

汉语分词系统

120L022013 王炜栋
哈尔滨工业大学
120L022013@stu.hit.edu.cn

摘要

本次实验的主要目的是对汉语分词技术有一个全面的了解。本实验的具体流程可分为 3.1: 词典划分 3.2 基于机械匹配的 FMM, BMM 算法 3.3: 评估算法 3.4: 速度优化 3.5 基于统计的分词 (N 元文法) 3.6: 基于隐马尔可夫模型实现未登录词识别的分词系统。在完成实验后, 我们得出的各性能 (以 F1 为评测标准) 基本为 FMM: 0.9657、BMM:0.9675、Uni_LM:0.9881、Bi_LM:0.9794, 当进行未登录词识别 (样本划分 50% 分别作为训练集和测试集) 时, Uni_LM:0.9519, Bi_LM:0.9444

1 引言

作为自然语言处理的基础与初识, 汉语分词是重要且关键的一课。词汇作为自然语言中能够独立表达含义的最小单位, 我们如何解读词汇将决定我们表达的含义。但是在生活常见的场景中, 汉语词汇并不会像英语词汇一样在句子中自然切分, 因此为了解读汉语句子的真正含义, 我们需要使用庞大的训练集来训练模型达到更高的切分词准确率。进而更好的解读表达的含义。为自然语言处理打下良好基础。

然而中文分词面临着许多的难点, 具体表现为: **1) 歧义切分:** 同一词汇在不同情境下可能由于表达不同的意思进行不同的切分, 例如某宾馆门前牌匾如是写道“下雨天留客天留客不留”可以有不同切分: “下雨天|留客天|留客不|留” (宾馆留客) 或 “下雨|天留客|天留|客不留” (宾馆不留客) **2) 未登录词:** 训练集不包含待切分词汇, 则切分效果较差。**3) 命名识别:** 例如小方高兴可以理解为 1 小方|高兴 2 小|方高兴 (即人名是小方还是方高兴) **4) 新兴词汇的更迭:** 诸如芭比 Q 了、老六、yyds 等, 我们难以随时将新产生的词汇收纳

入数据集并进行维护---等等诸如此类难点都较难以优化, 需要通过更高阶的文法联系上下文甚至全文进行识别。这也对自动分词技术提出了艰巨的任务。

2 实验相关介绍

当前主流的分词算法主要可分为三类: 基于机械匹配分词算法、基于经验统计分词算法、基于深度学习分词算法。其中本实验主要应用了前两种方法, 分别体现于 3.2-3.4 和 3.5-3.6, 而如今较为权威的分词系统例如 jieba 分词¹、THULAC²、SnowNLP³主要使用的是后两种方法、且有较多的附加功能。

基于机械匹配的分词算法, 在本实验中要求为 FMM、BMM, 基本思想为在数据集中搜集切分词汇, 并以此构造词典。当再次切分某个待分词句子时通过匹配是否有与之对应词汇进行分词。其中 FMM 为从前向后进行匹配, 对每一个字进行遍历, 直到该字与词典中某一个词匹配则跳到该词的下一个字继续匹配, BMM 与之类似, 但是由后向前匹配。

基于统计的分词算法, 在本实验中体现为 N 元文法, 即在机械匹配中引入数学知识, 基于贝叶斯条件概率公式计算在已知前一个字基础上后一个词汇的切分概率, 一个句子或许有更多的匹配概率, 那么采用动态规划的原理选取概率最大的路径, 将其定义为该句的最优切分。这样可以有效克服机械匹配的交集型切分缺陷, 如王天明天要去野游, 如果根据 FMM

