中文自动分词系统

1190501614 cgh

计算机科学计算与学院

1190501614@stu.hit.edu.cn

摘要

中文分词是指将中文语句中的词汇按照使用时的含义切分出来的过程，自诞生之初经历词表匹配分词、统计分词到如今基于机器学习分词的过程。而本次实验的目的是为了基本了解汉语自动分词系统，包括词典建立，匹配分词、统计分词和优化等环节。在此次实验中，我们首先探究了前向最大匹配机械分词和逆向最大匹配机械分词对给定的语料库的分词效果，并提出了基于词性的分析方法来解析 FMM 和 BMM 的分词差异。然后基于哈希映射和字典树结构对匹配分词算法进行了速度优化，显著地降低了计算成本。最后基于统计方法建立了一元文法和二元文法的中文分词模型，并实现了 HMM 对未登录词的识别。

I. 绪论

中文自动分词属于自然语言处理的一个重要分支，自 20 世纪 80 年代以来，中文自动分词就一直是一个研究热点，目前一直处于发展阶段。而国外对于自然语言的研究确实得到不少卓有成效的成果，但它们大多数是基于英文的研究，其研究结果和方法无法直接迁移到中文系统中。

对于中文分词，首要问题是要解决如何定义分词单位，而不同的分词规范对此有不同的定义，目前以国家规范为主。而对于如何解决分词任务的问题，最开始的时候，是基于词典的研究诞生了正向最大匹配机械分词和逆向最大匹配分词算法，取得了一定成果，但该类算法依赖于所构建的词典，不同的领域需要构建不同的词典，并且难以识别未登录词。后来，随着统计学的发展，出现了基于 HMM 的分词方法和 N 元文法分词。从目前来看，HMM 在未登录词识别方面仍然取得不错的效果，但不能单独用于分词任务，需要配合其它分词方法或模型。而 N 元文法随着 N 的越大，模型的复杂度也会越来越高。现在，对于中文分词的研究，有不少学者从机器学习和深度学习中得到启发，通过应用机器学习的方法去学习分词模型，取得了一定的成果。

在本次实验中，我首先构建了分词词典，然后基于所构建的词典对训练文本进行了正向最大匹配分词和逆向最大分词，并且通过分析两种分词算法对不同词性的出错偏向找到分词差异。此外，还基于哈希技术和字典树对两种分词算法进行了速度优化，显著地降低了分词的计算时间成本。最后，基于统计的方法建立了一元文法和二元文法+HMM 的中文分词模型。

II. 相关工作

A. 正向最大匹配分词

该算法的基本思想是假定分词词典中的最长词的长度为 length，则用被处理文档的当前字串中的前 length 个字作为匹配字段，查找字典。若字典中存在这样的一个相匹配词，则匹配成功，匹配字段被作为一个词切分出来。如果词典中找不到这样的一个词，则匹配失败，将匹配字段中的最后一个字去掉，对剩下的字串重新进行匹配处理，如此重复进行下去，直到匹配成功，即切分出一个词或剩余字串的长度为零为止。这样就完成了一轮匹配，然后取出下一个待匹配的字段进行匹配处理，直到文档被扫描完为止。该算法依赖于所构建的词典，取得了一定的分词效果。但该算法无法实现对未登录词的识别，也无法消除歧义的影响，一般需要配合其它算法进行分词。

B. 逆向最大匹配分词

这种分词算法的基本原理和正向最大匹配分词基本相同，不同的是分词切分的方向与正向最大匹配分词法相反，而且使用的分词辞典也不同。逆向最大匹配法从被处理文档的末端开始匹配扫描，每次取最末端的 2i 个字符（i 字字串）作为匹配字段，若匹配失败，则去掉匹配字段最前面的一个字，继续匹配。相应地，它使用的分词词典是逆序词典，其中的每个词条都将按逆序方式存放。在实际处理时，先将文档进行倒排处理，生成逆序文档。然后，根据逆序词典，对逆序文档用正向最大匹配法处理即可

C. 基于 HMM 的分词算法

隐马尔可夫模型(简称 HMM)是结构最简单的动态贝叶斯网络(dynamic Bayesian network)，这是一种尤其著名的有向图模型。在分词算法中，隐马尔可夫经常用作能够发现新词的算法，通过大量的数据学习，能够将未登录词识别出来，具有广泛的应用场景。其基本步骤是通过大量的数据学习利用 EM 算法去估计 HMM 模型的参数，然后通过动态规划算法去求解最优的序列状态，根据求解的序列状态去划分句子和文本。

