汉语分词系统

哈尔滨工业大学

刘璟烁

摘要

实验首先给出对汉语分词词典的三种建立方式,在不同的模型中对词典这一要素进行了很好的时空复杂夫度权衡;实现了前后向最大匹配分词模型,并在此基础上利用 trie 树及 hash 表结合的数据结构优化存储使得时间性能得到了巨大提升;本次实验同时尝试了几种概率分词模型的实现,得到了令人满意的分词效果提升.

1 引言

分词是将连续的字序列按照一定的规范重新组合成词序列的过程.而词是最小的能够独立活动的有意义的语言成分[1].在汉语中, 词与词之间不存在分隔符, 词本身也缺乏明显的形态标记, 因此,中文信息处理的特有问题就是如何将汉语的字串分割为合理的词语序列, 即汉语分词.汉语分词是句法分析等深层处理的基础, 也是机器翻译、信息检索和信息抽取等应用的重要环节.

国内外的研究者在汉语分词方面提出了很多有效的算法.我们可以粗略地将这些方法分为两大类:第 1 类是基于语言学的规则方法, 如:各种形态的最大匹配、最少切分方法、以及综合了最大匹配和最少切分的 N-最短路径方法[2];第 2 类是基于大规模语料库的机器学习方法,这是目前应用比较广泛、效果较好的解决方案.用到的统计模型有 N 元语言模型、信道-噪声模型、最大期望、隐马模型等.在实际的分词系统中, 往往是规则与统计等多类方法的综合.一方面, 规则方法结合使用频率, 形成了可训练的规则方法; 另一方面, 统计方法往往会自觉不自觉地采用一些规则排除歧义、识别数词、时间及其他未登录词.

汉语分词的主要瓶颈是切分排歧和未登录词识别.切分歧义和未登录词降低了自身正确切分的可能性, 同时还干扰了其相邻词的正确处理.并且, 未登录词往往和切分歧义交织在一起, 进一步增加了分词的难度.

2 实验说明

2.1 文件说明

除了实验给定两个文件“199801_sent.txt”和“199801seg&pos.txt”的编码方式为 **gbk**，本次实验中其余的所有文本文件均为 **utf-8** 的编码方式。

压缩包中含有 **Scores** 文件夹、**TrainFiles** 文件夹、**TestFiles** 文件架以及 **Unidics**、**Bidics**、**Uniresults**、**Biresults** 文件夹，以及程序源代码。先对其说明如下:

- **Scores** 文件架中含有第三部分要求生成的 **score.txt** 文件，以及后续介绍的十折交叉验证测得的一元文法的性能 uni_score.txt 以及二元文法的性能 bi_score.txt。
- **TimeCost.txt**，为实验第四部分要生成时间对比结果。
- **dic.txt**，为实验的一部分由第一个训练集 trian_1.txt 生成的词典

2.2 实验审查说明

- **第一部分**，请打开 P3_1.py 到最底端，取消带有客户端标记下的三行注释，程序将生成对应结果文件。
- **第二部分**，请打开 P3_2.py 到最底端，取消带有客户端标记下的三行注释，程序将生成对应结果文件。注意到这里是未进行时间优化的模型，在本机上生成文件大约需要六小时，时间较长，但保证结果的正确性。
- **第三部分**，请打开 P3_3.py 文件到最底端，取消带有客户端标记下的一行注释，程序将生成对应结果文件 score.txt。
- **第四部分**，请打开 P3_4.py 文件到最底端，取消带有客户端标记下的一行注释，程序将生成对应结果文件 TimeCost.txt。
- **一元文法实现**，请打开 P3_5_1.py，取消最后客户端后一行（仅一行）的注释，进行十折交叉验证，十次分词结果将写入到 Uniresults 文件夹中，性能见 Scores 文件夹中。
- **二元文法实现**，请打开 P3_5_2.py，取消最后一行的注释，进行十折交叉验证，十次分词结果将写入到 Biresults 文件夹中，性能见 Scores 文件夹中。
- 最后一部分性能测试，请打开文件 P3_5_1.py，根据输入提示进行输入，结果 seg_LM.txt 将在当前目录下。

