自然语言处理实验一：汉语分词系统

张瑞
哈尔滨工业大学
1190201421@stu.hit.edu.cn

摘要

本文通过实验构建了一个汉语分词系统，包括词典的建立、分词算法的实现、性能优化和评价等环节。主要涉及到语料库相关知识、正反向最大匹配分词算法、N 元语言模型、隐马尔可夫模型、分词性能评价常用指标、文件处理与数据统计等基本编程能力以及查找算法与数据结构实现能力。

1 引言

词是自然语言中能够独立运用的最小单位，是语言信息处理的基本单位。但由于汉语不同于英语一类的文字，没有天然的词的切分，所以需要在处理汉语信息前将句子转换成词序列，这便是汉语分词。

目前的汉语分词已经达到了较高水平，但仍然存在分词规范多样化、词定义不明确、交集型歧义、组合型歧义、未登录词和新词识别困难等问题。

在处理汉语分词的问题上，主要有两种方法：一种是理性主义的分词方法，另一种是经验主义的分词方法。

基于字符串匹配的分词算法（也称基于机械匹配的分词算法）就是理性主义的代表，它主要是按照一定策略将需要分词的字符串与词典中的词进行匹配，包括正向最大匹配分词、反向最大匹配分词、双向最大匹配分词和最少分词等方法。

经验主义的萌芽是从最大词频分词法开始的，其基本思想认为出现频率越高的词越可靠。利用统计机器学习模型学习大量已分词文本中的分词规律，便能很好地对未知文本进行分词，这便是基于统计语言模型的分词算法，主要包括N元语言模型和隐马尔可夫模型等。

本文将在对正反向最大匹配分词算法、基于二元语言模型的分词算法和基于隐马尔可夫模型的未登录词识别的实现与分析过程中，初步探索汉语分词。

2 词典的构建

本文所使用的语料全部来源于北京大学计算语言学研究所的 1998 年《人民日报》分词语料库。理想情况下，为了提高分词的准确率，词典自然是越丰富越好。但是在实验中我们能发现，词典如果过大，很容易产生性能上的瓶颈，时间复杂度和空间复杂度都会受到影响，而且并不是所有的词都适合被放入词典中。因此，在构建词典的时需要按照一定标准做出取舍。

2.1 分词单位标准

抽取 199801_seg&pos.txt 中的全部词语作为放入词典的词，但需要注意以下几点：

- 数词不放入词典。对于像“三百六十五”和“3000”这样的数词，理论上会有无数个，放进词典是很不现实的，而且必定对时间复杂度和空间复杂度都有影响。
- 时间词不放入词典。同数词一样，以“年”、“月”、“日”、“时”、“分”、“秒”和“点”结尾的时间词理论上也是有无数个，不应直接放入词典中。
- 不考虑短语。对于一些用“[]”括起来的机构团体名，分词标准中将其按照多个词语进行切分，故将各词语放入词典，而不把各个词语当做一个整体放入词典。

2.2 词典文件格式

生成的词典文件 dic.txt 采用 utf-8 编码，其中的词是经过排序的、不重复的，一词一行。若用 w 表示一个词，则词典格式如下：

$$\begin{matrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{matrix}$$

2.3 对词典的分析

经统计，源语料 199801_seg&pos.txt 中共有 1140931 个词，其中数词有 60854 个，以“年”、“月”、“日”、“时”、“分”、“秒”和“点”结尾的时间词有 9837 个。经过筛选与去重，最终生成的词典文件共包含 50703 个词。从上述数据可以看到，生成的词典在极大程度上简化了源语料，可被视为对源语料的一个抽象概括。

同时，词典中词的有序排列不仅更贴合实际词典的处理方式，可读性强，而且也能在一定程度上启发后续查找算法与存储结构上的优化。

3 正反向最大匹配分词实现

3.1 正向最大匹配（FMM）算法

（1）获取分词词典中最长词条长度 L；
（2）取待分词句子中前 L 个字符作为匹配字段，记录其起始下标 start，查找分词词典，若匹配成功，该字段作为一个词被切分出来；
（3）若词典中找不到该词，去掉匹配字段的最后一个字符，继续匹配，直至切分成功，记录分词长度 len；
（4）更新 start 为 start+len，若 start 到达句末，则切分完毕，返回结果；否则转（2）继续切分。