D. N 元文法的分词系统

基于词的 N 元语法模型是一个典型的生成式模型，它的基本思想是：首先根据词典对句子进行简单匹配，找出所有可能的词典词，然后，将它们和所有单个字作为结点，构造的 N 元切分词图，图中的结点表示可能的词候选，边表示路径，边上的 N 元概率表示代价，最后利用动态规划算法从图中找到代价最小的路径作为最后的分词结果。需要注意的是，N 元文法强烈依赖于训练语料库。

III. 实验内容

本次实验内容共分为 5 个部分。在第一部分 3.1 中，主要完成了构建词典和划分训练集、测试集的功能。在第二部分 3.2 中，主要实现了最简单的正向最大匹配分词和逆向最大匹配分词。在第三部分 3.3 中，实现了对最大匹配分词的分词效果分析以及两种算法的分词差异比较。在第四部分 3.4 中，基于哈希和字典树实现了对最大匹配分词算法的速度优化。在第五部分中 3.5 中，主要实现了一元文法和基于 HMM 的二元文法分词模型。

3.1 构建词典

首先读取人民日报的 1998 年两个月的数据文件，将数据文件按照互斥的原则划分为训练集和测试集，我们根据每 k 行中取最后一行作为测试集，前 k-1 行作为训练集的方法来划分数据。然后将训练数据和测试数据以文件的形式保存起来，并将编码方式改为 utf-8。为了得到一个好的词典，我们基于以下三个原则来构建词典。

词典大小要合适 太小的词典会极大影响导致算法的分词性能；太大的词典会导致算法计算过程中的查询时间成本变大，使得算法的运行变得低效。一个合适的词典需要平衡这两个因素的影响。

词典要易于理解和阅读 词典应当具有一定的规律（比如按照一定的顺序排序），这样可以方便查验是否存在一些异常错误，也能便于对所构建的词典进行评价。

词典中的词具有唯一性 词典中不应该存在相同的词，重复的词对于匹配分词是没有意义的。

最后根据得到的训练数据按照以上的原则生成一个词典，生成的词典保存在一个文件中。

3.2 最大匹配算法

正向最大匹配分词算法 首先读取测试文件数据集，每次取出一行，按照最大匹配长度取出第一个待匹配的词。然后查询词典，如果在词典中匹配不到该词，则长度减一，继续重复匹配，直到匹配成功或长度为 1；如果匹配成功，则继续处理剩下的分词。直到完成全部分词。算法的伪代码如下表所示

算法：正向最大匹配分词

Require：生成的词典 Words 和最大词长 Max_len

```
输入：测试文件 lines
For  i=0,1…N  do
    line= lines[i]
    ans , maxlen= [], Max_len
    While len(line) >=0  do
        word = line[:maxlen]   #取出第一个分词
        If   word  in Words  or  maxlen==1
                更新 line=line[len(word):]
                恢复 maxlen=Max_len
                ans.append(word)
        else
                maxlen --   #长度减一
    Result.append(ans)
return  Result
```

逆向最大匹配分词算法 逆向最大匹配算法和正向最大匹配分词基本相同，唯一不同的是，逆向最大匹配分词从一行中的最后面开始匹配。

3.3 分词效果分析

通过指标评价分析 一般情况下，我们通常使用准确率 precision、召回率 recall 和 F 值三个指标来评价分词算法的分词性能。通过准确率，我们可以分析在标准分词中被正确分词的情况；通过召回率，我们可以分析测试样本被正确分词的效果；通过 F 值，可以综合评价这两个指标。三个指标的计算公式如下所示

$$\boldsymbol{precision} = \frac{\textit{正确分词数}}{\textit{标准分词数}}$$

$$\boldsymbol{recall} = \frac{\textit{正确分词数}}{\textit{样本分词总数}}$$

$$F = \frac{precision * recall * (k^2 + 1)}{k^2 * precision + recall}$$

然而，仅根据准确率、召回率和 F 值不足以反映出 FMM 和 BMM 算法的分词效果差异特征，为了进一步挖掘出 FMM 和 BMM 的分词效果的特征，我们基于不同的角度提出了以下两种分析方法。

