N-gram 计算困惑度

班级：2016211303　　姓名：黄若鹏　　学号：2016212901

1.题目要求

- N-gram language Models： 30 points

-In this assignment you will explore a simple, typical N-gram language model.
-This model can be trained and tested on sentence-segmented data of a Chinese text corpus. “Word Perplexity” is the most widely-used evaluation metric for language models.
-Additional points: if you can test how does the different “Word Perplexity” of the different “N” grams, you will get addition 10 points.
-Additional points: if you can test how does the different “Word Perplexity” of the different smoothing methods, you will get additional 10 points.

2.运行环境

操作系统：windows10

使用软件：VScode

3.模型建立

3.1 N-gram 语言模型

语言模型（Language Model,LM）的一个常见任务，是已知一句话的前面几个词，预测下一个是什么，即对 $P(w_i|w_1^{i-1})$ 建模

N-gram 语言模型，是基于 Markov 假设，假设文本中的每个词只与前面的 n-1 个词有关，即

$$P(w_i|w_1^{i-1}) \approx P(w_i|w_{i-n+1}^{i-1}) = P(w_i|w_{i-1},\ldots,w_{i-n+1})$$

这可以通过对训练语料做极大似然估计

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{Count(w_i, w_{i-1}, \ldots, w_{i-n+1})}{Count(w_{i-1}, \ldots, w_{i-n+1})}$$

3.2 Perplexity（困惑度）

刚才我们通过训练集得到了语言模型，而 perplexity 是一种评价语言模型在测试集上表现的方法
对一句句子来说，

$$Perplexity(s) = P(s)^{-\frac{1}{N+1}} = 2^{-\frac{1}{N+1}\cdot \log P(s)}$$

对于 bigram LM 来说，就是

$$\sqrt[N+1]{\frac{1}{P(w_1|<\mathrm{s}>)P(w_2|w_1)\ldots P(w_N|w_{N-1})P(</\mathrm{s}>|w_N)}}$$

对于整个测试集，我们再对所有句子的 perplexity，求几何平均，得到整体的结果
这里用N′表示所有测试集中句子长度之和，即 $N' = \sum(N_k + 1)$,

$$Perplexity = P(S)^{-\frac{1}{N'}} = 2^{-\frac{1}{N'}\cdot \log P(S)} = 2^{-\frac{\sum \log P(s_k)}{\sum(N_k+1)}}$$

解释

注意上面的指数表达形式，其 $-\frac{1}{N'}\log p(S)$ 中 可以理解为 (对词平均的) 交叉熵（cross-entropy)，也就是 $H(q,p) = -\sum q(w)\log p(w)$

LM 拟合得越好，即模型越贴近真实分布，perplexity（交叉熵）越小，KL 散度越小，越接近真实分布的熵

$$H(q,p) = \mathbb{E}_q[-\log p] = H(q) + D_{KL}(q\|p) \geq H(q)$$

3.3 平滑方法

3.3.1．Laplace 平滑（add-one, add-α)

$$p = \frac{c+\alpha}{n+\alpha v}$$

$$\text{其中 } 0 \leq \alpha \leq 1,\ v = |V|$$

- $\alpha = 0$时，即为不做平滑的结果
- $\alpha = 1$时，即为常说的add-one

3.3.2.Good-Turing Smoothing

- 假设语料中出现了r次的词有N_r（出现r次的词的集合大小），语料大小为N，则$N = \sum_{r=1}^{\infty} rN_r$
  - 考虑unigram（n=1），出现r次的所有词，其概率为$\frac{r}{N}$
- 当r较小时，极大似然估计可能不准确，同时我们也要考虑一下那些没有出现（$r = 0$）的词，从而我们给所有r打一个“折扣”（discount），

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

  容易证明，$N = \sum_{r=0}^{\infty} d_r N_r$
- 根据Zipf’s law，r越大，N_r越小，所以，一般情况下，$r^* < r$
- 可以证明，$d_r \approx E(r) = E(c^*(w)|c(w) = r)$
  - 因为有未知的信息（unseen ngram），所以观测的统计量的方差较大（但仍是无偏的），所以设计一个条件概率来减小方差（?）