¹ <https://github.com/fxsjy/jieba>

² <https://github.com/thunlp/THULAC-Python>

³ <https://github.com/isnowfy/snownlp>

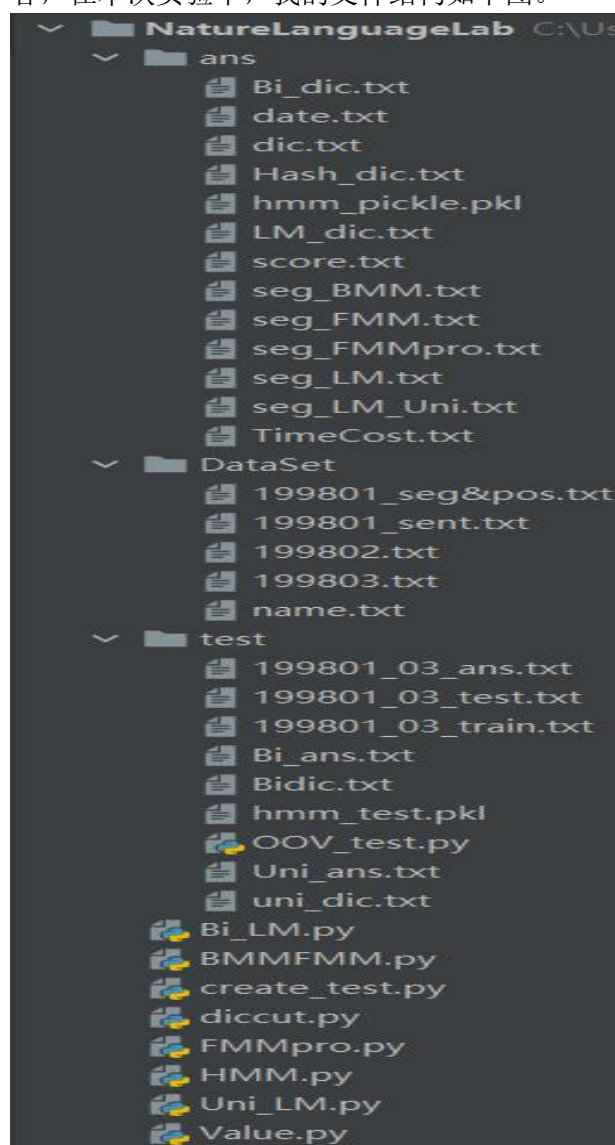
算法会被且分为王天明|天|要|去|野游|，而基于统计的分词方法会将其分为王天|明天|要|去|野游|，显然从句意上后者更胜一筹。基于统计的分词算法主要模型有：**N-gram**、**HMM**、**最大熵模型**、**CRF** 等等

3 实验文件说明

对本次实验的操作流程以及各文件的功能进行介绍，方便老师或助教进行进行检查

3.1 项目目录

NatureLanguageLab 为本次实验的全部内容，在本次实验中，我的文件结构如下图。



P1:文件结构示意图（其中 DataSet 文件夹为实验给出数据集，在下文不做介绍）

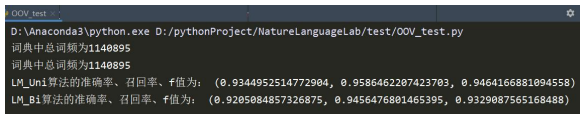
3.2 ans 文件夹为本次试验所有的生成结果

- Bi_dic: 二元文法词典，格式为前词\t后词\t在样本集中出现次数n---
- date.txt:日期，切分所有样本集剩下的日期文件（并未作为训练集进行训练）
- dic.txt:环节 3.1 要求输出，最基础的词典，形式为词汇/词性\n
- Hash_dic.txt:基于我自己构建的哈希算法构造的词典，用于机械匹配的优化，形式为词汇\t词性\t哈希值\n哈希值的计算在后续介绍
- hmm_pickle.pkl:基于 HMM 的训练模型，用于存储 pi、A、B 等 HMM 的基本参数，二进制文件
- LM_dic.txt: 基于统计的词典，形式为词汇\t词性\t出现次数n
- score.txt:环节 3.3（评估算法）的要求输出
- seg_BMM.txt、seg_FMM.txt:环节 3.2 的要求输出，FMM、BMM 算法的结果
- seg_FMMpro.txt:环节 3.4 的要求输出之一，FMM 基于哈希的速度优化输出
- seg_LM_Uni.txt:环节 3.5 的要求输出（实现了未登录词的识别）
- seg_LM.txt:环节 3.6 的要求输出，二元文法的输出结果（实现了未登录词的识别）
- TimeCost.txt:环节 3.4 的要求输出，消耗时间对比（由于运行 FMM 时间较长，已提前给出 FMM 优化前所需时间为：9249.723063468933（s），优化后所需时间为 74.53873038291931（s））

3.3 test 文件夹是未登录词识别的测试

- 199801_03_ans.txt:为分词标准结果，由样本集随机抽出（概率默认 50%，在 create_test.py 中给出）
- 199801_03_test.txt:为分词测试集，其抽取方法与 ans 一致，但是经过了去标签处理
- 199801_03_train.txt 为总样本集去除了 test 的部分
- Bi_ans.txt:二元文法的分词结果（未登录词）
- hmm_test.pkl:使用切分训练集的训练模型
- OOV_test.py: 该文件导入了二元文法 Bi_LM 一元文法 Uni_LM.py 以及评估模块，在此文件进行模型训练以及测试等工

