大作业一: 语言模型 discounting 算法

519030910383 张哲昊

1. 分词与概率字典建立

首先对于给定的训练数据首尾加上起止符，然后利用 nltk.tokenize 库中的的分词方法进行分词（TreebankWordTokenizer）。之后统计 1-5 grams 的出现次数，并将频率近似认为是概率构建 n-gram(1-5) 语言模型，核心代码如下：

```
sentence = open('train_set.txt').read()
tokenizer = TreebankWordTokenizer()
data = tokenizer.tokenize(sentence)
fdist_2 = FreqDist()
n=100
all_counts = dict()
for size in 1, 2, 3, 4, 5:
    all_counts[size] = FreqDist(ngrams(data, size)
    )
```

原始的 n-gram 模型存在数据稀疏的问题，即许多 gram 在训练集中出现的频率太低（大量存在仅仅出现 1-2 次的 gram），该情况同时会在 PPL 计算时导致不准确的问题。因此，接下来将只出现过一次的 gram 设置为 <UNK>。以 1-gram 为例，核心代码如下：

```
for key in all_counts[1].keys():
    Prob_1[key]=all_counts[1].freq(key)
min_value=min(Prob_1.values())
Prob_1['UNK']=0
for k in list(Prob_1.keys()):
    if Prob_1[k]==min_value:
        Prob_1.pop(k)
        Prob_1['UNK']+=min_value
```

2. Discounting 算法

2.1. Good Turing Discounting

该方法的基本的想法是将训练数据中出现次数为 r + 1 次的 n-gram 的概率重新分配给出现次数为 r 次的 n-gram。

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

其中 N_r 表示 count of counts 即在训练集合中出现次数是 r 次的 n-gram 的数量。从概率角度解释，即那些出现 r+1 次的 n-gram 重新分配给那些出现了 r 次的 n-gram，可以简单建模为如下公式。

$$P(X_r) \leftarrow P(X_{r+1}) \quad r = 1, ...K$$

其中 $P(X_r)$ 代表出现 r 次的 ngram 的频率之和。$P(X_r) = \sum_{i=0}^{N_r} P(x_r^i)$ 其中 $P(x_r^i)$ 为其中一个 ngram 的出现概率。将上述公式写成单个 ngram 的概率形式为：

$$P(x_r^i) = \frac{\sum_{i=0}^{N_{r+1}} P(x_{r+1}^i)}{N_r}$$

但是直觉上该方法将出现几率较高的 ngram 的概率分配给了出现几率较低的 ngram，与人们的直觉相违背，故只适用于低频词。本次项目将 Good Turing Discounting 算法进行相应改进，使得其更加符合人们的直觉，姑且命名为"α Discounting".

2.2. α Discounting

基于 Good Turing Discounting 的概率分配思想，本次项目从 $r = K$ 开始将概率分配方式进行了改变，基本思想为即将那些出现 r+1 次的 n-gram 的概率的 α 倍（$\alpha \in (0,1)$）重新分配给那些出现了 r 次的 n-gram，一直迭代到出现 0 次的 n-gram。具体算法伪代码如下：

其中，Redistribution 函数将出现次数为 r 的概率和分配给每一个 ngram。其中参数 α 为可调参数，甚至可以作为参数利用机器学习的方法进行学习。本次项目当中对 α 的值进行了不断的调整，比较了不同 α 下，测试集数据上的困惑度表现。

Algorithm 1: α Discounting 算法伪代码

Data: 原始 n-gram 概率 $P(X)$
初始化: K = small_count (超参数, 例如 5，4，3...)
r = K
while $r \leq 0$ **do**
  $temp = P(X_{r+1})$
  $P(X_{r+1}) \leftarrow (1-\alpha) * temp$
  $P(X_r) += \alpha * P(X_{r+1})$
  Redistribution$(P(X_{r+1}))$
  r=r-1

3. 实验结果与分析

3.1. 主要实验结果

本次项目不仅实现了上述 α Discounting($\alpha = 0.5$, $K = 4$) 算法，同时实现了 KneserNeyInterpolated, Laplace 平滑, WittenBellInterpolated, 普通最大似然这四种 discounting 算法[1]。以 3-gram 为例，实验结果如下表所示：可以从实验结果中发现，普通的极大似然法和 WittenBellInterpolated 均得到无穷打的困惑度，可以说明这两种方法无法解决测试数据中的 ngram 在训练集中出现频率过低的问题。而 α Discounting 与 KneserNey-Interpolate 能够有效解决该问题的出现，效果较好。

方法	困惑度
KneserNeyInterpolated	1903.27
MLE	∞
Laplace	51353.887
WittenBellInterpolated	∞
α Discounting	**140.86057**

3.2. 探索性实验结果

本次项目还对于 α Discounting 算法进行了 1-5gram 模型的实验结果对比，各个模型的困惑度如下图所示： 从实验结果可以看出，在没有回退策略的情况下，所使用的 gram 数量太大会使得大量没在训练集中出现的 gram 匹配到'<UNK>'，导致困惑度失准的情况。因此当 gram 较大时，使用回退策略是非常必要的。

[1]这四种方法使用了 nltk.lm 库进行训练，α Turing Discounting 则为本人实现

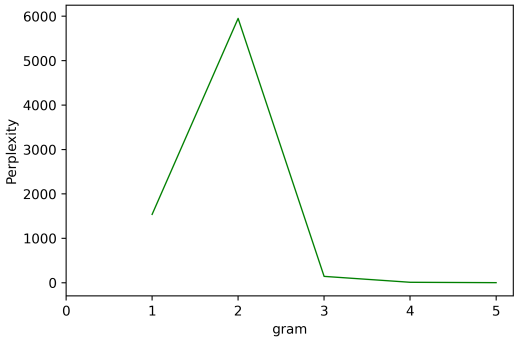

图 1: α *Discounting* 算法在 *1-5 gram* 模型上测试集的困惑度

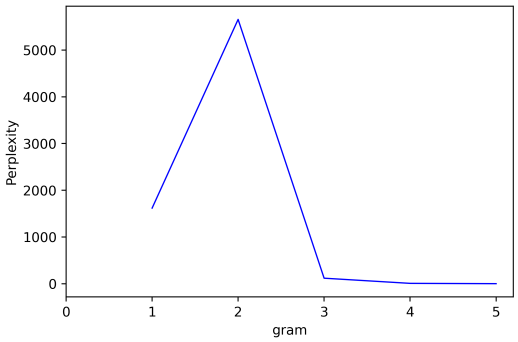

图 2: α *Discounting* 算法在 *1-5 gram* 模型上开发集的困惑度

4. 实验心得

本次实验加深本人对于 n-gram 语言模型的理解，以及对于 discounting 算法的认识。同时由于整个项目均为本人一**人**完成，对于代码能力也有极大的锻炼。