3 实验内容

- 为后续任务构建分词词典，根据分词模型的特点，我们利用十折交叉验证法分别构建了三个分词词典，在满足模型需求的条件下降低了时间和空间复杂度。
- 实现正反向最大匹配的分词模型。以 python 为编程语言，使用了原有的 list 作为数据结构，在代码量尽量少的情况下给出实现方法。建立在 list 数据结构上的元素查找和匹配所需时间与列表中元素个数呈线性关系，因此该实现方式的时间复杂度较高。
- 对于前面实现的机械匹配分词模型，利用准确率、召回率、F 值三个指标分别观察正反向最大匹配模型对测试集的分词效果，并对效果进行了分析。通过上述三个指标，我们可以看到在对特殊格式的内容（比如连续数字、英文等）进行处理后，我实现的前后向最大匹配算法在给定测试集上达到了三个指标的平均值均超过百分之九十的良好效果。通过后面的分析我们还可以看到，后向最大匹配在中文分词的性能上要略优于前向最大匹配模型。
- 通过设计并实现存储词典的数据结构，在极小程度提高空间复杂度的基础上，对第二部分实现的机械匹配分词模型在时间复杂度方面进行了优化，使得程序运行速度的提升达到了数千倍，从而极大缩短了运行时间，提高综合效率。
- **二元文法实现**通过统计模型实现分词功能，该部分内容包括实现一元文法以及更进一步的二元文法分词模型。基于统计的分词模型在训练数据量足够的情况下具有相对于机械匹配分词模型更好的分词效果，结合后面的内容我们可以看到，在给定的十个测试样本集上，一元文法分词模型的三个指标相对于上述两种机械匹配分词模型均有平均接近两个百分点的较大提升。通过选取良好的平滑方法，比如基于隐马尔可夫的平滑算法，可以使 n 元统计分词模型的分词效果进一步优化

4 实验过程

4.1 词典构建分析

现有的汉语自动分词系统大都是先基于词典进行匹配分词[3],再利用句法语义关系和统计方法进行歧义处理和未登录词处理。分词词典机制的优劣直接影响到分词系统的速度和效率,因而建立高效快速的分词词典机制势

在必行。从词典实用性的角度来看，构建的分词词典应根据分词模型类别的不同而有所不同，具体的，构建出的词典应该有以下几个特点：

- 对空间和准确率进行权衡.词典的内容往往对分词结果有着直接的影响，增大词典规模理论上可以提升分词效果，但通过实验给定的训练样本（即 1998 年 1 月人民日报语料）我们可以看到一些类似连续数字构成的日期标记、网站网址的词放入词典对其泛化能力提升收效甚微，但这些内容却大大消耗了词典所需要的存储空间，并且在后续词匹配任务中也会导致由于词典规模较大带来的匹配时间剧增，因而降低了词典在分词任务时间和空间方面的性能。通过对后续分词任务的结果分析可以发现，去除词典中的连续中英文字符串对分词效果影响十分微小，并且在分词任务中遇到响应字符串可以通过基于规则的处理方法保证分词的性能，因此在构建词典时不加入连续英文、数字以及符号构成的字符串，从而可以大大降低词典所需要的空间存储。

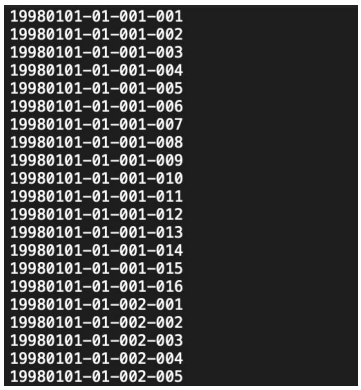

图 4.1.1 充满特殊标记的词典

- 词典应该服务于分词任务需求.以本次实验为例,对于机械匹配分词模型而言，并不需要统计词的词频，因此只需要将词本身放入词典，从而在保证分词效果不受影响的情况下减小空间上的开销；而对于后续的统计分词模型而言，由于模型的需求我们需要记录每一个词的词频，甚至在二元文法中我们还可以构建含有相对词频的词典从而提高分词任务的效率。

- 应对词典进行适当的组织.通过对词典中的词进行前缀或者后缀排序，将有利于后续分词任务的时间性能提升。比如在执行分词任务读取词典时，如果选择将词典存储于连续的内存空间之中，排好序的词典可以加快二分查找等匹配方法的时间性能。另外，合理的词典格式（有无空行、每行词数）可以减小读取时的难度，降低出错的概率。