作，结束后将结果写入 Bi_ans.txt 以及 Uni_ans.txt，运行未登录词识别结果如下图：



```
OOV_test
D:\Anaconda3\python.exe D:/pythonProject/NatureLanguageLab/test/OOV_test.py
词典中总词频为1140895
词典中总词频为1140895
LM_Uni算法的准确率、召回率、f值为: (0.9344952514772904, 0.9586462207423783, 0.9464166881094558)
LM_B1算法的准确率、召回率、f值为: (0.9205084857326875, 0.9456476801465395, 0.9329087565168488)
```

P2 未登录词识别结果（样本集测试集各为总集合 50%无交集）

- Uni_ans.txt:用于存储一元分词对未登录词的识别分词结果
- Uni_dic.txt:基于词频统计的一元文法词典

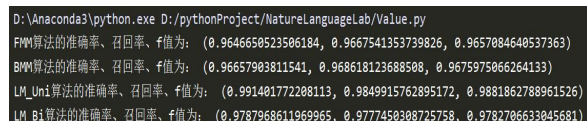
3.4 接下来按照实验流程介绍.py 文件：

- **diccut.py**:本实验的第一个任务，切分词典，在 3.4 之前构造 **diccut** 函数只需要完成简单的切分给出文本中的词汇即可，不需标注或记录其余功能，在环节 3.4 中添加 **My_Hash** 函数，将传入词汇哈希化，具体哈希函数为先计算词中每个字的 **ascii** 码，将其转化为十进制数累和，加单词长度模一万，即可得到冲突较小的哈希值。与其搭配使用的是 **Hashdic** 函数，输入原词典，可以将其转化为哈希散列值的新词典。在环节 3.5 添加 **LMdic** 函数，功能是构造基于词频统计的词典，当遍历训练集时，记录对应词汇出现次数，最后计入词典，环节 3.6 构造 **Bidic**，在每一句遍历前后词汇出现次数，将其写入二元文法词典中。**老师或助教验证 3.1 时**直接运行 **main** 函数即可，词典生成路径在 **main** 中有所体现。
- **BMMFMM.py**:本实验的第二个任务，即进行基于机械匹配的汉语词汇切分。主要功能即为实现基础的机械匹配算法。首先创建 **read_dic** 函数，用以读取初始词典，返回 **list** 形式⁴的词典（由于运行时间过长，可返回 **set** 形式的词典），词典中最大词长（用以直接限制最大匹配长度）**FMM**、**BMM** 函数即前、后向最大匹配算法**老师或助教验证 3.2 时**直接运行主函数即可，

⁴ 由于存储方式的不同，在 python 中 **list** 是线性存储，在搜索时需要遍历整个 **list**，时间复杂度为 $O(n)$ ，而 **set** 是散列存储，在搜索时可直接找到对应元素，时间复杂度为 $O(1)$ ，因此在需要多次遍历词典时，二者速度差距较大

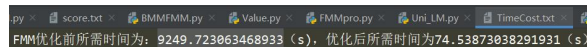
当前词典为 **set** 形式，因此分词耗时分别为 3s 左右。若将 **read_dic** 函数返回值去掉 **set()**，运行时间约为 9249.7230 秒

- **Value.py**:本实验的第三个任务，对分词结果进行评估。首先对待分词文件进行预处理，**PreProcess** 函数将文本中所有词汇提取，按照每一句*每一个词汇的方式进行处理，返回相同的形式。**calculate** 函数对两个文本进行比较，计算准确率和召回率以及 **f** 值，计算思想为统计两个文本中相同分词的数目（**TP**）以及两个文本总词数（**TP+FP**、**TP+FN**）精确率 $P=TP/(TP+FP)$ ，召回率 $R=TP/(TP+FN)$ ， $F1=2PR/(R+P)$ 。在判断是否正确时依据为两词完全相同，否则长度较短的文本合并其下一词条，直至相等。**老师或助教验证 3.3 时**，可基于任一格式符合要求的结果文件与标准分词文件进行比较，输出结果会写到结果文件中并在运行窗口显示。如下图



```
D:\Anaconda3\python.exe D:/pythonProject/NatureLanguageLab/Value.py
FMMpro算法的准确率、召回率、f值为: (0.9646658523586184, 0.9667541353739826, 0.9657084648537363)
BMM算法的准确率、召回率、f值为: (0.96657903811541, 0.968618123688508, 0.9675975066264133)
LM_Uni算法的准确率、召回率、f值为: (0.991401772288113, 0.9849915762895172, 0.9881862788961526)
LM_B1算法的准确率、召回率、f值为: (0.9787968611969965, 0.9777450308725758, 0.9782706633045681)
```