FMM 和 BMM 的泛化能力 我们使用一个比例因子$\alpha \in [0,1]$，来控制从 199801_seg&pos 中抽取数据作为训练集的比例，此时训练集可以为表示为等式(1)。为了方便，我们定义了 FMM 和 BMM 之间的性能差异 G 为准确率、召回率和 F 值的平方和（等式（2））

$$D_{train} = D_{199802} + \alpha * D_{199801} \quad (1)$$

$$G = (P_{FMM} - P_{BMM})^2 + (R_{FMM} - R_{BMM})^2 + (F_{FMM} - F_{FMM})^2 \quad (2)$$

我们可以看到调节α其实可以控制 FMM 和 BMM 对于分词的泛化能力，当$\alpha \to 0$的时候，相当于使用 199802 的数据来构建词典，然后测试 199801_sent 的数据，避免了直接使用同一份数据进行构建词典和测试算法，带来的较高的分词性能。最后我们跟踪 G 随α的变化来观察 FMM 和 BMM 分词的在不同程度泛化后的分词性能规律。

FMM 和 BMM 的出错词性分析 为了进一步挖掘出 FMM 和 BMM 的微小差异，我们通过词性的视角来看待二者的区别。我们通过对比标准分词文件和算法生成的分词文件，统计在标准文件中没有正确划分出来的词的词性。我们可以按照以下三个步骤进行这个过程：首先抽取出错误的分词；然后统计错误分词中的词性频率，并对其排序；最后，我们对比 FMM 和 BMM 的错误词性频率分布图，并从中探究二者的分词差异所在。

3.4 算法优化

在 3.2 中，我们使用最简单的数据结构 list 来存储词典并通过简单的语句来查询匹配是否成功，由此带来的巨大的时间计算成本。在这一小节中，我们从缩减搜索空间和减少匹配次数来降低时间复杂性，并通过一些辅助的数据结构来实现这一目标。

Trie 树 这种数据结构的核心的思想是利用空间换时间，利用字符串的公共前缀来限定部分单词在一颗子树上。Trie 树能够以更高效的方式进行查询和插入节点。

Hash 哈希通过一个散列函数将被查找的数据映射为哈希表的索引，不同的数据可能发生冲突映射到同一区域，因此这种数据结构的效率取决于所选取的散列函数和冲突处理。

我们首先自主设计了一个 Trie 树，并通过这个 Trie 树来存储词典中的单词。此外，我们通过一个哈希表来实现对 Trie 树中节点的所有子节点的存储。我们将哈希函数表示为字符的 unicode 码除以哈希表的长度，实验表明这个哈希函数可以很好地减少字符的匹配次数。而通过一个主体的 Trie 树，我们可以很好地利用前缀信息来减少搜索空间，将搜索空间限定在特定的子树上。

3.5 HMM

HMM 模型基于齐次马尔科夫假设和观测独立假设而建立，认为任意时刻的状态只与前一时刻的状态有关，而与其它时刻状态和观测无关，而观测状态只与当前时刻的隐状态有关。

将 HMM 模型应用于中文分词首先需要对训练文本进行词位标注，将每个词的位置状态作为隐变量。然后基于统计的方法，估计状态转移矩阵 A、观测概率矩阵 B 以及状态初始概率 Pi。最后根据所估计的参数，使用动态规划去求解出最优的观测序列。HMM 进行分词的基本过程如图 3.5.1。

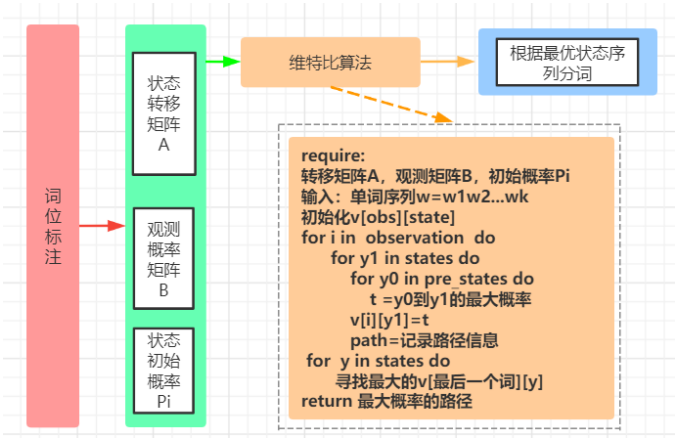

图 3.5.1

3.6 N 元文法分词系统

3.6.1 一元文法

根据一元文法的前提假设，我们可以认为每个词的概率都是独立不相关的。因此对于一个输入语句（文本）w，对应于一个分词序列$w_0 w_1 \dots w_k$的概率$P(w)$可用下面的等式(3)来描述

