NLP Lab #1 - 统计语言模型

Part #1 实验简介

本次实验要求采用 N-Gram 的方法，具体来说，采用模型 bigram 和 trigram 以对中文数据集进行建模。此外，本次实验的重心在于测试不同平滑方法，如 Laplace 平滑、Good-Turing 平滑等平滑方法，在数据集上的表现。在本次实验中，本人不仅测试了上述两种平滑方法的性能，同时也测试了更一般的加法平滑法的性能，并对它们的表现做出了个人的解释。

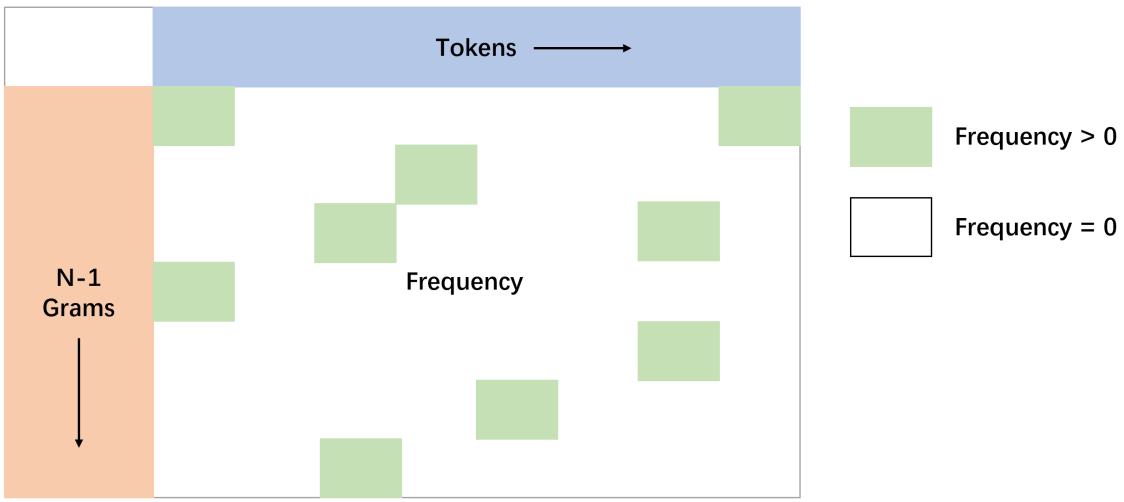
Part #2 技术细节

a) 句子补全

首先，本次实验的基础步骤，是将语料库处理为 N-Gram 的形式。出于 tokens 在被预测时具有一致性以及能确定句子结束的位置的考虑，本人为句子前后安插了 `<start>` 和 `<end>` 符号，以分别表示句子的开头和结尾。具体来说，对于 N-Gram 的模型，本人将在句子的开头插入 $(N-1)$ 个 `<start>`，并在句子的结尾插入一个 `<end>`，这样，就可以使得每一个单词在被预测时，都具有完整的 N-Gram 形式，同时可以预测句子的结尾。值得补充说明的是，在这样的设定下，`<end>` 是 token 库的成员而 `<start>` 不是。

