汉语分词系统

1190300321 郑晟赫
哈尔滨工业大学
1190300321@stu.hit.edu.cn

摘要

本次实验的主要目的是对汉语分词技术有一个全面的了解。基于这一目的，本实验主要实现了基于机械匹配的分词系统（正反向最大分词匹配）的实现与速度优化、基于 N 元文法的分词系统、使用隐马尔可夫模型作为识别未登录词的分词系统。最终实现的系统使用二元文法与基于字符的分词方法的融合，在训练集上分词结果 F 值达到 0.999，测试集上分词结果 F 值达到 0.955。

1 绪论

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。与英语等语言不同，汉语的词之间不存在空格或其他标识符作为词的边界标志，因此产生了汉语分词问题。而自动分词也是正确的中文信息处理的基础，分词系统性能的好坏将直接影响后续中文信息处理的性能。

目前中文分词存在如下难点：规范分词；歧义词切分；未登录词识别等。

目前较为主流的非深度学习的中文分词方法可以分为三种：1. 基于词典的分词；2. 基于规则的分词；3. 基于统计的分词方法。本实验实现的主要是第一种和第三种分词方法。在性能优化时使用二元文法与基于字符的分词方法的结合 [Wang et al.2009]。实验实现具体内容如下：

1. 词典的构建
2. 正反向最大匹配分词实现和效果分析
3. 基于机械匹配分词实现和效果分析
4 . 基于 MM 的一元、二元文法分词
5. 基于 HMM 的未登录词识别
6. 基于二元文法与字符分词结合的系统

2 相关工作

Jieba 分词[1]：Jieba 分词是一个 Python 中文分词组件，可以对中文文本进行分词、词性标注、关键词抽取等功能。由于本实验实现的是分词，因此介绍 Jieba 分词中的分词算法。Jieba 分词中使用的主要分词算法是 N 元文法分词，并使用隐马尔可夫模型作为未登录词处理的方式，同时使用了 Trie 树对词典存储与搜索做了一定的优化。其主要算法部分使用的算法与思想与本实验大部分内容基本重合。

SnowNLP 分词[2]：SnowNLP 是一个 python 写的类库，可以方便的处理中文文本内容。主要实现了分词、词性标注、情感分词、文本分类等中文文本处理功能。其分词算法使用的是基于字符的生成式模型，使用了三阶隐马尔可夫处理未登录词问题。本实验最终的优化方式思路与其相似。

3 实现过程

3.1 词典的构建与分析

词是组成词典的基本单位，当从训练样本中选择加入词典的词时需要有一定的选择性。词典的构建需要在以下目标中进行权衡：分词的正确率，分词算法运行的速率。这两个目标很多时候对于词典构建来说是难以同时达到最优的。因此在构建词典时考虑如下标准：

[1] https://github.com/fxsjy/jieba

[2] https://github.com/isnowfy/snownlp

3.1.1 选择性

从训练文本中可以发现，大多数长词（词长超过 10 视为长词）为英文单词、网站、外文译名（人名、地名）。在大多数情况下这类词在其他文本中大范围出现的概率并不大，但是词典中长词过多在进行机械匹配的时候将会使得匹配时间大幅度上涨。经过试验测试，对于 3.1-3.4 的机械分词任务加入长词基本不会使得分词性能上涨，但是使得分词时间大幅度上升（见 4.2 节）。因此选择将英文长词、词长超过 10 的中文译名、网站名从词典中删除。

3.1.2 任务相关性

不同的分词策略使用的词典可以不同。上述基于选择性原则生成的词典是最基本的词典。但是任务的不同对于词典的要求也不尽相同。例如，使用二元文法分词时需要生成的词典是词对与词对出现的次数，与一元文法的使用的词典明显不用。虽然这一部分可以在系统运行过程随代码运行生成，不做存储。但是为了加快系统分词的总耗时，选择将每一部分的词典生成后作为文件读取。同时通过实验得到，当使用统计语言模型分词时，长词的增多不会使得分词时间出现明显上涨。对于分词准确性，当使用基于 HMM 的未登录词识别时，词典中的词越多分词的性能就越好，基于这一特性，为统计语言模型分词建立其独有的词典。

3.1.3 易读性

词典的构建需要有一定的结构，并需要具有一定的可读性。虽然对于计算机来说只需要结构化即可，但是当词典具有易读性之后对其对性能造成的影响的可解释性分析将会更容易展开。因此词典使用如下结构：3.1-3.4 使用的词典中每一行包含词、词性、词在训练样本中出现的次数。其中三个部分使用 $\backslash t$ 分隔。而一元文法使用的词典结构与上述词典相同，但是加入中文译名。而二元文法词典结构为：词 1，词 2，词对出现的次数。其中三部分使用 $\backslash t$ 分隔。