$$P(w) = P(w_0 w_1 \dots w_k) \quad (3)$$

$$= P(w_0) * P(w_1) \dots P(w_k)$$

我们可以看到一个句子对于一个分词序列概率等于该分词序列中的各个分词的概率累乘。我们基于统计的方法来得到每个分词的概率，即统计每个分词出现的频率来估计来分词的概率。由于分词概率较低，最后结果可能导致溢出的问题，我们采取等式两边取对数的方式来避免该问题，即是将等式（3）修改为等式（4）

$$log(P(w)) = log(P(w_0 w_1 \dots w_k)) \quad (4)$$
$$= log(P(w_0)) + \cdots + log(P(w_k))$$

最后，我们采取极大似然估计法去估计分词序列的后验概率，选择使得句子概率最大的分词序列作为分词结果（等式（5））。

$$w = \arg\max_{w0\dots wk} log(P(w_0 w_1 \dots w_k)) \quad (5)$$

对于如何求解等式（5），我们先将原问题转化为一个有向无环图，然后可以采用动态规划的方法去求解最大概率路径。

值得注意的是，我们有可能遇到词典中不存在的词，这会导致整个句子的累乘概率为 0，取 log 之后变为负无穷大。我们可以在计算分词概率的 log 函数里面加上一个常数作为平滑处理，也可以将每一个词的概率估计通过下面的等式（6）来做平滑处理。

$$P(w_i) = \lambda P_{ML}(w_i) + (1-\lambda) * C \quad (6)$$

3.6.2 二元文法+HMM

二元文法认为每个单词的概率只和前一个单词有关，所以对于一个给定的待分词句子 w，对应于一个分词序列的概率$P(w)$可以描述为等式（7）。然后我们可以根据极大似然估计法去估计分词序列的后验概率，选择使得句子概率最大的分词序列作为分词结果（等式（8））

同样地，我们可以使用等式（9）来对每个分词概率做平滑处理。

$$P(w) = P(w_0 w_1 \dots w_k) = \prod_t^k P(w_t | w_{t-1}) \quad (7)$$

$$w = \arg\max_{w0\dots wk} log(\prod_t^k P(w_t | w_{t-1})) \quad (8)$$

$$P(w_i|w_{i-1}) = \lambda P_{ML}(w_i|w_{i-1}) + (1-\lambda) * P(w_i) \quad (9)$$

最后，我们可以通过动态规划的方法求解出最大概率的分词序列进行分词。至此，我们便完成了所有二元文法的内容。但是为了进一步提高二元文法的分词性能，我们在每次二元文法分词结束后，将所得的分词序列传入到 HMM 模型中来对未登录词进行识别，这个过程可以看作是对二元文法的后处理。此外，我们还可以对数据进行一些清洗和处理，以消除噪音数据的影响。这个过程可以看作是对二元文法的预处理。

IV 实验结果分析

在 4.1 节中，我们主要对所构建的字典进行了分析。在 4.2 节中，我们主要对正向最大匹配算法和逆向最大匹配算法的分词结果进行了性能分析和差异分析。在 4.3 节中，我们对优化后的分词算法进行了分析，证实了优化方法的高效性。最后，在 4.4 节中，我们主要分析了基于统计分词算法的分词效果。

4.1 词典分析

词典可以看作是词的集合，在不考虑词意的情况下，我们要求词典中的词具有唯一性。基于最大匹配算法的分词效果很大程度上依赖于所构建的词典，因此，一个好的词典往往能给分词效果带来显著的提升。为了提高分词性能，同时兼顾词典的可读性，我们构建的词典具有以下特性。

唯一性 词典中的每一个词具有唯一性，每个词只会出现一次。避免了重复的词出现多次，影响词典的大小和查询性能。

有序性和易读性 词典中的词按照一定的偏码顺序进行排序，首字母相同的词放在一起，方便查询和阅读。

选择性和针对性 针对不同的情况和分词任务下，我们设计了不同的词典，如图 4.1.1。其中 dic 词典用于最大匹配分词，为了压缩词典，不将量词加入；uni_dic 词典用于一元文法分词，记录每个词的出现的频率；bi_dic 词典用于二元文法分词，记录每个词相对于前一个词的频率。