3.2 反向最大匹配（BMM）算法

（1）获取分词词典中最长词条长度 L；
（2）取待分词句子中后 L 个字符作为匹配字段，记录其结束下标 end，查找分词词典，若匹配成功，该字段作为一个词被切分出来；
（3）若词典中找不到该词，去掉匹配字段的第一个字符，继续匹配，直至切分成功，记录分词长度 len；
（4）更新 end 为 end-len，若 end 到达句首，则切分完毕，返回结果；否则转（2）继续切分。

3.3 最少代码实现的收获

（1）为了达到最少代码的要求，在数据结构上选用了 Python 自带的 List，在查找算法上简单使用了 List 自带的关键字 in 来实现，这将导致每次查询的时间复杂度为 O(n)。当词典越大，分词速度会越慢，所以这种简单的方案是不适合在大规模数据上采用的。
（2）分词词典中最长词条长度 L 为 26，但绝大多数词长度为 1-4。当切分需要从 26 开始进行匹配的时候，效率并不高。

4 正反向最大匹配分词效果分析

4.1 评价标准

- 准确率 precision：

$$准确率P = \frac{切分结果中正确分词数}{切分结果中所有分词数} \times 100\%$$

- 召回率 recall：

$$召回率R = \frac{切分结果中正确分词数}{标准答案中所有分词数} \times 100\%$$

- F 值：

$$F - 指标 = \frac{2PR}{P+R}$$

4.2 分词精度差异分析

正反向最大匹配分词的差异往往出现在对相邻三字的处理上。例如 BMM 会把“开创新的业绩”分为“开/ 创新/ 的/ 业绩”，而 FMM 能正确划分为“开创/ 新/ 的/ 业绩”；FMM 会把“不平凡”分为“不平/ 凡”，而 BMM 能正确划分为“不/ 平凡”。

这一分词结果差异必定会导致两种方式的精度产生差异，而决定谁的精度更高的关键在于待分词文本中哪种类型占比更多，这是与语料内容高度相关的。

另外，分词的差异能分为两种——“真歧义”和“假歧义”。例如“必须/ 加强/ 企业/ 中/ 国有/ 资产/ 的/ 管理/”和“中国/ 有/ 能力/ 解决/ 香港/ 问题/”中的“中国有”就是“真歧

义”，它存在两种可实现的划分且都是正确的；而“中国/ 人民”和“各/ 地方”就是“假歧义”，看上去也能有多种划分，但实际只有一种是正确的。如果遇到“假歧义”，我们能直接对 FMM 与 BMM 的优劣下结论，并初步判断两者精度的高低差异；但在遇到“真歧义”时，我们难以对两者的优劣做出判别，因为每一种划分可能在这一个测试集上的是正确的，在另一个测试集上就是错误的了，这时对两者的性能好坏便难以界定。

但是，通过大量的统计实验可以发现，在汉语分词中，BMM 总体来说比 FMM 更有效，这是因为汉语本身的侧重点倾向于在句后，在一个已有词的基础上，如果在其词前加字则更容易改变原意变成非词结构。这样的特点使得 BMM 更容易检测出正确的符合原语义的词。

5 基于机械匹配的分词系统的速度优化

5.1 优化方案

为了提升分词速度，可以从数据结构和查找算法两个方面考虑进行优化。本文优化是基于 FMM 进行的优化。

在数据结构方面使用 Trie 树来进行优化。因为观察词典能发现，大量的词会拥有公共前缀，若合理利用该特点将词典构造成 Trie 树就能大量减少查询词典的次数。对于 Trie 树的每一个结点，需要保存的是单个字符、是否可作为一个词的词尾以及指向其子结点的指针。从根节点到任意可作为词尾结点的路径即为一个词。特别指出，根节点为空字符，其各个子结点中字符才是各词词头。

在查找算法方面使用 Hash 函数来进行优化，这样便可以减少查词典的时间。前面提到每个 Trie 树的结点都要存储指向其子结点的指针，这些指针是存储在 List 中的，而在 List 中的排列顺序是用子结点的 Unicode 编码作为 Hash 函数映射得来的，处理冲突采用的是线性探测法。

优化后的算法流程如下：

（1）取待分词句子中第一个字符作为匹配字段并记录；

（2）判断匹配字段是否在词典（Trie 树）中；

（3）若匹配字段在词典中，且最后一个字可作为词尾，则记录当前匹配字段，若尚未到达待分词句子句末，将目前匹配字段后一个字符纳入匹配字段内，跳转（2）继续匹配，否则跳转（5）；

（4）若匹配字段在词典中，但最后一个字不可作为词尾，若尚未到达待分词句子句末，将目前匹配字段后一个字符纳入匹配字段内，跳转（2）继续匹配，否则跳转（5）；

（5）若匹配字段不在词典中或已到达待分词句子句末，则输出目前所记录的匹配字段。若待分词句子中该匹配字段后无字符，切分结束；否则取该匹配字段后一个字符为新的匹配字段，跳转（2）继续匹配。

