基于规则的中文分词 实验报告

人工智能学院 张含笑 181220067

实验环境

Phthon 3.6.5 + windows 10 + vscode

实验方法

基于规则的中文分词，是基于前向、后向、双向最大匹配的分词方法，除此之外再根据训练集添加规则，针对特定pattern的字段再进行处理.

本实验中, 我先写了基本的前向(FMM(dic, text, window_size))、反向(RMM(dic, text, window_size))、双向最大匹配(BIMM(dic, text, window_size))的分词函数, dic是训练集所有字段去重获得的集合, window_size是字典里的最大词长.

- 前向匹配是指从待分词句子的头部开始匹配向后截取长度为window_size递减到1的字段, 在dic中搜索, 匹配成功则继续向后匹配, 匹配不成功则只按长度为1切分.
- 反向匹配与前向相似, 区别在从后向前匹配切分.
- 双向匹配旨在选则前向和反向中更好的那个(更好的标准是优先选分段少的结果, 其次选单字字段少的结果).

后续我对比程序在dev上运行的结果和dev正确的分词, 想要将所有相邻的、未成功匹配的、单字的字段合并起来. 于是我在FMM, RMM, BIMM中加了label列表用于记录分词后的各字段是否被成功匹配(0为匹配, 1为未匹配).tidy()函数用于合并字段; 但是符合条件的全部合并是很草率的做法. 因此我根据结果和标准对比添加规则, 将其他label改为1, 需要合并的字段对应的label改为0, 这样在原来基础上不需要在匹配规则后对分词列表有什么实际的改动, 而是最后根据label直接合并, 且依旧只需遍历一遍, 复杂度还是$\mathcal{O}(n)$. 后续的方法证明用label标记确实是比直接改动更方便.

添加规则后合并低频词, 这里的低频词是指在按照规则分好的字段中出现次数较少的字段. 这里我直接把label重新理解为出现的次数, label小于shreshold的字段进行合并, 因此之前被规则匹配到的字段label改为0并不影响其在之后的处理中被合并.

添加足够多的规则后再逐渐向训练数据里添加额外的数据集, 观察实验结果的变化, 选择对提升结果有用的数据.

实验结果

以下是我记录的有提升的改进和提交结果.

	较上次的改动	**test-F1**
1	未加其他规则的正向最大匹配	0.8532675987855692
2	未加其他规则的反向最大匹配	0.8552775124215001
3	未加其他规则的双向最大匹配	0.8555176722700004
4	将相邻的全数字字段/全字母字段合并	0.8701435111714941
5	将['.',':',' : ',':']和其前后的数字合并	0.8710951115682409
6	将'%'和其之前的数字合并	0.8712905235126378

	较上次的改动	test-F1
7	将姓氏和其之后的1-2个单字字段或1个双字字段合并	0.8714393299185395
8	列举量词并将其和其之前的数字合并	0.8713132103253117(为什么会下降?)
9	将末位为数字和首位为数字的相邻字段合并	0.8732143165355871
10	将'.'和前面的英文合并起来(表示缩写)	0.8733083225925244
11	将'&'和其前后的字段合并起来(一些品牌名)	0.8733945587626019
12	将非首字段的'-'和其前后的字段合并起来	0.8734171386471428
13	将'↓↓'分割为'↓', '↓'	0.8735489292147959
14	将','和其前后数字合并	0.8736999027265351
15	将'@'和其前后的英文合并	0.8737099605721358
16	将姓氏匹配改为合并姓氏和其之后的2个单字字段	0.8753619857504022
17	增加“~”前后数字的拼接	0.8753540913713751
18	加入English_Cn_Name_Corpus（48W）.txt	0.87762620157716
19	增加 '·' , '•' 前后中文的拼接	0.8780305211632594
20	加入ChengYu_Corpus（5W）.txt	0.8817508962646005
21	加入chengyudaquan.txt	0.8818008870282241
22	加入THUOCL_chengyu.txt	0.8817583586801901(下降, 去掉)
23	加入dieci_bank.txt	0.8818828427687504
24	加入idioms.txt, idioms_2.txt	0.8818403141209875(下降, 去掉)
25	加入road_bank.txt	0.8818892172796521
26	加入famous_people.txt	0.8822314647168731
27	加入global_locations.txt	0.8834549695282522
28	加入locations_2.txt	0.8842929269535411
29	加入pku_training.txt	0.8893574167103458
30	将所有未匹配到的连续单个字字段合并	0.8904983028474449
31	低频单字字段合并, threshold = 10	0.896
32	shreshoid = 9	0.8956
32	threshold = 11	0.896886190622999
33	threshold = 12	0.89752369197173
34	threshold = 13	0.8979550636519095

	较上次的改动	test-F1
35	threshold = 14	0.8985934408110148
36	threshold = 15	0.8989155509048966
37	threshold = 16	0.8995170346402741
38	threshold = 17(退回16)	0.8994526415992385
39	去掉pku_training.txt	0.9015818254359408
40	threshold = 17	0.9021166195132831
41	threshold = 18	0.9028586196162273
42	threshold = 19	0.9040564373897708
43	threshold = 20	0.9046341406010968
43	threshold = 21	0.9051010677748609
44	threshold = 22	0.9053898846160642
45	threshold = 23	0.9056957436650104
46	threshold = 24	0.9058544583770776
47	threshold = 25(返回25)	0.9056744032377962
48	加入idioms.txt, idioms_2.txt, threshold = 26	0.9059853273604004
49	threshold = 27	0.9062341312485976
50	加入food_bank.txt, threshold = 28	0.9062559062559061
51	threshold = 29	0.906463087486262
52	threshold = 30	0.9068936975286744
53	threshold = 31	0.9069473186306392

结果分析

一开始我想直接合并连续未匹配单字字段, 分词后需要遍历一遍, 程序复杂度为$\mathcal{O}(n)$.

添加规则后依旧只遍历一遍结果, tidy()复杂度依然为$\mathcal{O}(n)$.

最后添加的合并低频词方法对程序运行时间影响较大, 在main函数中, 我按句调用分词函数, 合并左右的结果, 统计所有的字段出现的次数, 再将label改为次数, 因此时间复杂度是$\mathcal{O}(n^2)$. 添加合并低频词之前程序运行时间为2-3s, 添加之后变成60-70s.

基于规则的分词方法几乎全是依赖于训练集和测试集的相似性, 以及不同的训练集在某几个方面的分词习惯不同于测试集, 因此会带来误差. 因此实验结果的差距主要来自于数据集的大小、选取和规则的细致程度.

使用的数据集

ChengYu_Corpus（5W）.txt

chengyudaquan.txt
dev.txt
dieci_bank.txt
English_Cn_Name_Corpus（48W）.txt
famous_people.txt
food_bank.txt
global_locations.txt
idioms.txt
idioms_2.txt
locations_2.txt
road_bank.txt
train.txt

致谢

感谢陈思佳、葛书虬同学对我添加规则方面提供的帮助, 以及陈思佳、葛书虬、杨旖纯同学对我实验进度的督促.