- **FMMpro.py**:本实验的第四个任务，对机械匹配算法进行速度优化，我构造了基于哈希的优化算法。**hash_dic** 函数将哈希词典导入，创建二维数组，在对应哈希值的索引加入对应元素，由于哈希值是模 10000 得出的，因此创建二维数组空间为 10000 行即可。**Hash_FMM** 即为基于哈希快速匹配的 **FMM** 算法，**老师或助教验证 3.4 时**，需将 **BMMFMM.py** 中 **read_dic** 方法返回值改回 **list** 形式，再运行主函数否则会覆盖已经跑好的程序结果



```
py score.txt BMMFMM.py Value.py FMMpro.py Uni_LM.py TimeCost.txt
FMM优化前所需时间为: 9249.723063468933 (s), 优化后所需时间为: 74.53873038291931 (s)
```

- **Uni_LM.py**:基于统计词频的一元文法分词，**read_pre_dic** 函数用于读取词典，将基于统计词频的词典创建为{词: 频率}形式的字典列表。其中存在某词因为不同词性而被错分的现象，由于一元文法不对词性区分，因此检查若词已经出现在字典中，频率等于总和。**join_DAG** 函数利用前缀词典将待分词句子切割成有向无环图，每一个节点 **i** 存储的是第 **i** 个字与之后字结合成词的可能路径。**best_way** 函数是本文件的关键函数，用于寻找该句子的最佳分词形式。通过计算有向无环图中概率最

大路径返回具体路径。**Uni_seg** 函数将前几个函数统一调用，对每一个句子进行上述切分操作。**老师或助教验证 3.5 时**，无需经行额外操作，直接运行即可，在评估算法中已经将该输出地址作为评估参数之一。

- **HMM.py**: 隐马尔可夫模型类，用于训练隐马尔可夫模型，包含 HMM 类和 HMMTRAIN 类，HMM 包含基于 BMES 模型，（即每一个字符更倾向于在词语中的什么位置）训练隐马尔可夫模型基本参数，即 $\{A, B, \pi\}$ （状态转移概率、发射概率、初始状态集），HMMTRAIN 用于训练模型，将其以序列化存进 pickle 中（pickle 的优点为已知类的成员变量则在读取时只需反序列化读取文件即可直接获取模型），在本次实验中，HMM 模型主要用于训练未登录词识别，即 3.6 优化部分。
- **Bi_LM.py**: 实现二元文法，首先使用 **get_Bi_dic** 函数获取前缀词典，生成模式为对应第一个词若已存在则将其定义为字典索引，将其后出现的第二个词汇及其概率作为字典放入该索引的 value 中，即 {词 1: {词 21: freq1}{词 22: freq2}---} 然后是划分等价类算法: **generate_equal_class** 以及还原等价类算法: **replace_equal_class** 算法目的是识别未登录词中诸如日期、百分数、小数等格式统一且会耗费大量算力的词汇，将其转化为单一不易出现的单字，例如 @#¥%.....，在识别结束后将其复原。在未登录词时可能会出现大量概率为 0 的情况，这时需要平滑算法 **cal_log** 防止无法计算的情况出现。本实验中采用一元二元相结合的线性插值法，这样可以保证相差概率最大化。**best_way** 即根据有向无环图进行动态规划求一个句子划分的最优解
- **Create_test.py**: 切分测试集，将给出的三个样本文件切分为无交集的测试集和训练集，为未登录词识别提供训练集和测试集。

4 算法伪代码实现:

由于算法代码量过大，在此给出较复杂算法的伪代码或流程图介绍:

FMM、BMM

输入: 词典 (dic)、待分词文本 (text) (以每个句子为例)