空间高效性 不考虑将量词加入，词典的空间大小压缩为原来的 60%，减少了 40%的量词的空间开销。

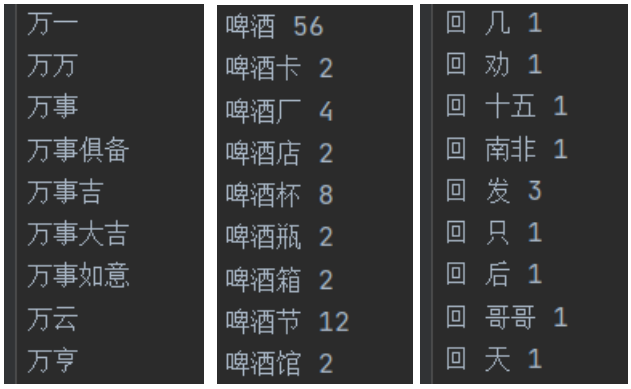

图 4.1.1（左：dic; 中：uni_dic; 右：bi_dic）

我们对比了将量词加入词典和没有将量词加入词典对分词性能的影响，主要考量了词典的大小（表征词典的空间性能）、分词正确率、召回率、F 值四个指标，结果如下表。

	词典大小	正确率	召回率	F 值
没加量词	69877	94.62	92.25	93.45
加入量词	112789	96.3	96.92	96.63

去掉量词之后，词典大约减少 40%，压缩率为 0.6。而应用 FMM 分词的正确率、召回率和 F 值大约下降 3%。权衡考虑词典复杂性和分词性能之后，默认情况下，不将量词加入词典，以提高词典的空间性能。

4.2 最大匹配分词结果分析

我们将 199801 和 199802 这两个月的数据作为训练集，使用的 199801_sent.txt 作为测试集，分别应用 FMM 和 BMM 对其进行分词，分词效果如下表 1

	准确率	召回率	F 值
FMM	94.52	92.43	93.46
BMM	94.91	92.77	93.87

表 1

从表中可以看出，BMM 比 FMM 的分词效果稍微略好，但基本接近。

为了挖掘出 BMM 和 FMM 对于泛化能力的差异。在表 2 中，我们给出了 FMM 和 BMM 之间的性能差异 G 随α的变化。我们可以看到，随着α的不断变小，FMM 和 BMM 的 G 值也在不断变小，也就是说，二者基本具有相同的泛化能力，这算是 FMM 和 BMM 共同点。

α	0.5	0.33	0.25	0.125	0
G	0.2541	0.2421	0.1944	0.1658	0.147

表 2

为了探究出 FMM 和 BMM 的分词差异。我们站在词性的角度去分析，并且成功地找到了 FMM 和 BMM 的分词差异所在。

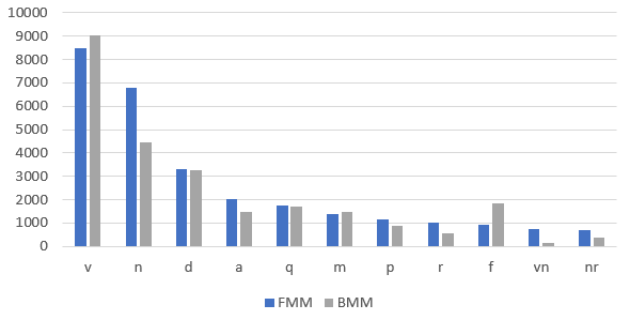

图 4.2.1

由于当α越大的时候，FMM 和 BMM 之间的 G 越大，即差异越大。所以我们令α=1，即是使用 199801 和 199802 作为训练集来构建词典以获得最大差异。为了消除量词的影响，我们在词典构建中选择加入量词，然后应用 FMM 和 BMM 对 199801_sent 进行分词。然后我们比较分词所得的结果和标准分词结果，统计出现错误的词性的频率（以 FMM 为基准，取出错误词性频率大于 500 的结果），结果如图 4.2.1。我们可以直观地看到，导致 BMM 比 FMM 分词性能稍好的主要原因为 BMM 对名词的分词结果比 FMM 要好，尽管 FMM 在动词方面比 BMM 分词要好一些。