3.2 正反向最大匹配分词

正反向最大匹配基于如下思想：在分词过程中使得分词的每一个词都尽可能长，当匹配结束之后就得到分词结果。由于本部分实现最小代码实现，因此词典储存在 Python 的 list 中，在查找时使用遍历查找。

FMM 算法逻辑：FMM 在分词过程中将词典中的最长词的词长设为匹配的词长上限。从需要分词的文本的句首开始，从这一词长开始，遍历词典中的每一个词，如果词典中当前词长的词与需要分词的文本中不存在匹配关系，将当前词长减一继续匹配。直到当前词长为 1 时仍无法匹配，则词典中不存在相关匹配，将文本中当前字单独分词。算法逻辑可见下图：

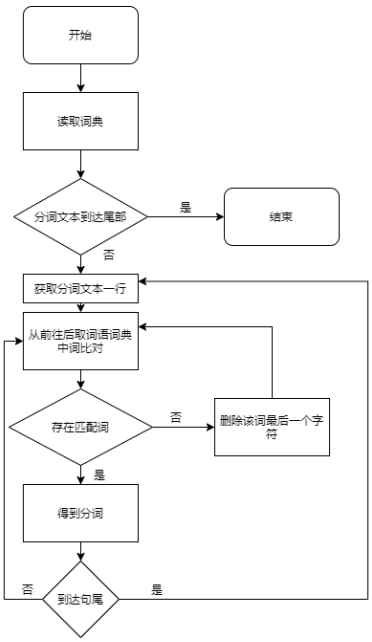

图 1: FMM 算法流程图

BMM 算法逻辑：与 FMM 不同点在于 BMM 从需要分词的文本的句尾开始匹配，找到最长的匹配词。算法逻辑基本类似，不再赘述。

收获：经过测试发现虽然这一部分代码相对来说简单，但是分词速度极慢，如果要对实验文本整体切分，需要大于 5h 的时长。由此可见在代码已经进行优化的前提下，代码的复杂程度很多时候代码能实现的时间或准确率性能指标是成反比的。因此下面几部分实现了对于机械分词速度的优化。

3.3 正反向最大匹配分词效果分析

在分词效果分析中，主要是实现了对于分词准确率、召回率以及 F 值的计算。其中三个值的计算公式如下：

$$\text{准确率}(P) = \frac{\text{切分结果中正确分词数}}{\text{切分结果中的所有分词数}}$$

$$\text{召回率}(R) = \frac{\text{切分结果中正确词数}}{\text{标准答案中所有分词数}}$$

$$F = \frac{2PR}{P+R}$$

FMM 与 BMM 分词性能比较：对 FMM 与 BMM 分词性能做了多次不同规模的封闭性测试与开放测试集测试，观察测试结果可知在测试出现的所有情况中 BMM 的分词性能均优于 FMM(见 4.3 节)。

FMM 与 BMM 分词性能分析：以下从直观与理论分析两个层面对于 BMM 分词结果优于 FMM 进行分析：1. 直观上来看，如果不考虑歧义问题，FMM 分词出现错误的情况就是当前分词结果后加上几个字符可以组成一个词，而这个词更符合实际情况的分词需求；而 BMM 对应的错误就是当前词前加上几个字可以组成一个新词，这个新词更符合当前情景下的分词结果。直观上来说，对于汉语来说当前词之后加上几个字符构成新词的概率要远远大于一个词前加上字符组成符合当前情景的新词的概率（具体原因见下述分析）。

2. 从语言特性上分析，在正常的句子中将会出现较多偏正结构，对这些结构的处理将直接影响分词性能。也就是将会出现一个词修饰另外一个词的情况。对于汉语来说，偏正结构主要分为定中结构和状中结构，可以发现在修饰词不太长的时候中文倾向于将修饰词置于中心词之前。也就是说对于大部分偏正结构来说重点是放在后一部分的，后一部分需要成一个词才能成为正确的分词方式。而对于 FMM 来说如果出现修饰词恰好能和中心词的一部分形成一个词，那么其选择划分，这就背离了需要将中心词单独成词的原则。而对于 BMM 来说，由于其从句尾开始匹配，因此有更大概率能抽取到中心词。例如，如果出现：“当时间” 这一短语，FMM 大概率将其分为“当时/间”，而 BMM 将会将其分为 “当/时间”。