b) 统计方法

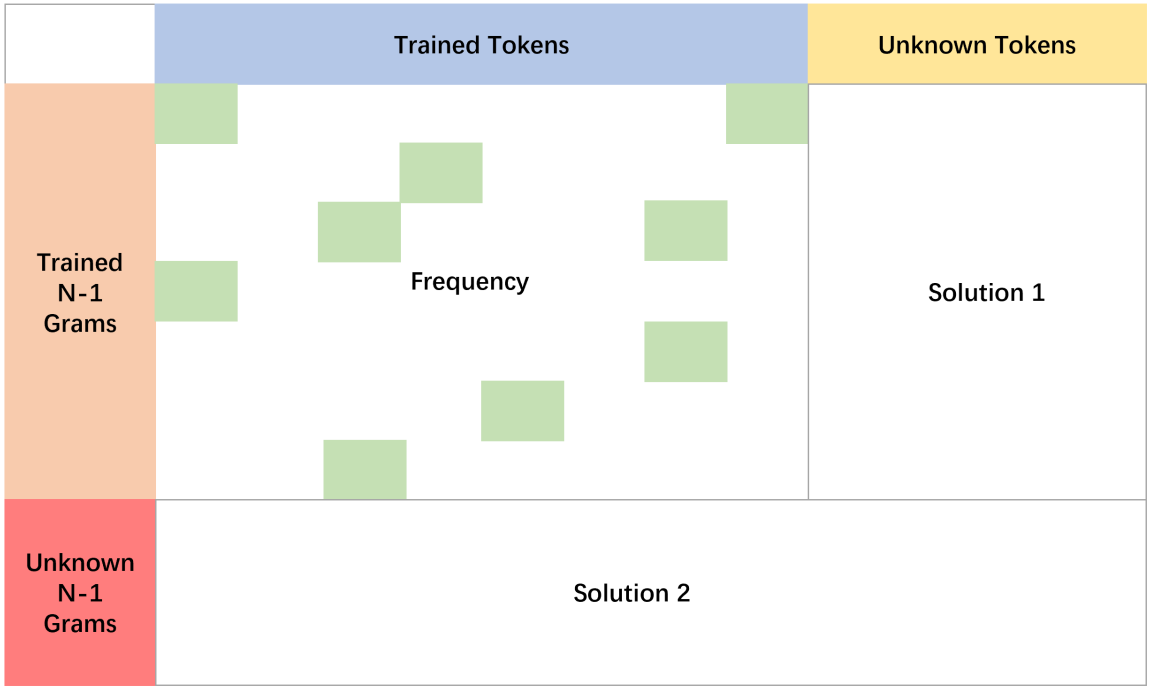
为了统计每种 N-Gram 的数量，本人在实验中采用的是 Python 中的双层字典数据结构，以期望能在接近常数的时间内索引频率并尽可能节约存储空间。具体来说，其逻辑结构如下图所示：



本人在使用了一个名为 base_word_count 的字典，其键为 (N-1)-Gram 对应的 tuple，其值均为一个子字典。子字典的键为 token，其值为一个整数。每当遇到形式为 $w_1, w_2, \cdots, w_n$ 的 N-Gram 时，我们就做对 $base\_word\_count[(w_1, w_2, \cdots, w_{n-1})][w_n]$ 的值加上一即可。值得指出的是，此时整个数据结构的大小仅与上图中绿色方框的个数成正比，于是大大减少了数据的存储空间。另外，本人的实现除了可以完成实验要求的 Bigram 和 Trigram 以外，任意的 N-Gram 统计模型都是容易实现的，仅需要改变参数即可。

c) 对于 unknown tokens 的处理

正如本次试验中对于测试数据集所描述的那样，我们的模型需要处理 unknown tokens，伴随着 unknown tokens 产生的还有 unknown (N-1)-Grams，如下图所示：



本人对于上图中的三个部分，分为三个不同的策略进行处理：

1. N-Gram 的前 (N-1)-Gram 曾学习过，最后一个单词也学习过：无论 N-Gram 是否出现过，采用平滑策略即可；
2. N-Gram 的前 (N-1)-Gram 曾学习过，但是最后一个单词未学习过（solution 1）：这种情况可以被化归为第一种情况进行处理，只需要在新的数据集上将全部的新 tokens 更新，并将它们全部视作没有出现的 N-Gram 即可。

   值得注意的是，为什么不“设置一个 unknown token，每次遇到未学习到的词汇就将其按照它归类呢”？本人的理由是，对于更完善，数据集更完整的模型而言，unknown tokens 的比例非常低，这种处理方法是合理的；但是在本实验中，根据实验表明，学习过的所有 tokens 的总数为 8w+ 而在整个测试集中，所有的 tokens 的数量达到了 10w+。如果将所有的 2w+ 的未学习 tokens 全部归类为一个 unknown token，这势必将大大增加所有未学习 tokens 计算出的概率（也会增加已学习 tokens 计算出的概率），这势必会不合理地降低模型的 perplexity，即困惑度；
3. N-Gram 的前 (N-1)-Gram 未学习过，且最后一个单词也未学习过（solution 2）：这种情况下，本人认为模型处于失效状态，但是本人认为可以做如下的补救：
   $P(w_n|w_1, w_2, \cdots, w_{n-1}) = P(w_n)$，即使用单词的词频来近似这一概率。本人认为这是非常不合理的行为，但是这实属无奈之举。类似的，对于词频为 0 的词语，本人采用平滑处理来估计其词频；