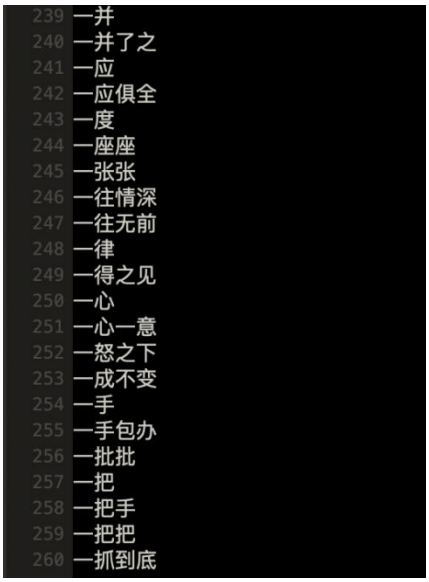

图 4.1.2 合理组织的词典

- 对于给定任务而言，词典应具有公平性.特定的模型分词性能评比任务要求用给定的训练集生成词典，肆意的扩大词典容量将有违对模型性能的客观分析。因此词典应该是基于特定训练集所生成的。

4.2 正反向最大匹配分词模型

正向最大匹配和反向最大匹配算法均属于机械匹配分词模型，其按照一定的策略将待分析的汉字串与一个“充分大”的机器词典中的词条进行匹配，若在词典中找到对应字符串，则匹配成功（即识别出一个词）。

- **正向最大匹配分词算法**

正向最大匹配算法的逻辑是，在读取分词词典后，首先记录分词词典中的最长词条所含汉字个数 len，随后取被处理文本当前字符串序数中的前 len 个字作为匹配字段，查找分词词典。若词典中有这样的一个词条，则匹配成功，匹配字段作为一个词被切分出来，重新对字符串剩余的部分进行相同操作；否则匹配失败，将匹配字段去掉最后一个汉字，重新尝试匹配分词词典中的词条，直到切分成功为止。

- **反向最大匹配分词算法**

反向最大匹配分词算法的逻辑与正向最大匹配思想相近，其不同点在于每次从待处理文本的后 len 个字开始进行匹配处理，len 仍然为读取的分词词典中最长词条所含的汉字个数。则匹配成功，匹配字段作为一个词被切分出来，并对字符串剩余的部分进行相同操作；如果匹配失败，将匹配字段去掉最前面一个汉字，重新尝试匹配直到切分结束。

- **实验过程**

结合实验指导书，在实现正反向最大匹配分词的基础上，我对算法实现进行了以下几部分处理。

- **使用十折交叉验证生成分词词典**。应用十折交叉验证的思想，对于给定文件“199801_seg&pos.txt ”使用十折交叉验证方法生成十个训练集及对应的互补测试集，先前我们选用第一个训练集生成词典 dic.txt，在这里使用该词典进行前向最大匹配。

- **应用 list 数据结构**。根据实验指导书，该部分的实现使用了 python 内置的 list 结果存储读取的词典到变量 dic 中，并在匹配过程利用操作符 in 来判断给定的匹配字段是否为分词词典中的词条。

- **13 行代码实现 FMM 及 BMM**。通过使用内置数据结构 list 以及三重循环遍历（见附件 P3_2.py），完成对正/反向最大匹配的实现以及文件的读写操作。值得注意的是，我使用额外的一行代码用于记录总的行数，从而达到了写出的分词文本行数与原文本完全一致而没有结尾的冗余空行。

- 处**理非中文字符串**。文章 4.1 部分提到词典构建的过程中并未加入由数字、英文及其他符号构成的特殊字符串，这是因为其不仅不能很好的提高在其他文本上分词的效果，而且对存储空间有大量的消耗。在这里我们将在实现 FMM 及 BMM 的基础上，可以对未知给定文本上的特殊字符串进行处理，其思想是即识别被切分为单个字的连续序列，识别其中的特殊字符串部分并组合为一个词，函数实现请见附件(Handle_number.py)

4.3 正反向最大匹配分词效果分析

4.3.1 实现

根据实验指导书要求，该部分需要利用准确率、召回率以及 F 值三个指标，对上一部分两个分词模型生成的文件进行效果分析，并给定文件“199801_seg&pos.txt”作为标准切分文件。

