**汉语分词系统**

**哈尔滨工业大学**

**林燕燕 1190200501**

**摘要**

绪论

中文分词技术，是由于中文与英文为代表的拉丁语系语言相比，英文以空格作为天然的分隔符，而中文由于继承自古代汉语的传统，词语之间没有分隔。古代汉语中除了连绵词和人名地名等，词通常就是单个汉字，所以当时没有分词书写的必要。而现代汉语中双字或多字词居多，一个字不再等同于一个词。且在中文里，“词”和“词组”边界模糊。

中文分词中存在歧义识别和新词识别两大难题。

相关信息

## **实验目标**

本次实验目的是对汉语自动分词技术有一个全面的了解，包括从词典的建立、分词算法的实现、性能评价和优化等环节。本次实验所要用到的知识如下：

* 基本编程能力（文件处理、数据统计等）
* 相关的查找算法及数据结构实现能力
* 语料库相关知识
* 正反向最大匹配分词算法
* N 元语言模型相关知识
* 分词性能评价常用指标

## **编程语言与环境**

Python 3.7.9 ，Windows11，VScode

## **项目目录说明**

目录中存在Code和io\_files两个文件夹，Code文件夹中存放第一部分到第四部分实验代码，io\_files文件夹中存放第一部分到第四部分实验产生文件和依赖文件。

**io\_files文件夹：**

* **199801\_sent.txt** 为标准文本，是1998 年 1 月《人民日报》未分词语料，用于产生训练集和测试集
* **199801\_seg&pos.txt** 为标准文本，是1998 年 1 月《人民日报》的分词语料库，用于产生测试集对应的分词标准答案
* **dic.txt**为自己形成的分词词典，存放根据训练集产生的词典
* **train.txt** 为训练集，取分词语料库中 的数据作为训练集用于生成词典
* **std.txt** 为标准答案， 取分词语料库中另外 的数据作为标准答案，与分词结果进行比对计算准确率、召回率和F 值
* **test.txt** 为测试集，在未分词语料中取与标准答案相对应的 的数据作为测试集产生分词结果
* **seg\_FMM.txt** 为全文的分词结果，使用正向最大匹配分词，使用train.txt文件作为训练集，将199801\_sent.txt文件进行分词
* **seg\_BMM.txt**为全文的分词结果，使用反向最大匹配分词，使用train.txt文件作为训练集，将199801\_sent.txt文件进行分词
* **score.txt**为第三部分生成的评测分词效果的文本，其中包括准确率(precision)、召回率(recall)和F 值
* **seg\_FMM\_1\_10.txt** 为测试集分词结果，使用正向最大匹配分词，使用train.txt文件作为训练集，将test.txt文件进行分词
* **seg\_BMM\_1\_10.txt** 为测试集分词结果，使用反向最大匹配分词，使用train.txt文件作为训练集，将test.txt文件进行分词
* **better\_seg\_FMM.txt** 为测试集分词结果，使用优化后的正向最大匹配分词，使用train.txt文件作为训练集，将test.txt文件进行分词，计算分词时间与seg\_FMM\_1\_10.txt分词时间进行比较
* **better\_seg\_BMM.txt** 为测试集分词结果，使用优化后的反向最大匹配分词，使用train.txt文件作为训练集，将test.txt文件进行分词，计算分词时间与seg\_BMM\_1\_10.txt分词时间进行比较
* **TimeCost.txt** 为分词所用时间，存放优化前和优化后的分词时间

**Code文件夹：**

* **part\_1.py** 为实验第一步词典的构建代码，其中包括生成分词词典函数以及生成训练集、测试集和标准答案的函数
* **part\_2.py** 为实验第二步正反向最大匹配分词实现代码，其中包括读取词典内容函数、正向最大匹配分词函数和反向最大匹配分词函数
* **part\_3.py** 为实验第三步正反向最大匹配分词效果分析代码，其中包括计算评测得分函数，计算总词数和正确词数函数，计算准确率、召回率和f值函数以及获取词对应下标的函数
* **part\_4.py** 为实验第四步基于机械匹配的分词系统的速度优化代码，其中包括Trie树的实现以及其中添加字符串函数，查找字符串函数，在子节点中查找字符对应位置函数和返回哈希值函数，还有获得正向最大匹配的词典树函数，获得反向最大匹配的词典树函数，优化后正向最大匹配分词函数，优化后反向最大匹配分词函数，全文分割函数以及计算时间函数

训练测试

训练集、测试集和标准答案来源于199801\_sent.txt 和199801\_seg&pos.txt文本文件，训练集为199801\_seg&pos.txt文件的 ，标准答案为剩下的 ，对199801\_seg&pos.txt文件各行取模，若行数模10余0，则加入测试集，剩余的加入标准答案。