5.2 优化技术效果

分别对优化前后的分词程序进行计时（不包含初始化时间，仅计算分词所用时间），可以发现速度有显著提升。优化前的 FMM 运行时间约为 24559s（约 7 小时），优化后的 FMM 运行时间约为 7.9s。

5.3 进一步优化思路

若要进一步优化分词系统，可以考虑从 Hash 函数入手，找到更为合适的函数从而减少冲突，降低因冲突而导致的额外查找开销。当然也可以考虑增加每个结点的子结点列表长度，这样也能减少冲突，以时间换空间，达到速度进一步提升的目的。

6 基于统计语言模型的分词系统实现

本文实验采用二元语言模型来探究统计语言模型。

6.1 二元语言模型

基础理论

由贝叶斯公式可得

$$\begin{aligned} Seg &= arg\max_{Seg} P(Seg|Text) \\ &= arg\max_{Seg} \frac{P(Text|Seg)P(Seg)}{P(Text)} \\ &\propto arg\max_{Seg} P(Text|Seg)P(Seg) \\ &= arg\max_{Seg} P(Seg) \end{aligned}$$

将 Seg 简写为 S，二元语言模型认为每个词的出现仅与它的前 1 个词有关，则有

$$P(S) = P(w_1)\prod_{i=2}^{n} P(w_i|w_{i-1})$$

于是最终的分词应满足

$$\text{Seg} = argmaxP(w_1)\prod_{i=2}^{n} P(w_i|w_{i-1})$$

生成词典

首先生成前缀词典与二元语法词典。提取训练集中的所有词及其对应词频，再将每个词的所有前缀都加入词典中，对于未在词典中出现过的前缀词，将其词频全部设为 0，这便得到了前缀词典。取出训练集中的每一行，在行首行末分别加入“<BOS>”和“<EOS>”，再将任意两个相邻分词作为一对词对放入词典，并统计每对词的出现频率，这便得到了二元语法词典。

DAG 构造

由于二元语言模型涉及到求最大概率路径，要进行有向图切分，这里再介绍下 DAG 的构造方式。

本文实验采用字典存储 DAG，key 为待分词文本中各字对应的下标，value 为能以 key 作为词头的所有词的词尾下标。用一个待分词句子生成 DAG 的算法流程如下：

（1）遍历句子，对于每个下标 x，取下标 x 对应的字为匹配字段；

（2）判断该匹配字段是否在前缀词典中；

（3）若在词典中且词频大于 0，则将该字段的尾字下标 y 加入到 x 对应的 value 值中，若尚未到达句末，将匹配字段后一个字纳入匹配字段中，转（2），否则转（6）；

（4）若在词典中且词频为 0，若尚未到达句末，则将匹配字段后一个字纳入匹配字段中，转（2），否则转（6）；

（5）若匹配字段不在词典中，则以 x 作为词头的词已经搜索完毕，若尚未到达句末，转（1）继续遍历，否则转（6）；

（6）已到达句末，构建结束。

动态规划

得到 DAG 后，便可以进行动态规划，其算法流程如下：

对于词图中的每个词w_i：

（1）若w_i为“<BOS>”，则将其概率对数$p_\log(w_i)$设为 0，标记它的前词为 None（意即该词必在句首出现）；

（2）若w_i有前词，遍历考虑其所有前词w_{i-1}，取$\max(P(w_i|w_{i-1}) + p_\log(w_{i-1}))$为其概率对数，并标记其前词为$w_{i-1}$；

（3）若w_i没有前词，将其概率对数设为极小值-99999999（这样做是为了保证下一步计算可进行），并标记其前词为 None。

完成动态规划之后，从句尾（“<EOS>”）开始逐渐确认前一个分词的结果直至遇到“<BOS>”，便得到整个句子的最大概率切分路径。

平滑处理

若二元语言模型中某 1 对词对未在词典中出现，词频被记为 0，就会导致概率连乘时其余词对的概率对最终结果无贡献的问题。为了解决这一问题，在计算$P(w_i|w_{i-1})$时，我们可以假设每个二元语法词对的出现次数都比实际的次数多一次，即

$$P(w_i|w_{i-1}) = \frac{1 + n(w_i w_{i-1})}{N + \sum_I n(w_i w_{i-1})}$$

同时，在计算$P(w_i|w_{i-1})$时，也可以考虑w_i和w_{i-1}是否条件独立，分别计算独立与不独立两种情况下的概率，并将其线性组合作为最终概率。本质上来讲，这一组合方式是对一元语法和二元语法的融合。

