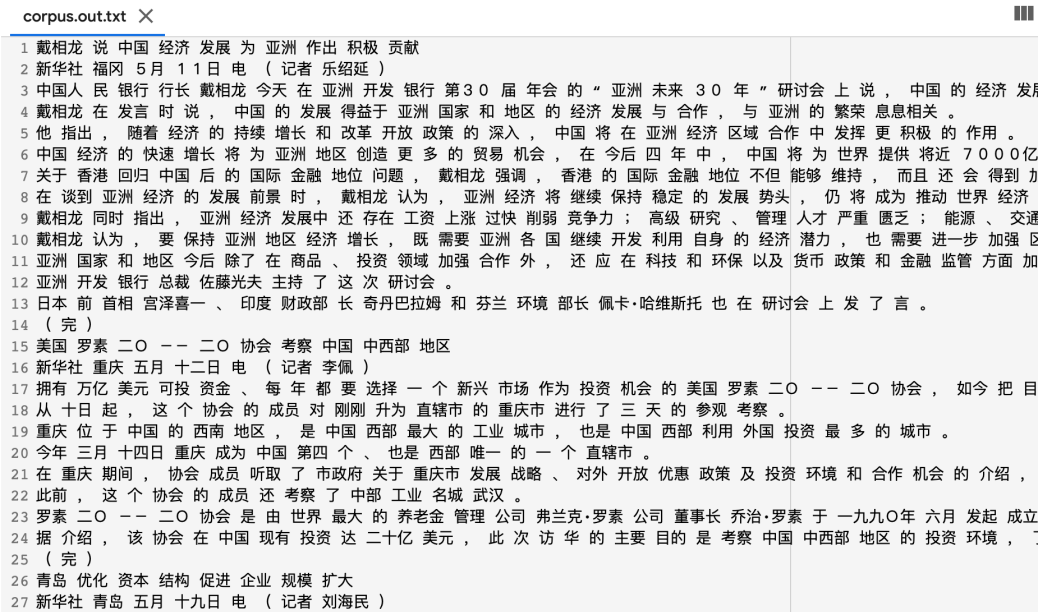


图 2: corpus.out.txt

Section 3

评价分词结果

将获得的分词结果保存为 corpus.out.txt 后，与 corpus.answer.txt 进行匹配，衡量该分词任务的效果。
该部分的核心函数如下

```
def check_performance(match_result, correct_match):
    OutputFile = open("performance.txt", "w", encoding="utf8")
    OutputFile.write("Index\t Precision\t Recall\t F1-score\n")
    # 写Header
    compare_len = len(correct_match)
    check_index = 0
    for i in range(compare_len):
        check_index += 1 # 句子序号
        temp_len = len(match_result[i]) # 当前校验的句子的分词结果的长度
        valid_tokens = correct_match[i].split() # 正确的分词结果
        valid_len = len(valid_tokens) # 正确的分词结果的长度
        correct_sum = 0 # 维护一个变量记录多少个正确
        answer_pos = valid_pos = 0 # 位置指针
        seg_len = val_len = 0 # 两个变量来表示长度位置
        # 开始和答案配对
        current_corpus_len = len(corpus[i]) - 1 # -1是因为下标从0开始
        while(seg_len < current_corpus_len and val_len < current_corpus_len):
            # 被校验的分词结果和正确结果任意一个达到了最大长度则终止
            # 以下匹配答案，需要注意长度不一，分为三种情况
            if seg_len == val_len:
                if (match_result[i][answer_pos] == valid_tokens[valid_pos]): correct_sum += 1
```

```

seg_len += len(match_result[i][answer_pos]) # 第i个句的当前位置
val_len += len(valid_tokens[valid_pos])
answer_pos += 1
valid_pos += 1
elif seg_len > val_len:
    val_len += len(valid_tokens[valid_pos]) # 当前tokens前移一位
    valid_pos += 1
else: # 小
    seg_len += len(match_result[i][answer_pos]) # 当前的position前移一位
    answer_pos += 1
# 计算三个指标 Precision Recall 和 F1-Score
Precision = correct_sum/temp_len
Recall = correct_sum/valid_len
F1_score = cal_F1(Precision,Recall) #函数如上
OutputFile.write("%d\t %.2f\t %.2f\t %.2f\n" % (check_index, Precision, Recall, F1_score))

```

该部分输出的示例如表 1 所示

表 1: 分词结果评价 (前十句)

Index	Precision	Recall	F1-score
1	1.0	1.0	1.0
2	1.0	1.0	1.0
3	0.95	0.95	0.95
4	1.0	1.0	1.0
5	1.0	1.0	1.0
6	1.0	1.0	1.0
7	1.0	1.0	1.0
8	1.0	1.0	1.0
9	0.98	0.96	0.97
10	1.0	1.0	1.0

最后将分词结果的 Precision/Recall/F1-Score 单独输出到文件中，将输出结果转为 csv 格式后，借助 Pandas 可以查看总体的分词效果。

```

print("average Precision",np.mean(df[' Precision']))
print("average Recall",np.mean(df[' Recall']))
print("average F1-Score",np.mean(df[' F1-score']))

# average Precision 0.9947198007471995
# average Recall 0.9923412204234126
# average F1-Score 0.99348692403487

```

总体评价：

- Precision:0.9947
- Recall:0.9923

- F1-Score:0.9934

Section 4

案例分析

可视化 803 条句子的 Precision/Recall/F1-Score，可以看到存在一个极端值，如图 3 所示

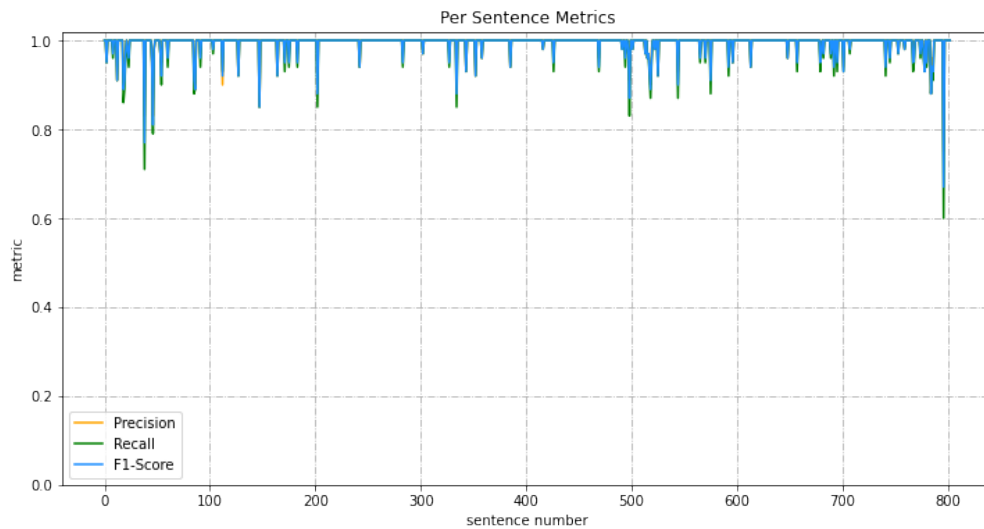


图 3: Word Segmentation Evaluation

分析得知在这个数据集内，表现最差的为第 797 句，单独对其进行分析。第 797 行文本为“【水蒸蛋糕】”，最大匹配分词结果为【水蒸 蛋糕】，答案为【水 蒸 蛋糕】，水蒸这个词在词典里，所以被匹配了，整个句子比较短，所以导致了准确率和召回率都出现了明显的下降，但并不影响该算法在整体文本上的有效性。