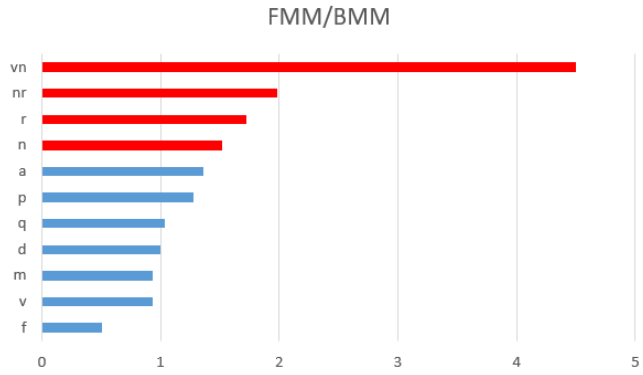

图 4.2.2

有趣的是，当我们计算 FMM 和 BMM 各个词性的出错频率之比时（FMM/BMM）（如图 4.2.2）。我们可以发现 FMM 分词错误对动名词(vn)，人名（nr）、名词（n）敏感性较强（相比 BMM），特别是动名词，FMM 的出错率是 BMM 的四倍以上。尽管 FMM 在介词（f）的方面分词效果较好，但是在一般的文本中，名词所占的比重往往要远高于介词所占的比重，这也是为什么 FMM 分词结果比 BMM 稍差的原因所在。

4.3 优化后效果分析

我们使用 199801_seg&pos.txt 的全部数据作为构建词典的来源，然后将词典里面的分词存储在 Trie 树中，并结合哈希技术将词映射到合适的位置，然后调用最大匹配分词算法对测试集进行分词。在这里，我们选取全部的 199801_sent 作为测试集，计算优化后的运行时间。但是，由于优化前的最大匹配算法测试完全部的数据需要较长的时间，我们使用抽样的方法进行测试，每 k 行抽取一行数据，相当于计算测试 1/k 的数据所需的时间，最后乘以 k 得到总的估计时间，并以此来估算优化前的所需的运行时间。为了尽可能降低 k 的主观取值所带来的误差，我们计算了不同的 k 值的估算时间，最后取平均作为实际估算时间(单位是小时)，如表 3 所示。

k	8	12	16	平均值
FMM/hour	4.84	4.99	5.09	4.97
BMM/hour	4.61	4.72	4.79	4.70

表 3

	优化前时间/小时	优化后时间/秒
FMM	4.97	8.89
BMM	4.70	7.27

表 4

在表 4 中，我们可以看到，使用 Trie 树优化后，算法的时间复杂度得到了极大的改进，程序运行时间从 5 小时左右降到 8 秒左右，证明了使用 Trie 树+哈希的优化方案是

可行并且是相当高效的。此外，我们还能发现 BMM 的时间性能总是比 FMM 好一些。

4.4 基于统计的分词效果分析

4.4.1 一元文法分词测试

我们遵循互斥的原则来划分训练集和测试集，从人民日报 1998 年 1 月份的数据中抽取 1/k 作为测试集，剩下的数据加上 1998 年 2 月份的全部数据作为整个训练集。我们对一元文法中的每个分词的概率估计做平滑处理，在原来的基础上加上一个常数 C=1/N，避免取 log 后出现负无穷的情况。最后，我们取不同的 k 值，降低主观性带来的误差，得到一元文法的分词性能如表 5 所示。

从表 5 中，我们可以得到一元文法的准确率基本维持在 94.1%左右，召回率在 70.73%左右，F 值在 80.77%左右，基本不受 k 值的影响。值得注意的是，一元文法虽然准确率较高，但召回率和 F 值偏低，说明一元文法对于我的测试样本效果欠佳，将错误的分词分为了多个短词，导致我的样本数据分词总数增大。

k	1	3	5	8	平均值
准确率	93.08%	94.46%	94.55%	94.44%	94.1%
召回率	69.01%	71.29%	71.14%	71.49%	70.73%
F 值	79.26%	81.25%	81.19%	81.37%	80.77%

表 5

k	1	3	5	8	平均值
准确率	92.76%	94.18%	94.27%	94.15%	93.84%
召回率	68.84%	71.15%	71.01%	71.35%	70.58%
F 值	79.03%	81.06%	81.00%	81.18%	80.56%