d) 平滑的设计

本质上来说，无平滑处理以及 Laplace 平滑处理都可以被视作加法平滑处理的特例，于是在实现时，本人将这一类方法统一为了加法平滑。

对于 Good-Turing 平滑，在样本数量非常少的时候，有可能出现“没有任何一种事件发生的频数为 1”的情况。查阅资料后，本人发现一般这种情况可以使用线性回归来估计，但是本人认为这样的做法估计成分过多。于是，对于 Good-Turing 平滑，本人仍然设置了加法平滑的参数，当出现“没有任何一种事件发生的频数为 1”的情况时，我们采用加法平滑来代替这一次 Good-Turing 平滑。

Part #3 实验结果与分析

在所有测试中，本人采用所有句子的 perplexity 的中位数作为衡量指标。

Bigram 测试结果

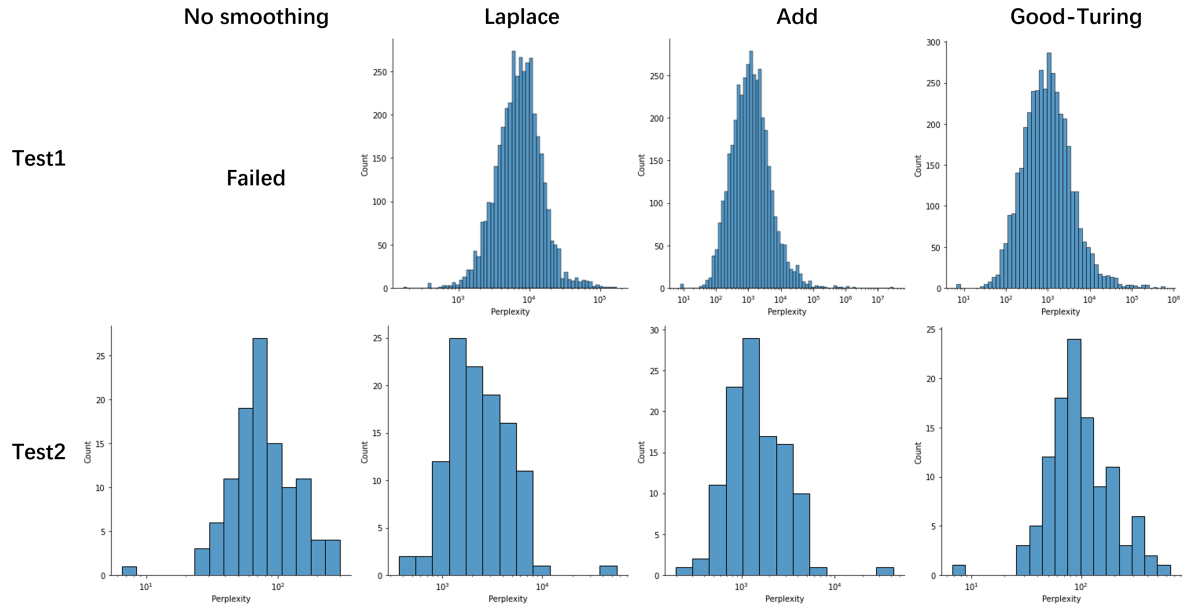
困惑度中位数*	不平滑	加一平滑	加法平滑	Good-Turing 平滑
test1	FAILED**	7289.02	1131.73 (2e-3)	932.06 (1e-2)
test2	72.31	2056.01	1293.48 (5e-1)***	90.66 (1e-1)