3.4 基于机械匹配的分词系统速度优化

对于机械匹配分词系统来说，优化速度主要优化的就是词典中的查找算法由于 3.2 中词典中每一个词的查找都是遍历查找，对于每一个词的查找时间复杂度都是 O(n)。因此在速度优化中实现了基于哈希表的速度优化。经过优化后，在对完整测试集进行划分时相对最简单的版本耗时缩小为 $\frac{1}{2500}$（从 5h 优化到 8s 左右），相对于二分查找耗时缩小为 $\frac{1}{8}$（从 64s 优化到 8s 左右），由此可见优化方案是可行的。

速度优化方案分析：优化过程中针对 FMM 进行速度优化。速度优化中主要目标就是将查找的时间复杂度降为 O(1)。因此在实现过程中选择采用哈希表的方式。对于每一个词散列函数为：将每一个词分为单字，单字的散列值等于其 utf-8 编码值乘 6143，其中 6143 是一个素数。每一个词的散列值等于组成其每一个单字的散列值之和。对于在散列过程中将会出现的冲突情况，为了尽可能优化性能，使用拉链法，为每一个冲突位置建立一个链表存储出现冲突的几个值，避免了使用开放定址法时出现的额外时间开销。使用哈希查找的算法逻辑如图 2：

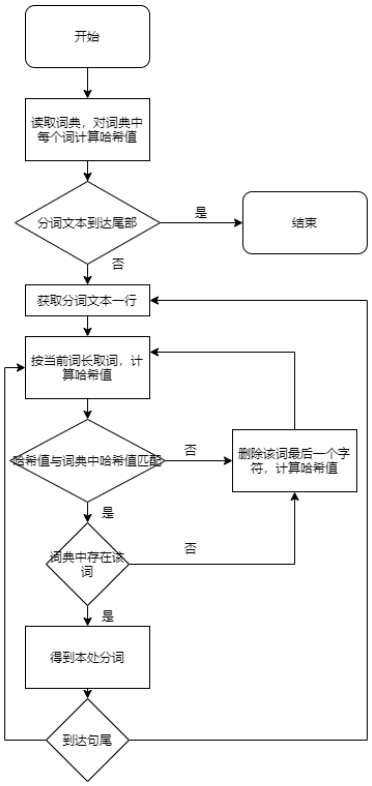

图 2: 哈希速度优化流程图

继续优化方案：与 Python 中的数据结构 set() 构建的词典相比，本分词系统实现的时间优化仍有较大优化空间。因此提出如下两种优化方案：**方案 1**：当前的哈希表仍存在一定量的冲突，而且存在一定程度的元素聚集现象。因此如果希望从哈希表入手需要提出一个更适合的哈希函数，目前使用的线性散列函数过于简单，并不适合。如果考虑实现的简易型，可以考虑使用乘余取整法、数字分析法等常用哈希函数进行优化；如果希望尽可能减少冲突，可以考虑使用 SHA，MD5 等相对较为复杂的散列函数。**方案 2**：从匹配次数入手，可以建立双 trie 树进行优化。基于双 trie 树的查找算法在极大减少存储空间的同时，由于其使用的是前缀匹配的方式，因此可以在一定程度上减少匹配的次数，减少计算的次数。尽管如果使用哈希表实现双 trie 树其查找时间度也是 O(1)，但是相对单纯使用哈希表将会有一定时间优势。

3.5 基于统计语言模型的分词系统实现

本部分实现了一元文法分词、二元文法分词，由于基本想法一致，因此主要对于二元文法的实现进行介绍。

3.5.1 二元文法整体分析

对于 N 元文法来说，其主要思想就是当前词出现的概率依赖于前面出现过的 N 个词。这一点相对来说是比较好理解的，在一个连续文本中，当前词与上文一定是会有一定的关系的。但是如果选用全体前文作为上文预测当前词出现的概率，容易出现数据稀疏的情况。因此二元文法就是假设当前词出现的概率与前一个词有关，因此可以得到下式：

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

其中 S 表示分词序列。因此二元文法的目标就是最大化 P(S)。因此在分词过程中首先做出句子的全切分图，在全切分图上使用维特比算法作为寻找最大化 P(S) 的路径。最终还原这一路径就是需要的分词结果。详细算法逻辑见图 3：

