# ngram 报告

*520030910379 刘宜 rong*

## 1. 算法描述

### 1.1. 补全 learn 函数

learn 函数补全的部分需要我们先统计 gram 的频次，此过程比较简单，只需要遍历统计即可。代码如下

```
gram = stc[i-j-1:i]
gram = tuple(gram)
if gram not in self.frequencies[j].keys():
    self.frequencies[j][gram]=1
else:
    self.frequencies[j][gram]=self.frequencies[j][
    gram]+1
```

learn 函数需要补充的第二部分是完成对 $N_r$ 的计算，这一部分的代码，我们需要理解前面的对 frequencies 的排序，之后就能明白我们只需要按照 ncounts 的字典结构填入信息即可，在理解前面的代码后，这一部分也比较容易理解了。

```
for key, l in grams:
    gram = key[0]
    num = key[1]
    if gram not in self.ncounts.keys():
        self.ncounts[gram]={}
    self.ncounts[gram][num]=len(list(l))
```

### 1.2. 补全 d 函数

$$\lambda = \frac{N_1}{N_1 - (\theta+1)N_{\theta+1}}$$

$$d'(W_{k-n+1}^{k-1}, w_k) = \lambda \frac{(r+1)N_{r+1}}{rN_r} + (1-\lambda)$$

d 函数空缺的部分，我们只需要按照上面的公式计算即可得到 d'。

```
n = len(gram)-1
theta = self.discount_threshold
r = self.frequencies[n][gram]
if self.frequencies[n][gram]>theta:
    self._d[gram]=1
else:
    division = (self.ncounts[gram[:-1]][1]-(theta
    +1)*self.ncounts[gram[:-1]][theta+1])
    lamda = self.ncounts[gram[:-1]][1]/division
    self._d[gram] = lamda*(r+1)*self.ncounts[gram
    [:-1]][r+1]/(r*self.ncounts[gram[:-1]][r])+1-
    lamda
self._d[gram] = max(self._d[gram],self.eps)
```

此部分自己在写的时候出现了很多错误，在同学的帮助下，我添加了一些对特殊情况的修补然后完成了函数。这里只展示了主体的部分。

### 1.3. 补全 alpha 函数

$$\begin{aligned}\alpha(W_{k-n+1}^{k-1}) &= \frac{1-\sum_{w_k \in V_+} P(w_k|W_{k-n+1}^{k-1})}{\sum_{w_k \in V_-} P(w_k|W_{k-n+2}^{k-1})} \\ &= \frac{1-\sum_{w_k \in V_+} P(w_k|W_{k-n+1}^{k-1})}{1-\sum_{w_k \in V_+} P(w_k|W_{k-n+2}^{k-1})}\end{aligned}$$

这部分也需要按照公式的要求，把各个数据准备好带入公式即可。

```
if gram in self.frequencies[n-1]:
    # TODO: calculates the value of $\alpha$
    numerator = 1.
    denominator = 1.
    for key in self.frequencies[n].keys():
        if key[:-1]==gram:
            numerator = numerator - self[key]
            denominator = denominator - self[key
    [1:]]
    self._alpha[n][gram] = numerator/denominator
else:
    self._alpha[n][gram] = 1.
self._alpha[n][gram] = max(self._alpha[n][gram],
    self.eps)
```

### 1.4. 补全 getitem 函数

$$P(w_k|W_{k-n+1}^{k-1}) = \begin{cases} d(W_{k-n+1}^{k}) * \dfrac{C(W_{k-n+1}^{k})}{C(W_{k-n+1}^{k-1})} & C(W_{k-n+1}^{k}) > 0 \\ \alpha(W_{k-n+1}^{k-1}) * P(w_k|W_{k-n+2}^{k-1}) & \text{否则} \end{cases}$$

我们按照以上的公式计算即可，注意进行分类讨论。

```
if gram in self.frequencies[n].keys():
    self.disfrequencies[n][gram] = self.d(gram)*
     self.frequencies[n][gram]/self.frequencies[n
     -1][gram[:-1]]
else:
    self.disfrequencies[n][gram] = self.alpha(gram
     [:-1])*self[gram[1:]]
```

### 1.5. 补全 log_prob 和 ppl 函数

这一部分相对前几部分就比较容易理解和实现了，只需要套用很简单的两个公式即可。但是这里我认为有时候数值会过于小，此时会比较影响模型的效果，因此当数值过于小时，将其拉高至 eps 的值，这算是一个小小的补丁。

```
for i in range(2, len(sentence)+1):
# TODO: calculates the log probability
    if i == 2:
        result = self[tuple([sentence[1]])]
    elif i==3:
        result = self[tuple(sentence[1:3])]
    elif i==4:
        result = self[tuple(sentence[1:4])]
    else:
        result = self[tuple(sentence[i-4:i])]
    result = max(result,self.eps)
    log_prob = log_prob + math.log2(result)
log_prob = log_prob/(len(sentence)-1)
return log_prob
```

```
return 2**(-self.log_prob(sentence))
```

## 2. 结果

最终结果如下

```
Loaded model.
17001.1902360811
753.7819437601232
6918.048768216905
1364.4434372326446
2589.6954856683374
196.20769435267388
1872227.061014132
617.613981950264
2803.8938893346385
7103.177653488322
225.6720917583247
740.7908737953668
16501.103354546336
6714.945219057215
17241.55231251803
17033.714646066026
3049.4258475981983
33874.771656875106
7060.26314522653
270.19303288236847
1800.2692883588759
2353.7786674091503
9467.85840321846
36519.12817634024
2868.389113724471
11691.317273335724
3243.8259896818854
422.6863766013103
26143.76500373065
413.93044818389507
8727.005011398816
14933.574456270848
6887.087255494094
3032.885664585309
1048.6138457047903
7570.169700701642
12147.329444681445
3088.1696455773754
11037.078280473028
3463.313962240211
6877.307472529193
4484.895871619789
2171.1167745270905
160364.87083905528
2957.0377403827365
19439.948624203815
818.0542776853721
2196.816124730158
93561.01035268964
39869.733438004565
Avg:  50277.770276153606
```