1. Function FMM (dic, text)
2. Max = max(len(i) for i in dic)
3. For j in range(len(text)):
4. 从前向后 (FMM) 从后向前 (BMM) 遍历每一个字，从字到最大长度组成的词语是否与词典中词匹配
5. If match:
6. Seg.append(词)
7. J += 词长
8. If not match:
9. 词长 -= 1

viterBi 算法

输入: 观测值状态取值空间 O、状态转移空间 S 观测序列 Y 转移矩阵 A

1. function Viterbi(O, S, Y, A)
2. for si in S:
3. T1[i, 1] = Π_i
4. T2[i, 1] = 0
5. For i in range(2, T):
6. For sj in S:
7. T1[j, i] = max(T1[k, i-1] A_{k,j})
8. T2[j, i] = max(T1[k, i-1] A_{k,j})
9. ZT = max(T1[k, T])
10. XT = SZT
11. For i in range(2, T):
12. Zi-1 = T2[Zi, 1]
13. Xi-1 = SZi-1

5 结果分析:

[1] : 3.4 环节给出速度结果分析, 分析要素为分词速度, 结果如下:

分词算法	FMM	Hash_FMM	Set_FMM
运行时间(s)	9249.723033	74.538730	4.215648

由表可知我的哈希算法虽然可以较为有效的减少程序运行时间, 但是与 python 自带的 Set 形式仍有较大的差距。原因为我定义的哈希算法冲突较多, 在遍历时平均每个哈希值有 6 个值左右, 因此在遍历每一个句子中的每一个字时, 搜索带来的差距约为 20 倍。但是与线性查找的 list 对比, 性能优化了 150 倍。

[2] : 3.3 环节给出性能结果分析, 分析要素为: 精确度、召回率、F 值, 结果如下

```
Value (1)
D:\Anaconda3\python.exe D:/pythonProject/NatureLanguageLab/Value.py
FMM算法的准确率、召回率、f值为: (0.9647564110547554, 0.9668033381234262, 0.9657787899965934)
BMM算法的准确率、召回率、f值为: (0.9667009320055222, 0.9667035547762499, 0.967701207303084)
LM_Uni算法的准确率、召回率、f值为: (0.9914004881401753, 0.9849942650723128, 0.988186994142435)
LM_Bi算法的准确率、召回率、f值为: (0.9820157200116968, 0.9769820657165583, 0.9794924258645864)
```

分词算法	FMM	BMM	LM_Uni	LM_Bi
准确率	0.964756	0.966700	0.991400	0.982015
召回率	0.966803	0.966703	0.984994	0.976982
F 值	0.965778	0.967701	0.988186	0.979492

由表格可知在进行封闭集训练时, 基于统计的分词方法性能显著优于基于机械匹配的分词方法, 这是因为机械分词无法处理分词歧义, 而基于统计的分词方法可以寻找概率最大路径, 从而去除因前词虽也能成词, 但后词概率更大而错误分词的情况。此外二元分词结果略低于一元分词, 这是因为我的平滑函数使分词概率发生改变, 对于低概率成词的二元词汇概率可能会因为第二个词汇概率偏高而大幅上升, 因此性能有所下降。

[3] : 未登录词结果分析, 分析要素为准确率, 召回率, F 值, 结果如下

```
OOV_test
D:\Anaconda3\python.exe D:/pythonProject/NatureLanguageLab/test/ooV_test.py
词典中总词频为1754886
LM_Bi算法的准确率、召回率、f值为: (0.9370529821976714, 0.951940975308198, 0.9444383093384618)
词典中总词频为1754886
LM_Uni算法的准确率、召回率、f值为: (0.9428969289146973, 0.9612524166936377, 0.9519862018204235)
```

分词算法	LM_Uni	LM_Bi
准确率	0.942896	0.942896
召回率	0.951940	0.961252
F1 值	0.944438	0.951986

测试集选取使用留出法, 在总样本集中选取 50%作为训练集, 另外 50%作为测试集, 二者互不交叉, 通过对比分词结果, 我们发现二者准确率较封闭训练性能均下降 5%左右, 且二元文法较一元文法性能更差, 因为二元文法依赖词典中是否含有词对比一元文法依赖性更高, 因此更难以实现准确预测。

6 参考文献:

- [1] 宗成庆. 统计自然语言处理. 清华大学出版社: 第二版
- [2] <https://blog.csdn.net/baimafujinji/article/details/51281816>
- [3] <https://blog.csdn.net/u011407081/article/details/70673309>
- [4] 尹陈, 吴敏. N-gram 模型综述[J]. 计算机系统应用, 2018, 27(10): 33-38. DOI: 10.15888/j.cnki.csa.006560.
- [5] 于童, 刘淑芬. 构建单字词表识别未登录词的方法[J]. 吉林大学学报(理学版), 2015, 53(02): 307-310. DOI: 10.13413/j.cnki.jdxblxb.2015.02.29.