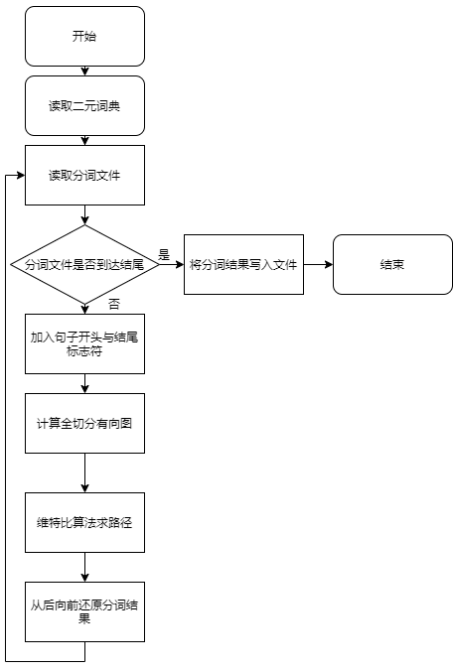

图 3: 二元文法流程图

3.5.2 二元文法实现分析

二元文法分词实现过程中主要部分如下：生成词典、生成全切分有向图、维特比算法、概率平滑。接下来逐部分分析。

生成词典：二元文法需要生成前缀词典，这一前缀词典中每一个对应项的格式应为：词对:词对出现次数。在生成过程中对训练样本中每一句的开头加上 <BOS> 标志句首，在句尾加上 <EOS> 标志句尾。对于处理后的训练样本中连续出现的两个字记录一个出现次数，将这些出现次数汇总就是二元文法词典。

概率平滑：由于对于一些词对在词典中并不出现，而如果这样的词在分词进行的过程中出现就会导致错误，因此需要对词对出现的次数进行平滑。本实验中采用的是实现较为简单的加法平滑方法。

生成全切分有向图：生成全切分有向图的过程是一个遍历的过程：从句首开始遍历每一个位置，在当前位置取出其后续能够组成词典中词的所有组合形式并进行记录每一条路径。当从句首遍历到句尾的时候就得到了这个句子对应的全切分有向图，而这个全切分有向图的每一条路径对应一种分词方式。

维特比算法：维特比算法是寻找全切分有向

图中的一条最优路径的方式，其本质是一种动态规划算法，算法伪代码如下：

算法 1 维特比算法

输入: 观测值取值空间 $O = o_1, o_2, ...o_N$; 状态空间 $S = s_1, s_2, ...s_K$; 观测序列 $Y = y_1, y_2, ...y_T$; 转移矩阵 $A_{K\times K}$，其中 $A_{i,j}$ 存储从状态 s_iz 转移到 s_j 的概率；初始情况矩阵 π_K 其中 π_i 存储的是 $x_1 == s_i$ 的概率。

输出: 最可能的观测序列

function VITERBI(O, S, π, Y, A)
  for each state s_i: **do**
    $T_1[i, 1] \leftarrow \pi_i$
    $T_2[i, 1] \leftarrow 0$
  end for
  for $i \leftarrow 2, 3, ...T$ **do**
    for each state s_j **do**
      $T_1[j, i] \leftarrow max_k(T_1[k, i-1]A_{k,j})$
      $T_2[j, i] \leftarrow argmax(T_1[k, i-1]A_{k,j})$
    end for
  end for
  $Z_T \leftarrow argmax_k(T_1[k, T])$
  $X_T \leftarrow S_{Z_T}$
  for $i \leftarrow 2, 3, ...T$ **do**
    $Z_{i-1} \leftarrow T_2[Z_i, 1]$
    $X_{i-1} \leftarrow S_{Z_i-1}$
  end for
  return X
end function

3.6 性能优化

3.6.1 基于 HMM 的未登录词处理

原理分析：由于使用 N 元文法的时候难以很好的处理未登录词，对于未登录词 N 元文法将会将其分割为数个其见过的词，而一些地名、人名在词典中不太可能全部出现过，因此需要一种方法来对未登录词处理。而 HMM 算法对于处理未登录词有一定的特点和优势。HMM 算法通过对训练集文本每一个字符的“BMES”（表示该字在词中的位置）进行标注之后得到训练集上的转移概率、发射概率和初始状态概率。当实际使用在分词中时：使用维特比算法取出全切分有向图中出现可能性最大的一条路径。HMM 算法伪代码如下：

算法 2 HMM

输入: 观测序列 $O = o_1, o_2, ..., o_T$, 模型参数 $\lambda = (A, B, \pi)$

输出: 观测序列 P(O|λ);