三个指标的计算方法如下:

$$\text{Precision} = \frac{\text{正确的分词数}}{\text{分词结果的总分词数}} \quad (1)$$

$$\text{Recall} = \frac{\text{正确的分词数}}{\text{标准分词数}} \quad (2)$$

$$F = \frac{(\alpha^2+1)*\text{Precision}*\text{Recall}}{\alpha^2*(\text{Precision} + \text{Recall})} \quad (3)$$

基于上述三个指标，我在附件（P3_2.py）中实现了 calc_score()方法用于计算分词结果的准确率、召回率以及 F 值，根据工程经验，F 值公式中的变量α常取为 1。

·**方法的实现思想**。对于每一对对应的非空行，从左到右匹配相同的分词并统计总数。具体的，先将两行文本进行相应切分得到两行文本中词的个数，并累加到分词结果总切分数及标准总切分数中。随后利用两个指针从左到右按照被切分词的长度变化，当两个指针连续两次一一对应时可以判断这两次指针值之前的片段是匹配

的，从而正确切分数加一。对于整个文本，如此处理所有行并统计出总的切分数、标准分词数、正确分词数，代入上述三个公式分别计算 Precision、Recall、F 即可.

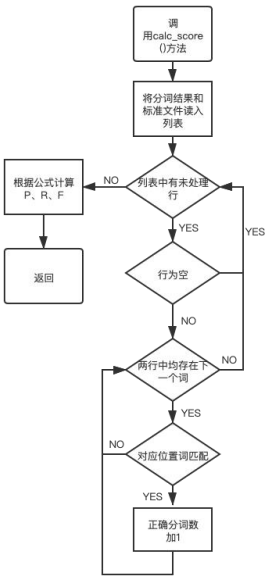

图4.3.1 方法流程图

4.3.2 两种模型精度差异分析

给定标准切分文件“199801_seg&pos.txt”，调用上一部分实现的方法对 4.2 节 FMM 及 BMM 切分结果进行性能分析结果如下：

```
FMM
 准确率:  97.14827761345643%
 召回率:  96.89990016924774%
 F值: 97.02392993249069%

BMM
 准确率:  97.2798918845773%
 召回率:  97.03365058886119%
 F值: 97.15661521368827%
```

图 4.3.2：分词性能结果

经过对比可以发现，在该任务中 FMM 的切分性能略差于 BMM。

- **BMM 结果整体优于 FMM 切分结果结果**，仍然利用十折交叉验证的方法，我们分别在十个模 10 生成的训练样本中生成词典并分别利用 FMM 和 BMM 在对应测试集(十分之一的)进行分词任务，经过性能计算得到十次性能的平均值如图 4.3.3，可以看到上述部分在“199801_sent.txt”上的结果并非偶然，BMM 切分的性能要整体 FMM 模型.(注意，此时分析使用的标准文件为各个测试集对应的标准文件而非全部标准文件“199801_sent.txt”,因此三大性能指标有所下降).

```
FMM
 准确率:  91.92452101783475%
 召回率:  94.52930092815092%
 F值: 93.20865113934957%

BMM
 准确率:  92.10647058190526%
 召回率:  94.70241689746786%
 F值: 93.3863437963407%
```

图 4.3.3 分词性能差异

- **中文语言习惯造成二者消歧能力差异**，给定一个分词词典，FMM 和 BMM 切分算法分别从待处理文本的前端和后端对文本中的片段进行词匹配，而在某一个片段附近可能产生“AB C”或者“A BC”的交集型歧义，FMM 与 BMM 会因此切分出不同的结果。而根据中文的语言特点可以得知，短语或语句的主要内容倾向于放置在后半部分内容，因此对于中文而言 BMM 具有了在消除交集型歧义方面的天生优势。对比下图实验文本中的“赵家父子”，由于“家父”一词在词典中，导致 FMM 无法很好的消除歧义，将其切分为“赵/家父/子”，但中文语句信息的重心偏处于后半部分，BMM 将其切分为“赵家/父子”，很好的消除了交集型歧义，这是 BMM 算法在中文分词任务中的优势所在。

```
19980129-01-004-007/ 李/ 鹏/ 和/ 赵/ 家父/ 子/ 围/ 坐/ 在/ 一起/
19980129-01-004-007/ 李/ 鹏/ 和/ 赵/ 家/ 父子/ 围/ 坐/ 在/ 一起/
```