*所有结果保留 2 位小数，如果结果中出现括号，表示当前数值选择的加法平滑的参数，在没有特别说明时，所选参数最优，即使得困惑度最小；

**由于没有采用平滑，算法失效；

***由于test2的特殊性，导致不采用平滑算法仍然可以生效，此时不平滑的困惑度最低，所以最优的加法平滑将退化为不平滑，在此时本人考虑测试一个中间值来证实 “对于test2，困惑度与加法平滑参数正相关” 的猜想。

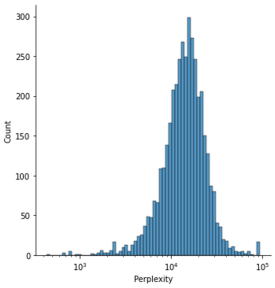
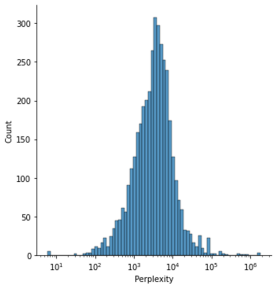
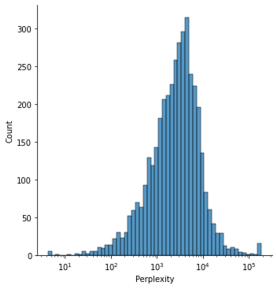
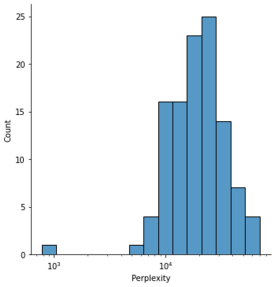
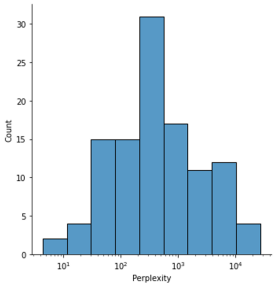
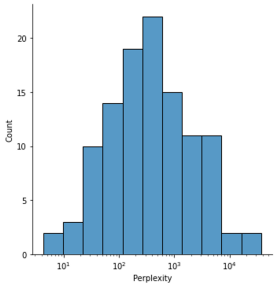
同时，本人给出所有测试中困惑度的分布如下表所示，注意横坐标采用了对数坐标：



Trigram 测试结果

困惑度中位数	不平滑	加一平滑	加法平滑	Good-Turing 平滑
test1	FAILED	14537.43	3497.27 (1e-3)	2798.50 (1e-5)
test2	FAILED	19324.87	419.48 (9e-5)	328.07 (4e-5)

给出所有测试中困惑度的分布如下表所示：

	No smoothing	Laplace	Add	Good-Turing
Test1	Failed			
Test2	Failed			

结果分析

1. 不平滑的策略在多数情况下是无效的（除了在精心筛选的 test2 数据集上）；但是，如果数据集足够完整以至于不会出现任何需要平滑的情况，不平滑的策略反而是困惑的最低的；
2. 在实验中，Laplace 平滑的效果总是最差的，本人认为原因在于，N-Grams 的总数量相比于 tokens 的总数量过少，导致简单的加一会使得很多信息被丢失，亦即本人认为，数据集需要在 tokens 总数固定的情况下进一步丰富，加一平滑的效果才会得到改善；
3. 加法平滑往往能大幅改善模型的效果，但是相比于加法模型中通常使用的参数 0.5，本人所取得的参数通常小若干个数量级，本人猜测这一原因类似于 2 中的分析；进一步地，本人猜想这一参数的最优值与 N-Grams 总数量和 tokens 总数量的比值有关；
4. Good-Turing 平滑的效果往往比加法平滑更好，其原因可能是为 0 频数事件分配的概率更合理；
5. 在加法或者 Good-Turing 平滑参数最优时，参数往往在中间值取得，本人认为原因是 perplexity 增加的主要原因有两种：一种是有频数的概率因为过平滑而概率下降过多，另一种是没有频数的概率因为欠平滑而概率过低。参数过小导致欠平滑，参数过大导致过平滑，当这两种情况达到平衡时，中位数最低；
6. 本次实验中，本人实现的 Trigram 效果没有 Bigram 好，本人认为原因主要有：一、测试数据集未知单词比例过多，导致 Trigram 大规模失效；二、训练数据集需要在 tokens 总数固定的情况下进一步丰富。