1: 计算时刻 1 各个隐藏状态 s_i 的前向概率：$\alpha_i(i) = \pi(i)B_{i,o_i}$
2: 递推 2,3,...,T 时刻的前向概率:$\alpha_{t+1}(i) = [\sum_{j=1}^{N} \alpha_t(j)A_{ji}B_{i,o_{t+1}}]$
3: 最终结果:$P(O|\lambda) = \sum_i^N \alpha_T(i)$;
4: **return** :$P(O|\lambda)$;

代码细节分析：1. 在实际未登录词处理中，由于一些未登录词是共性的，例如 1997 年，在进入生成全切分有向图之前对于这类未登录词使用正则表达式的方式进行提取并将这类未登录词变换为一类特定 ID，在分词过程中，遇到这类 ID 视为一个词。2.HMM 的未登录词处理作为二元文法的后处理方式，也就是对于一句话先使用二元文法进行分词，将分词结果输入 HMM 模型进行进一步未登录词处理。

3.6.2 基于二阶 HMM 的优化（二阶 HMM+二元文法）

对于未登录词（OOV）HMM 有较好的识别效果，但是 HMM 的缺点也较为明显——对于词典中的词（IV）确未能很好地识别。主要是因为 HMM 本质上是一个 Bigram 的语法模型，未能深层次地考虑上下文。因此，下述方法将使用二阶 HMM 作未登录词识别的方法。类比于 HMM，二阶 HMM 的状态转移依赖于其前两个状态，因此分词模型如下：
$\arg\max_{t_1^n}[\prod_{i=1}^{n} P(t_i|t_{i-1}, t_{i-2})P(c_i|t_i)]P(t_{n+1}|t_n)$

其中 t_i 表示状态，c_i 表示当前字符。

因此二阶 HMM 事实上可以看做 HMM 与二元文法的融合，只不过是二元文法正常情况下处理的是字符，在这里处理的是字符的标注。

而 [Brants2000] 基于二阶 HMM 提出 TnT(Trigrams'n'Tags) 序列标注方案，对条件

概率 $P(t_3|t_2,t_1)$ 计算方式如下：$P(t_3|t_2,t_1) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_2,t_1)$

记一元、二元、三元文法公式如下：

$Unigram : \hat{P}(w_3) = \frac{f(w_3)}{N}$

$Bigram : \hat{P}(w_3|w_2) = \frac{f(w_2,w_3)}{f(w_2)}$

$Trigram : \hat{P}(w_3|w_1,w_2) = \frac{f(w_1,w_2,w_3)}{f(w_1,w_2)}$

基于上述式子，为了求解系数 λ，TnT 提出如下算法：

算法 3 TnT 算法参数求解

set $\lambda_1 = \lambda_2 = \lambda_3 = 0$;
for each trigram $t_1, t_2, t_3$with $f(t_1, t_2, t_3) > 0$
取三个值中的最大值
do
    case : $\frac{f(t_1,t_2,t_3)-1}{f(t_1,t_2)-1}$: λ_3 +=$f(t_1, t_2, t_3)$;
    case $\frac{f(t_2,t_3)-1}{f(t_2)-1}$: λ_2 +=$f(t_1, t_2, t_3)$;
    case $\frac{f(t_3)-1}{N-1}$: λ_1 +=$f(t_1, t_2, t_3)$;
end for

在二阶 HMM 分词实际使用过程中，实际操作就是将文本中的每一个字标注为“BMES”之一，表示其在词中的位置。标注结束之后对分词结果进行还原。除了转移概率计算方式不同，其余计算方式与 HMM 基本一致，因此详细过程在此不再赘述。

4 实验结果

4.1 实验设置性能比较

实验进行过程中，为使得性能指标更能反映分词系统的真实性能，采用了三组不同的实验方式：对训练样本分词、对已有数据进行 K 折交叉验证、使用单独测试集对分词系统进行测试。使用数据集均为人民日报分词语料集[3]。