图4.3.4 BMM 及 FMM 分词对比

4.4 基于机械匹配分词系统的速度优化

该部分内容要求通过利用自行实现的查找算法和数据结构，对 4.2 节的机械匹配分词算法实现速度优化。

4.4.1 优化思想

通过减少在数据结构中查找词的所需匹配次数从而减小时间复杂度。通过利用 trie 树存储词典，可以在常数时间内完成对需要匹配字段的查找，这里的常数时间与词典中的最大词长相关。而 4.2 节使用的 list 数据结构为一种线性表结构，其用一块连续的内存空间，来存储一组具有相同类型的数据。若想要通过原有的实现方式，利用 in 判断一个元素是否在列表内，根据底层函数 list_contains()的代码逻辑，需要从头遍历列表，最好情况是第一个元素就成功，最坏情况是最后一个元素匹配成功，或者元素根本就不在列表中，这两种情况均会完整遍历列表元素，因此原有的模型在查找一个字段是否在词典中时，需要耗费 O(n)的时间复杂度，其中 n 为列表长度。

4.4.2 数据结构及优化算法实现

总体上，通过利用 trie 树作为主要结构，结合 hash 表减少时间开销，从而形成优化后的数据结构，该部分我实现了 FMM 以及 BMM 两者的优化版本。

·类 TriNode 实现 trie 树的节点类型，每一个 trie 树的节点中包含了四个属性，分别为：由父节点到该节点路径上对应的字符 char，该节点的子节点数 child_num,存储子节点的列表 child_list 以及布尔变量 is_word：由根节点到该节点的路径上是否形成了一个词。TriNode 类实现了向节点插入子节点以及查找节点功能，每个子节点应用了 hash 函数结合线性探测计算其对应的在 child_list 中的索引值，并且在子节点列表已满时扩充列表大小为原来的二倍。

·TrieDic 为 trie 树的实现类，该类通过 init 方法创建并初始化一个根节点，并将给定词典中所有的词依次插入到该根根节点上形成一棵 trie 树。

·优化后的 FMM 实现思想：给定词典构建的 trie 树，对待分词文本的每一行，从第一个字开始在 trie 树中进行匹配，若有对应的子节点则继续在该节点的子节点中对下一个字进行匹配，直到将该行文本匹配完或者没有对应的子节点为止。匹配过程中通过判断节点属性记录直到匹配到所有前缀词，最后结束时选择最长的前缀词作为切分出的词，并对该行剩余部分继续进行上述处理直到该行被切分完成。

注意，BMM 分词模型的优化思想与 FMM 基本相同，少数的区别在于可以通过在 trie 树中倒序存储词典中词并在词首字母所在节点标记为词的方法适应 BMM 的搜索方法，在对词进行匹配的过程中对每一行文本从后向前逐字在 trie 树中进行前缀匹配即可。

4.4.3 优化结果分析

通过已实现的方法 time_compare()对两个模型优化前后所需的执行时间进行计算并写入到文件 TimeCost.txt 文件中，得到结果如下

```
FMM:
    时间为23086.262369155884s
BMM:
    时间为22455.278384685516s

优化FMM:
    时间为3.6020150184631348s
优化BMM:
    时间为3.8570430278778076s
```

图 4.4.1 模型优化前后分词时间对比

可以看到，给定文件“199801_sent.txt”，模型在优化后对文本的分词速度提升达到了数千倍。这主要得益于单词的查找时间由 O（n）降到了常数时间。

4.4.4 进一步优化的关键

由上述部分可知，通过减小查找对应字段是否位于词典之中的复杂度可以大幅提高分词的时间效率。另外，对于给定的两个模型（即 FMM 和 BMM），还可以通过减少用于查找的字段数量来减小时间损耗，进而优化分词速度。本次实验使用了 trie 树作为存储词典的主体结构，而鉴于对空间开销的考虑，对于每一个节点的子节点均使用 hash 表的方式进行存储。为了实现进一步的优化，可以从以下两个方面对该数据结构进行改动：首先可以选择更优的 hash 函数，使得子节点在散列表中更为分散，从而加快对某一个节点的子节点散列的查找速度，或者可以简单的扩大调整每个节点的子节点散列规模从而分散子节点加快查找速度，即进一步用空间换时间。另一方面，可以将 trie 树与 hash 结合的方式更改为双数组 trie 树的结构，通过构建 base 和 check 两个数组，利用 trie 树实现中所浪费的大量空闲空间，并能够在消耗更少量空间的情况下达到比上述 hash 匹配更快的查找速度。