表 6

4.4.2 二元文法分词测试

我们基于和一元文法相同的原则划分数据集和测试集，然后使用单纯的二元文法（没有平滑处理）进行分词测试，测试结果如表 6 所示。

对比表 6 和表 5，我们看到一元文法和（没有平滑处理）二元文法的分词结果基本接近，无明显差别。对于造成这个结果的成因，我们首先探究了平滑因子对结果的影响。首先固定 k=3，然后调整二元文法中对分词概率估计的平滑因子λ，对λ取一系列不同的值，最后分别测试其分词效果，结果如表 7 所示。我们可以看到，平滑处理并不能有效地解决召回率和 F 值偏低的问题。

需要说明的是，前面的数据均是在没有加入量词的默认情况下得到的。然而，有趣的是，当我们在训练集中加入量词时，此时固定二元文法的平滑因子λ =0.5 时，得到一元文法和二元文法的分词效果如表 8 所示。可以看到，一元文法的分词性能并没有任何变化，然而对于二元文法，其召回率和 F 值却得到了较大的提升。于是，在这里，我可以认为造成召回率较低的原因与量词有关，而二元文法优于一元文法在于对量词的处理能力。但需要注意的是，这并不是根本原因，这只是一个粗略的结论。对于召回率低的真正原因，将在 4.4.3 中继续讨论并给出有效的解决方法。

λ	0.3	0.5	0.7	0.9	平均值
准确率	94.32%	94.29%	94.27%	94.24%	94.28%
召回率	71.24%	71.23%	71.22%	71.20%	71.22%
F 值	81.17%	81.15%	81.14%	81.12%	81.14%

表 7

	k	1	3	5	8
准确率	1-gram	93.08%	94.46%	94.55%	94.44%
	2-gram	92.87%	94.25%	94.34%	94.24%
召回率	1-gram	69.01%	71.29%	71.14%	71.49%
	2-gram	81.92%	84.49%	85.03%	85.01%
F 值	1-gram	79.26%	81.25%	81.19%	81.37%
	2-gram	87.05%	89.33%	89.44%	89.38%

表 8

k	1	3	5	8	平均值
准确率	94.48%	95.89%	96.01%	95.84%	95.55%
召回率	90.89%	94.29%	94.72%	94.29%	93.54%
F 值	92.65%	95.09%	95.38%	95.06%	94.55%

表 9

4.4.3 二元文法+HMM 分词测试

由于量词的加入能够使得二元文法改进召回率，通过观察分词结果，发现二元文法对于时间量词的划分容易出错。于是，我们猜想未登录词才是影响召回率偏低的真正原因。为此，我们尝试加入 HMM 作为二元文法的后处理过程，以此来增强二元文法对未登录词的识别能力，同时不将量词和标点符号加入到二元文法的处理过程中，以提高二元文法的性能，因为这些部分将由 HMM 来处理。我们固定平滑因子λ=0.5，对 k 取一系列不同的值，得到二元文法+HMM 的分词效果，如表 9 所示。

对比表 9 和表 7，可以看到加入 HMM 模型之后，各项评价指标均得到了不同程度的改进，特别是召回率和 F 值得到了显著的提升。于是，我们认为，造成之前二元文法召回率偏低的真正原因是不能有效地识别未登录词。我们显然可以通过加入 HMM 来改进这一点。

V 结论

本次实验内容覆盖了基于词典的匹配分词和基于统计的分词模型，对于一些基本的分词算法和模型均有涉及。同时，我们也基于字典树和哈希技术对最大匹配分词算法进行了速度优化，显著降低了算法的运行时间。值得特别注意的是，本次实验不仅仅只是重复了前人的工作，我们基于词性的视角成功观察到了正向最大匹配算法和逆向最大匹配算法的微小差异，对于全面认识最大匹配算法有了新的理解。同时我们也找到了基于文法的分词模型出现召回率偏低的原因以及一元文法和二元文法对于量词的处理差异，并加入 HMM 模型提高了二元文法的分词召回率。

VI 参考文献

[1]宗成庆. 统计自然语言处理[M]. 北京: 清华大学出版社，2013
[2] 利用前缀树实现最大匹配分词算法 | DC's Blog (gitee.io)
[3] https://blog.csdn.net/SLP_L/article/details/112917666
[4] https://zhuanlan.zhihu.com/p/126233051