3.4 输入输出

输入：'1998-01-105-带音.txt' 可以每次运行，重新生成，训练集 train.txt 和测试集 test.txt
无需自己设置训练集和测试集
输出：unigram、bigrams、trigrams 模型的困惑度

3.5 实现过程

先是清洗数据，得到只有词的数据，之后再按 80%为训练集，20%为测试集来划分数据。

实现代码

```
def partition(self):
        for line in self.filename.readlines():
            line = re.sub(r"/\w*","",line)
            line = re.sub((r"[%s]+"%(punctuation)), "", line)
            line = re.sub(r"\d*","",line)
            line = re.sub(r"-*","",line)
            line = re.sub(r"{.*}","",line)
            line = re.sub(r"\]\w*","",line)
            line = re.sub(r"\[","",line)
            self.datalist.append(line)

        c_train, c_test =
cross_validation.train_test_split(self.datalist, test_size=0.2)
        for i in c_train:
            self.out_train.write(''.join(i)+'\n')

        for i in c_test:
            self.out_test.write(''.join(i)+'\n')
def __init__(self):
        self.out_train = open('train.txt', 'w',encoding='UTF-8')
        self.out_test = open('test.txt', 'w',encoding='UTF-8')
        self.filename = open('1998-01-105-带音.txt','r',encoding='UTF-8')
        self.datalist = []
```

之后在使用 n-gram 模型，Laplace 平滑后计算困惑度

实现代码

```
#2-gram 模型
    def big_gram_laplace_smooth(self):
        v = len(self.bigrams_test)
        self.probability = 0


        for i,j in zip(self.unigram_test,self.bigrams_test):
            if j not in self.bigram_data:
                self.bigram_data[j] = 0
            if i not in self.unigram_data:
                self.unigram_data[i] = 0
```

```
            self.probability   +=   (math.log2((self.bigram_data[j]+1)   /
(self.unigram_data[i]+v)))*((self.bigram_data[j]+1)                    /
(self.unigram_data[i]+v))
        self.perplexity=-self.probability
        print("2-gram 困惑度：",self.perplexity)
```

4.结果分析

测试了多次，这里截了三次连续测试的图。

```
PS C:\Users\Rocair\Desktop\finalNLP\N-Gram> cd
s\ms-python.python-2018.12.1\pythonFiles\ptvsd_
1-gram 困惑度: 3266.8999821732255
2-gram 困惑度:  237.87709197908367
3-gram 困惑度:  49.25925263707749
PS C:\Users\Rocair\Desktop\finalNLP\N-Gram> cd
s\ms-python.python-2018.12.1\pythonFiles\ptvsd_
1-gram 困惑度: 3363.3452559002444
2-gram 困惑度:  236.34561214877658
3-gram 困惑度:  48.97806425854142
PS C:\Users\Rocair\Desktop\finalNLP\N-Gram> cd
s\ms-python.python-2018.12.1\pythonFiles\ptvsd_
1-gram 困惑度: 3383.8884989267017
2-gram 困惑度:  233.4905459927504
3-gram 困惑度:  47.65436261453146
```

结果符合预期，随着 n 的增大，困惑度在降低，并且，降低的幅度越来越小。这也符合课件里一般 n 也不会取很大，一来计算复杂度会随着 n 的增大而增大，并且提升的效果 也并没显著提升，一般 n 取小于 6。

5.补充说明

如果这里 import 出问题

```
from sklearn.model_selection  import train_test_split
```

可以改写成

```
from sklearn import cross_validtion
```

并且将

```
c_train, c_test = train_test_split(self.datalist, test_size=0.2)
```

改成

```
c_train, c_test = cross_validtion.train_test_split(self.datalist,
test_size=0.2)
```

这个是划分训练集和测试集的函数，这里 import 包花了不少功夫，感觉 python 太多东西，由于版本问题，可移植性有点弱啊（也可能是我 python 用得少，不熟吧）