4.5 基于统计语言模型分词系统实现

自然语言是一种上下文相关的信息表达方式，统计语言模型就是为自然语言这种上下文相关的特性建立的数学模型。给定一个一个足够大的语料库，统计每个词的频率（利用极大似然估计的思想，它近似等于概率），以及任意一组词的频率。从而计算出任意一个句子出现的概率，对句子进行分词。

这一部分就利用了动态规划算法，从给定的文本文件生成了统计语言模型，包括一元文法分词模型（请见实现于附件 P3_5_1.py ）、以及二元文法模型（P3_5_2.py），并实现了未登录次识别功能。

对于统计语言模型的性能分析仍然使用上述的十折交叉验证方法，通过对十次分词得到 score 的平均值分析模型的性能。

4.5.1 一元文法模型

一元文法模型是 n-gram 模型的一个特例，它认为句中词的概率满足朴素贝叶斯条件，即：

$$P(w_1,w_2,...w_n) = P(w_1)*P(w_2)*...*P(w_n) \quad (4)$$

一个句子存在的概率等于该句子各词概率的累乘。大体实现方法是，给定一个含有词频的词典，对于一个带切分的语句，计算所有可能分词结果中概率(各词概率累乘)最大的一种切分，并将这种切分返回。根据后面的实验结果我们可以发现一元文法的性能相对于前面的机械匹配分词算法要提升接近两个百分比。

4.5.1.1 重要代码说明

下面对一元文法分词模型实现所需要的的主要代码逻辑进行说明。

- **根据给定文本生成含词频的词典**，附件 P3_5_1.py 中的方法 dic_for_uni() 对给定的训练样本进行分解，利用字典结构 word_dic 保存所有词到其词频的映射，并对所有词的前缀也予以保存，并以此生成词典 uni_dic.txt。
- **分词前读取词典**，从上述 uni_dic.txt 中读取词以及词频，存入到字典 dic 中供分词使用。
- **每个句子生成 DAG 并进行切分**，利用前缀词典对句子进行分析，从而生成对应的有向无环图 DAG。随后调用方法 calc_route()计算该句的切分路径，具体思想是利用动态规划算法，从后先前标

记开始位置，计算从句子的某一位置开始到任意 DAG 中值位置对应的词的概率，从而生成整个句子的所有可能切分的概率并选择其中的最大者。在计算词对应的概率时，为了防止词频为零导致的浮点数下溢，应对词频为 0 的情况取词频为 1 并将最终的概率结果取对数，减小浮点数精度造成的损失。得到对句子的切分后，掉用 Handle_number.py 中的 handle_specond()方法对切碎的特殊字符串进行处理。

4.5.1.2 实验结果分析

利用十折交叉验证的方法，我们分别用十个训练集生成前缀词典以及一元文法分词模型，并对相应的测试集进行切分、计算切分性能 score，得到平均结果如下：

```
uni_gram:
 准确率:  93.13159254301763%
 召回率:  95.83646009632119%
 F值: 94.46460469844469%
```

图 4.5.2 一元文法性能

```
FMM
 准确率:  91.92452101783475%
 召回率:  94.52930092815092%
 F值: 93.20865113934957%

BMM
 准确率:  92.10647058190526%
 召回率:  94.70241689746786%
 F值: 93.3863437963407%
```

图 4.5.3 前后向最大匹配性能

相对于机械匹配分词算法，一元文法三个指标均有接近两个百分点的提升，由于概率模型对自然语言特性的反映，在给定足够训练量的条件下，一元文法效果要好于机械匹配分词算法。

4.5.2 基于统计语言模型的二元文法实现

一元文法虽然实现较为简单，但是对于自然语言的上下文相关的特性并没有很好的进行反映，因此模型的分词性能还是存在提升的空间。而对于大于一元的 n-gram 模型，如果模型的阶数 n 过大，会导致相应的空间消耗指数级爆炸，另外随着句子长度的增大，其对应概率可能因为连续的词数过多而过小。因此常取 n=2，即为二元文法，给定二元文法的公式如下：

$$
\begin{aligned}
p(\boldsymbol{w}) &= p(w_1 w_2 \cdots w_k) \\
&= p(w_1|w_0) \times p(w_2|w_1) \times \cdots \times p(w_{k+1}|w_k) \\
&= \prod_{t=1}^{k+1} p(w_t|w_{t-1})
\end{aligned}
$$