在199801\_sent.txt中取与标准答案对应行加入测试集。

词典构建

构建词典时，不将量词加入词典，可以减少词数，带来空间和时间上的优化，且词典中不存在重复词。

读取训练集内容，进行逐行提取，以词前的空格和词后的 ' / ' 字符提取词，若词前存在 ' [ ' 字符则去除该字符。

将提取出来的词加入列表，在训练集所有内容提取完毕后对列表进行排序。

将列表输出至词典文件生成词典，一行存储一个词便于后续读取。

正反向最大匹配分词实现

## **正向最大匹配分词-最少代码量**

**基本思想：**

假定分词词典中的最长词有i个汉字字符，则用被处理文档的当前字串中的前i个字作为匹配字段，查找字典。若字典中存在这样的一个i长度的字词，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i长度的字词，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。

**代码逻辑：**

读取词典文件生成词典列表并获取词典中最长词词长，读取测试集，对测试集各行进行如下操作：

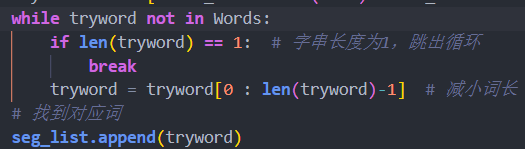
读取该行字符串，读取前最大词长个词，在词典列表中遍历查找是否有符合词，若没有，则将该词最后一个字符去除，即词长减一，再进行查找，若有匹配词，则将该词加入分割后字符串，并用 ' / ' 字符及空格进行分割，在未分割字符串中去除对应字符，再将去除后的字符串继续进行分割。

由于词典中不存在量词，则在分割过程中，判断连续的量词、字母、英文符号组合视为一个词，在其后添加' / ' 字符及空格符号进行分割，以实现部分量词的分割。

该行分割完毕后使用换行符分割各行，所有行分割完毕后将分割后字符串写入文件。

**最少代码：**

为实现最少代码量，则使用python list数据结构结构以及自带的关键词**in**进行遍历查找，该部分代码行数为21行。



**结果分析：**

由于使用最少代码量代码，并未对数据结构及查找算法进行优化，对十分之一的测试集需要半个小时才能运行结束，则对于所有文本则需四到五个小时，效率非常低。

## **反向最大匹配分词-最少代码量**

**基本思想：**

与正向最大匹配分词相反，取当前字串的后i个字作为匹配字段，若匹配不成功，则去掉匹配字段最前面一个字，对剩下的字串重新进行匹配处理。

**代码逻辑：**

读取词典文件生成词典列表并获取词典中最长词词长，读取测试集，对测试集各行进行如下操作：

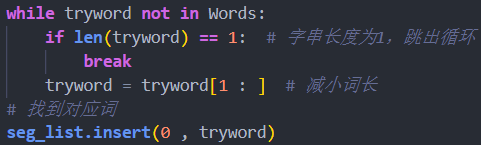
读取该行字符串，读取后最大词长个词，在词典列表中遍历查找是否有符合词，若没有，则将该词最前一个字符去除，即词长减一，再进行查找，若有匹配词，则将该词插入入分割后字符串的前面，并用 ' / ' 字符及空格进行分割，在未分割字符串中去除对应字符，再将去除后的字符串继续进行分割。

由于词典中不存在量词，则在分割过程中，判断连续的量词、字母、英文符号组合视为一个词，添加' / ' 字符及空格符号进行分割，以实现部分量词的分割。

该行分割完毕后使用换行符分割各行，所有行分割完毕后将分割后字符串写入文件。

**最少代码：**

为实现最少代码量，则使用python list数据结构结构以及自带的关键词**in**进行遍历查找，该部分代码行数为23行。



**结果分析：**

与正向最大匹配分词最少代码量实现一样，并未对数据结构及查找算法进行优化，所需时间长，效率非常低。

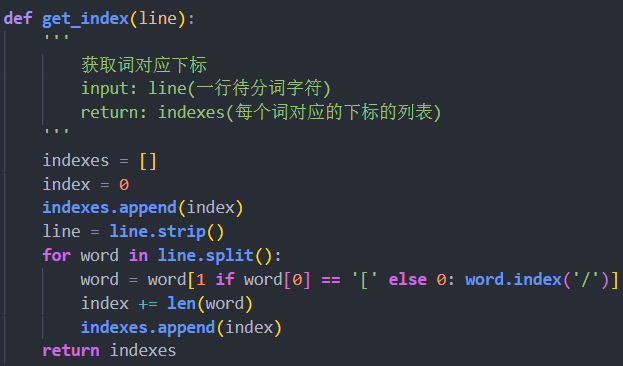
1. 正反向最大匹配分词效果分析

该部分代码实现对全文的分词结果的评测，全文分词结果保存在seg\_FMM.txt， seg\_BMM.txt文件中，将其与标准分词结果199801\_seg&pos.txt进行比对，计算准确值、召回率以及F值。