6.2 未登录词识别

本文实验采用隐马尔可夫模型（HMM）对未登录词进行识别。

HMM 是一个双重随机过程，由一条隐藏的状态转移链和观察状态对应值的随机过程组成。

HMM 主要有三个重要的矩阵：初始状态概率分布矩阵π、状态转移矩阵 A 和符号发射概率矩阵 B。

其中，π中元素π_i表示初始状态为 i 的概率，即

$$\pi_i = P(q_1 = s_i)$$

A 中元素a_{ij}表示从状态 i 转移到状态 j 的概率，即

$$a_{ij} = P(q_t = s_j|q_{t-1} = s_i)$$

B 中元素b_{jk}表示在状态 j 下观测到符号 k 的概率，即

$$b_{jk} = P(O_t = v_k|q_t = s_j)$$

本文实验中将训练集中的所有分词结果进行单字符拆分，并标注其状态为 S（单字词）、

B（词头）、M（词中）或 E（词尾），再根据以上定义，很容易便能进行参数的初始化。

当初步分词（如二元语法分词）完成后，将该结果按行进行未登录词识别处理即可，由于未登录词会被切碎成一个个单字词，所以我们主要关心的便是分词结果中连续出现的单字词，算法流程如下：

（1）读取一行分词结果，遍历每一个分词，并判断是否为单字词；

（2）若为不在词典中的单字词，将其加入缓冲区等待后续处理，若为一行中最后一个分词，进行缓冲区处理，否则跳转（1）继续处理；

（3）若为多字词或在词典中的单字词，则进行缓冲区处理（若缓冲区非空），再单独输出该单字词，跳转（1）继续处理。

上述提到的缓冲区处理流程如下：

（1）读取缓冲区内容，遍历每一个单字，若缓冲区只有一个单字，直接划分，处理结束；

（2）计算第一个单字c_0的 4 个状态对应概率对数$\log p(c_0)$，并记录 4 个状态对应的状态链结果（此时仅为第一个单字状态本身）；

（3）计算第一个单字后各个单字c_i的 4 个状态对应概率对数$\log p(c_i)$（需要遍历前一个字c_{i-1}的所有可能状态，选取能使$\log p(c_i)$最大的状态作为c_{i-1}的状态），并记录4个状态对应的状态链结果（此时为第一个单字c_0到当前单字c_i的各个状态）；

（4）选取能使对应概率对数最大的状态作为最后一个单字的状态，查询该状态下的状态链结果，并按该结果划分，处理结束。

6.3 分词性能比较

本文实验采用 199802.txt 作为训练集，199801_sent.txt 作为测试集，用二元语法实现统计语言模型分词系统，用 HMM 实现未登录词识别，结果如下：

分词方法	准确率 P	召回率 R	F 值
2gram	91.48%	94.58%	93.00%
2gram+hmm	91.52%	94.60%	93.03%

可以看到二元语法模型已经有不错的表现，其中准确率低于召回率是因为未登录词等被切碎为单字词导致的，这一值在进行未登录词识别后有所上升，也从侧面说明了未登录词处理的正确性。同时，由于未登录词处理倾向于把单字词合并，这可能破坏某些单字词的正确划分，所以也会导致召回率的下降。但是从整体上来看，最终的 F 值会略有提升。

7 总结与展望

本文实验从机械匹配和统计语言模型两个角度分别实现了汉语分词，但仍然为十分基础的工作，还有相当大的提升空间。其中，机械匹配分词的优化可以从数据结构和查找算法入手；统计语言模型分词的优化可以从数据平滑处理、未登录词识别处理入手。同时，实验所采用的词典也相当关键，还可以作进一步的筛选与补充。

致谢

感谢杨沐昀老师及两位助教为本文实验所提供的支持。

参考文献

李舰.中文分词中的统计学[J].中国统计,2020(10):34-35.

邹佳伦,文汉云,王同喜.基于统计的中文分词算法研究[J].电脑知识与技术,2019,15(04):149-150+153.DOI:10.14004/j.cnki.ckt.2019.0537.

丁洁,赵景惠.基于N-gram模型的中文分词算法的研究[J].福建电脑,2017,33(05):110+116.DOI:10.16707/j.cnki.fjpc.2017.05.059.

黄昌宁,赵海.中文分词十年回顾[J].中文信息学报,2007(03):8-19.