Pinyin Input Method Editor
Design Report

Yihong Gu
gyh15@mails.tsinghua.edu.cn
Department of Computer Science
Tsinghua University

1 Introction

报告分为三个部分：

- Language Model: 介绍所用的语言模型
- Search Algorithm: 介绍所用的搜索算法以及优化
- Experiments: 给出实验结果并作相关分析
- Code Structure: 简要介绍代码结构

2 Language Model

2.1 Probability Model

总体来说，我们使用以下语言模型：

$$\mathbb{P}(w_1 \cdots w_n) = \prod_{i=1}^{\min(n,m)} \mathbb{P}(w_i | w_{\max(i-m+1,1)} \cdots w_{i-1}) \tag{1}$$

我们把这个模型称为 m-gram 模型。
在这里面 w_i 表示第 i 个汉字，举个例子，取 $m = 2$：

$$\mathbb{P}(\text{清华大学}) = \mathbb{P}(\text{清})\mathbb{P}(\text{华}|\text{清})\mathbb{P}(\text{大}|\text{华})\mathbb{P}(\text{学}|\text{大}) \tag{2}$$

事实上，这里面我们没有考虑拼音的影响，那么，我们作最简单的假设，假设拼音和 $m-gram$ 独立并且条件分布是离散分布

$$\mathbb{P}(w_1 \cdots w_n | t_1 \cdots t_n) = \prod_{i=1}^{\min(n,m)} \mathbb{P}(w_i | w_{\max(i-m+1,1)} \cdots w_i - 1)\mathbb{P}(w_i | t_i) \tag{3}$$

我们让

$$\mathbb{P}(w_i|w_{i-m+1}\cdots w_{i-1}) = \frac{\#\{w_{i-m+1}\cdots w_m\}}{\#\{w_{i-m+1}\cdots w_{m-1}\}} \tag{4}$$

其中 $\#\{w_{i-m+1}\cdots w_i\}$ 为词组 $w_{i-m+1}\cdots w_i$ 在 corpus 中出现的频数，并且让 $\mathbb{P}(w_i|t_i)$ 为 1 当且仅当汉字 w_i 存在发音 t_i，否则为 0，我们也尝试了其他的模型 (均匀分布，按汉字的词频归一化的离散分布，但是发现实际上这些方法会引入大量噪声，实际效果并没有之前这种简单也不归一化的方法好，因为前一种方法让文本完全由 corpus 决定，不引入拼音造成的噪声)。

2.2 Frequency Count

在计算 $\#\{w_{i-m+1}\cdots w_i\}$ 的过程中，我们使用 sina 新闻 2016 作为 corpus，且把所有的 6763 个汉字作为 w_i 的字母表 Σ，把新闻正文中不属于 Σ 的部分作为分隔符，统计在 Σ 中的连续 m 个 token(中间不能有分隔符) 出现的次数。

由于总的次数过于多，我们考虑只保留部分 m-gram 的频数统计的结果，我们选取最大的 k，使得频数 $\geq k$ 的 m-gram 的频数之和大于总频数之和的 $100\sigma\%$，我们把 σ 称为 significance level，在这里我们取 $\sigma = 0.95$，最后我们保留频数 $\leq k$ 的 m-gram。

2.3 Probability Smoothing

首先，为了方便计算，我们同意使用概率取对数进行计算，这样原来的乘积就变成了求和。

由于词频很多时候都为 0，所以我们需要用对 $\log\mathbb{P}(w_i|w_{i-m+1}\cdots w_{i-1})$ 进行平滑处理。

我们下面考虑具体的处理过程 (递归处理)：

- 如果当前发现 $w_{i-m+1}\cdots w_i$ 和 $w_{i-m+1}\cdots w_{i-1}$ 的频数均非 0，那么就直接按照原式计算 $\log\mathbb{P}(w_i|w_{i-m+1}\cdots w_{i-1})$。
- 如果发现 $w_{i-m+1}\cdots w_{i-1}$ 的频数均为 0，并且 $m > 2$，计算 $m' = m-1$ 的结果 p_{m-1}，然后输出就是 $p_{m-1} - 100$，作为平滑处理的惩罚项。
- 如果发现 $w_{i-m+1}\cdots w_{i-1}$，并且 $m = 2$，计算 $m' = m-1$ 的结果 p_{m-1}，然后输出就是 $p_{m-1} - 2\times 10^8$，作为平滑处理的惩罚项。

另外，由于我们是 (要通过搜索) 需要最大化对数似然值，所以我们设置答案的下界为 -1×10^9，也就是说，像第三项的那种平滑处理不能超过 5 次。

3 Search Algorithm

有了 Langugae Model 后，我们的问题就转变成了最大化

$$w_1^*\cdots w_n^* = \mathrm{argmax}_{w_1\cdots,w_n}\mathbb{P}(w_1\cdots w_n|t_1\cdots t_n) \tag{5}$$

其中 $t_1\cdots t_n$ 是给定的拼音，同时 $w_1^*\cdots w_n^*$ 就是我们输出的结果。

我们考虑使用 A^* 算法来解决这个问题。

3.1 A^* Algorithm

我们把 $w_1\cdots w_i$ 称为一个状态 s_i，当 $i = n$ 的时候即到达终点，一个状态 s_i 的收益为 $v_i = \log\mathbb{P}(w_1\cdots w_i)$，我们需要最大化到达终点的收益 v_n。

服从 A^* 的记号，我们发现 $g(s_i) = v_i$，另外我们让 $h(s_i) = 0$，即可用 A^* 来优化。此时我们发现，这个问题实质上变成了一个最长路径问题，这时候的 A^* 也就等价于传统的 Dijkstra 算法。

3.2 Improvement

我们从以下一个角度来优化这个搜索过程：

SLF 优化

我们发现，是否对 OPEN 表排序 (即使用堆来维护 OPEN 表) 不影响时间消耗，所以我们不对 OPEN 表排序，这样的搜索算法就等价于传统的 SPFA 算法，我们沿用了 SPFA 算法的一个非常经典的优化手法 SLF 优化，即如果放入队尾的状态比放入目前队头的要优的话，把队头队尾的元素交换，这样可以使得效率提升 3 倍。

记忆化

我们发现，计算 local log probability ($\log \mathbb{P}(w_i|w_{i-m+1} \cdots w_{i-1})$) 非常消耗时间，所以我们对这一部分进行记忆化，这样效率也可以提升 1 倍。

政府工作报告 7 次提及李克强为何再赠 4 字？武汉最懒大学生：两周不收衣服鸟儿在内做窝特朗普称叙化武袭击事件是“对人类的羞辱”郎平: 女排备战奥运会培养新人已着眼下个周期