4.2 词典设置

测试词典性能中，使用的分词方式为 FMM，主要的考核指标为分词合理性与分词时间指标。实验结果如图 4：

[3]https://github.com/fangj/rmrb

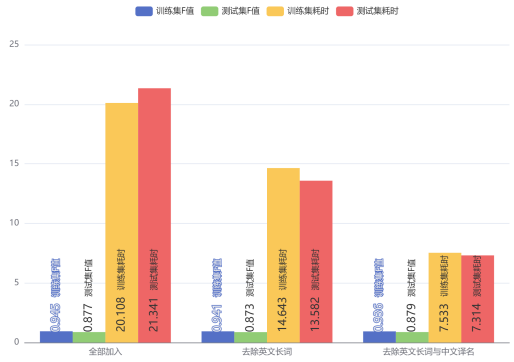

图 4: 词典选择性测试

可以发现将训练文本中的词全部加入之后并不会对分词性能有较大提升，但是将导致分词耗时有较大上涨。因此实际使用的词典选择去除英文词汇与中文长译名的形式。

4.3 FMM 与 BMM 性能比较

本节测试过程中使用在训练集与测试集上分别对 FMM 与 BMM 进行性能测试（不考虑时间）性能测试结果如下：

测试方式	FMM F 值	BMM F 值
训练集全分词	0.936	**0.938**
训练集 K 折交叉	0.891	**0.902**
测试集全分词	0.879	**0.884**

表 1: FMM 与 BMM 性能比较

测试发现在所有测试情况中 BMM 性能均优于 FMM，详细分析可见 3.3 节。

4.4 机械匹配时间性能优化

本节主要比较使用不同存储结构存储时对训练样本进行全分词时的耗时。时间性能见表 2：

查找方式	时间 (s)
list 遍历查找	15425.861
二分查找	51.907
自定义哈希	7.534
Python set	**4.251**

表 2: 不同查找算法时间性能

测试方式	FMM	BMM	一元文法	二元文法	OOV	二阶 HMM
训练集全分词	0.936	0.938	0.947	0.953	0.966	**0.999**
训练集 K 折交叉	0.891	0.902	0.915	0.926	0.935	**0.984**
测试集全分词	0.879	0.884	0.893	0.915	0.931	**0.955**

表 3: 不同分词方法分词结果 F 值比较

可以发现自定义哈希算法已经对最基本的遍历查找进行了较大程度的优化，但相比 Python 中的内置数据结构 set 仍有较大优化空间。

4.5 不同分词性能比较

本实验中实现了机械匹配分词（FMM、BMM），统计语言模型（一元文法、二元文法），二元文法 + 未登录词处理，本节主要考虑分词结果 F 值，暂不考虑时间性能。使用测试方式为：训练集全分词，训练集 K 折交叉验证，测试集全分词。测试结果见表 3。

经过比较可知，当训练样本充足时，统计模型性能将略优于机械匹配分词。同时如果不存在数据稀疏问题，二元文法分词性能明显优于一元文法，尤其是针对未见过的样本（测试集）。可以发现未登录词处理使得分词性能略有上升，经过对分词文件的分析，上升不明显的原因是 HMM 对于词典中的词处理并不好，可能出现将二元文法分正确的词进一步划分时出错。因此使用二阶 HMM 作为最终优化结果，优化性能较理想。

5 未来工作

对于分词性能的提升上，仍存在优化空间，基于目前已完成工作，提出优化方向：模型平滑处理，词典进一步筛选。

模型平滑处理：目前二元文法使用的平滑方式是加一平滑法，这一平滑方式尽管较为简单，但是平滑的性能并不是最佳，因此后续可以使用 Katz 平滑方法 [Katz1987] 对平滑进一步优化。

词典进一步筛选：词典中的词将会在很大程度上影响分词的效果。目前筛选的方式仍较为简单。未来可以考虑对于训练语料中出现的一些特定的符号（例如"[]"）等标志的词汇进行特殊处理，这一类词汇大概率是一个专有名词；同时可以对英文单词、数字在分词之前进行预取，将数字、英文、汉语进行分别分词后组合。

6 总结

实验实现了机械匹配分词并自定义数据结构优化查找时间性能，最终时间性能优化了 2500 倍以上；同时实验实现的统计模型分词方式从另外一个角度对汉语分词进行了解释与实现；最终优化性能使用的 HMM 未登录词处理与二阶 HMM 对于未登录词以及词典中以登录词分词性能达到了较为理想的状态。

参考文献

[Brants2000] Thorsten Brants. 2000. TnT – a statistical part-of-speech tagger. In *Sixth Applied Natural Language Processing Conference*, pages 224–231, Seattle, Washington, USA, April. Association for Computational Linguistics.

[Katz1987] Slava Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401.

[Wang et al.2009] Kun Wang, Chengqing Zong, and Keh-Yih Su. 2009. Which is more suitable for Chinese word segmentation, the generative model or the discriminative one? In *Proceedings of the 23rd Pacific Asia Conference on Language, Information and Computation, Volume 2*, pages 827–834, Hong Kong, December. City University of Hong Kong.