(5)

二元文法模型又称作一阶马尔可夫链。它假设是一个词语的出现仅依赖于前一个词语，而与其所在位置和其与词无关。它相对于一元文法利用了更多上下文信息，并且在一定程度上控制了句子过长导致的效率降低问题。

4.5.2.1 重要代码说明

- **根据给定样本生成二元文法词典**，由于二元文法需要比较词对之间相对概率的大小，因此要提前生成含有训练样本中相邻词对出现的频率，从而利用极大似然估计的思想近似给出词对的条件概率。该部分内容由附件 P3_5_2.py 中的方法 gene_bi_dic()实现，其通过为所有前序词（包括二元文法所需的‘BOS’）创建字典，并将对应的相邻词及其频率作为键值对插入字典中，进而生成总的二元文法词典。
- **模型开始切分前读取词频词典 uni_dic.txt 和二元文法词典 bi_dic.txt**，前者方便统计所有词的总出现次数，后者为了得到所有已有词对的相对频率。

- **相对频率的计算**，calc_logp()方法计算相邻词的条件概率，分别利用前者的词频的对数和词对出现的频率的对数，相减得到条件概率。但为了处理相对词频为 0 的情况，为相对词频做加一处理，并将前序词的总频率加上词典中的总词数，将极大似然估计更改为极大后验估计。

- **最大概率路径的计算**，给定待切分文本及其 DAG，calc_route()方法应用动态规划算法，从前向后执行。首先计算每一个初始词（‘BOS’之后）出先的概率，随后计算出其每一个可能相邻词搭配并计算出对应的概率，保存前面的词及其对应的组合概率。如此到句子末尾，得到最大概率，随后从末尾再向前回溯，寻找前相邻的最大概率词直到句首，即可得到最大概率路径。

- **二元文法实现**，经过上述初始化后，对于带切分文本的每一行，将其处理为二元文法适用的以‘BOS’开头并以‘EOS’结尾的格式，随后调用前一节实现的 gene_DAG()（见附件 P3_5_1.py）方法生成该句子的 DAG，并调用 calc_route()方法得到最大概率路径，从而进行切分。

4.5.2.2 实验结果分析

通过十折交叉验证方法，对十个切分结果进行分析得到如下性能：

```
bi_gram:
 准确率:  92.4935633037692%
 召回率:  94.93449636185771%
 F值: 93.69806462005856%
```

图 4.5.4 二元文法性能

可以看出，在给定的样本较少且缺少平滑处理的情况下，二元文法的表现比一元文法略差。因此需要进一步对其进行更优的平滑处理。

5 结论及展望

5.1 实验总结

- 本次实验通过对词典的实用性进行分析，了解了不同分词任务对词典的不同需求。同时使用时间复杂度较高但代码量最少的方式实现了正反向最大匹配算法，并对分词结果进行了性能分析。在第四部分通过自主构建 trie 树，对机械匹配分词效率有了数千倍的提升。

- **实现两种统计语言模型：一元和二元文法，**分词系统的实现从概率上对汉语言分词提供了新的解决方案，模型同时对自然语言的上下文信息进行了吸收。

5.2 展望

- 这次实验并没有应用学过的算法对二元文法进行平滑处理，希望日后能够实现相关功能。同时，实验第四部分的数据结构仍然有改进的地方，比如使用双数组 trie 树进行词典存储。

参考文献

[1] 朱德熙. 语法讲义. 北京: 商务印书馆, 1982 (Zhu Dexi. Lectures on Grammar (in Chinese) . Beijing: Commerce Publishing House, 1982)

[2] 张华平, 刘群. 基于 N-最短路径的中文词语粗分模型. 中文信息学报, 2002, 16 (5) : 1 ~ 7 (Zhang Huaping, Liu Qun. Model of Chinese words rough segmentation based on N-shortest-paths method. Journal of Chinese Information Processing (in Chinese) , 2002, 16 (5) : 1 ~ 7)

[3]武红. 分词词典的构建[D].内蒙古师范大学,2010.