**代码实现：**

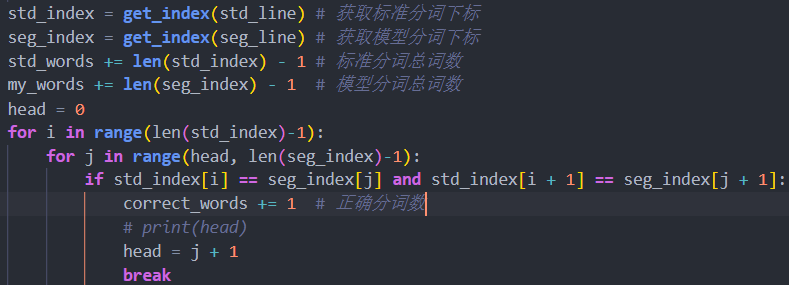
对模型分词结果和标准分词结果同一行进行处理：

去除各行中的分隔符，获取每个词开头对应的下标，添加到列表中，最后添加上总字数即结尾下标。

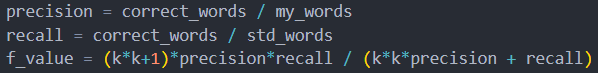


下标个数减一即为当前行分词数，加入总词数。

比对时使用滑动窗口法，将标准分词该行的每个词和模型分词该行的每个词作为窗口，若模型分词中某词对应的两个下标与标准分词中某词的完全相等，则正确词数加一，若不等，则取下一对下标进行比对，最终计算出正确分词数。

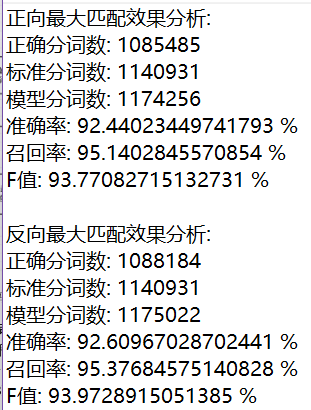


使用公式代入正确分词数、标准分词数和模型分词数计算准确率、召回率和F值。



**结果分析：**

FMM和BMM的分词效果存在一定性能差异，且均是BMM效果较好。



这是由于汉语中偏正结构较多，从后向前匹配，可以适当提高精确度。所以，反向最大匹配法比正向最大匹配法的误差要小。统计结果表明 ，使用正向最大匹配的错误率为 1/169，使用反向最大匹配的错误率为 1/245。例如切分字段“硕士研究生产”，正向最大匹配法的结果会是“硕士研究生 / 产”，而逆向最大匹配法利用逆向扫描，可得到正确的分词结果“硕士 / 研究 / 生产”。

1. 基于机械分词系统的速度优化

该部分对机械分词系统进行优化，在代码中使用Trie树优化字典的数据结构，并使用哈希函数优化查找。

Trie树又叫字典树，是对于字典的一种存储方式。这个词典中的每个词就是从根节点出发一直到某一个目标节点的路径，路径中每条边的字母连起来就是一个词。

**代码实现：**

代码实现一棵Trie树以及其查找字符、添加字符串的函数。树的每个节点存放一个字符、该节点是否是一个词的结尾、该节点的子节点以及该节点在同父节点的子节点中的位置。

使用哈希值查找某字符对应位置，使用ord()函数得到该字符对应数，将此数模子节点的个数得到哈希值。



get\_child\_by\_char()函数在子节点中查找某字符对应位置，使用哈希函数计算出位置，若位置对应节点为空，则返回false及该位置；若不为空，则判断该位置上是否为查找的字符，若是，则返回true即该位置，若不是，则将位置加一再模子节点个数，继续循环查找，直到节点为空或字符匹配。



insert\_word()函数为树添加字符串，即将词典中的各个词加入Trie树。对需加入词的每一个字符进行遍历，使用get\_child\_by\_char()函数获取字符的位置及该字符是否已经存在在对应路径上，若不存在，则新建节点，如果该字符为该词最后一个字符，则将节点中该节点是否是一个词的结尾的值置为true，如果不是最后一个字符，将父节点设为该字符，考察下一个字符。

search\_word()函数查找树中字符串。对需查找的字符串进行遍历，使用get\_child\_by\_char()函数获取字符的位置及该字符是否已经存在在对应路径上，若字符对应位置的节点是一个词的结尾，记下该节点位置为res，持续遍历直到某字符不存在则返回上一记下的res值或遍历结束时返回该字符长度。

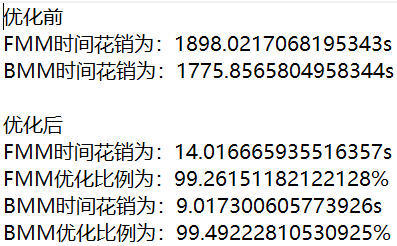
对FMM，将字典中每个词插入字典树，对文本中每行取前最大词长个词，在字典树中查找，若不存在，去掉一个字符，继续查找，与优化前进行相同处理实现分词。

对BMM，将词典中每个词反向，再插入字典树，对文本中每行取后最大词长个词，进行与FMM相同操作，得到分词后将词反向获得分词结果。

使用十分之一数据作为测试集，计算优化前与优化后分词时间，输出至TimeCost.txt文件。

**结果分析：**

对代码中数据结构和查找算法进行了优化，使用字典树加哈希值进行优化，得到如下结果。



可以看到时间得到了极大的优化，且BMM消耗时间更少。

参考文献