N-gram Language Models说明文档

1. 任务定义

设计一个典型的N-gram语言模型。用所给的已分词数据一部分训练得到该模型，用另一部分数据测试该模型。最后用Word Perplexity评价该模型。本文设计的是采用Lidstone（add-Delta）平滑稀疏数据的unigram、bigram和trigram，通过对不同n在不同的Delta下的word perplexity值进行评价，从而选出不同的n对应的最优Delta。

1. 输入输出

输入为nlpdata.txt（所给的语料集）。

输出为unigram、bigram、trigram分别对应的最优Delta，当前的word perplexity。

delta=0.000092

Trigram Best

wp(Unigram)=7945.517950

wp(Bigram)=3321.947539

wp(Trigram)=1988.857597

delta=0.002000

Bigram Best

wp(Unigram)=6769.388344

wp(Bigram)=2187.795216

wp(Trigram)=2958.880434

delta=1.000000

Unigram Best

wp(Unigram)=4976.329345

wp(Bigram)=11568.252959

wp(Trigram)=20976.149275

1. 源码运行环境

使用python 3.6进行编写，在Windows 10上运行通过。源代码为ngram.py。如果需要测试寻找最优Delta的过程，还需要导入matplotlib.pyplot（即将注释掉的一句import加上）。

1. 方法描述

* 数据处理

观察所给的语料集，其中：

每一条语料前面都有一个日期；（日期为0-9 与 '-' 组成）

每一条语料内还有换行；（换行为'\r\n'）

每一个分好的词后都有一个词性标注；（由'\'与英文字母组成）

有的词用[]括起来，其中右括号后面跟着一个nt；

有的词还会有拼音标注，{}括号内为拼音。（括号内由英文字母与数字组成）

其中不需要日期，拼音标注。词性标注在此认为需要保留，认为动词与名词之间有区别。使用正则表达式，根据其数据特征，将日期，拼音标注滤去，完成数据清洗。取前80%作为训练集，后20%作为测试集。因为需要测定不同Delta下的word perplexity值，所以在此固定训练集与测试集，并不采用随机挑选的方式。

对于一个ngram，w1w2w3…wn，认为这是一个句子内部的语句（或称为词），即w1w2w3…wn中间不会被分隔开，即中间不会出现'，' '。' '；' '！' '：'等句子的分割标志，如若出现，将分割标志替换为n-1个flag，表示这是一个新的句子。进行前面的处理是因为，假设认为不同句子之间通过分割标志相邻的词没有联系。

* 构造ngram模型

对于unigram模型，unigram数据集直接等价于训练集数据，因为unigram是只计算单个词出现的概率。最后统计其中每个词出现的频次，存到cnt\_unigram字典里。

对于bigram模型，bigram数据集的每一个元素由两个词组成，取训练集中每两个相邻的作为一个元素，在python中实现如下：

seg\_bigram = [(train1[i],train1[i+1]) **for** i **in** range(len(train1)-1)]

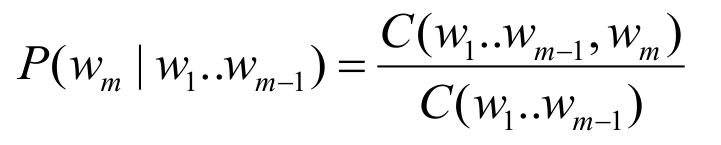
然后统计seg\_bigram中每个元素的频次，即每两个相邻词出现的频次，存到cnt\_bigram里。

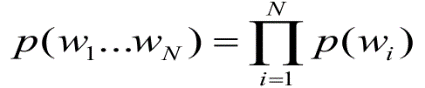
对于trigram模型，trigram数据集的每一个元素由三个词组成，取训练集中每三个相邻的作为一个元素，在python中实现如下：

seg\_trigram = [(train2[i],train2[i+1],train2[i+2]) **for** i **in** range(len(train2)-2)]

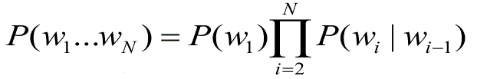
然后统计seg\_trigram中每个元素的频次，即每三个相邻词出现的频次，存到cnt\_trigram里。

* 测试ngram模型并计算word perplexity

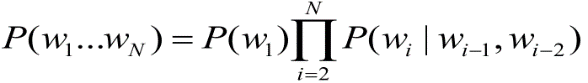
由上一步将会得到每个模型对应的频次字典，此时我们可以计算某个gram出现的概率。计算公式如下：



对于unigram，有

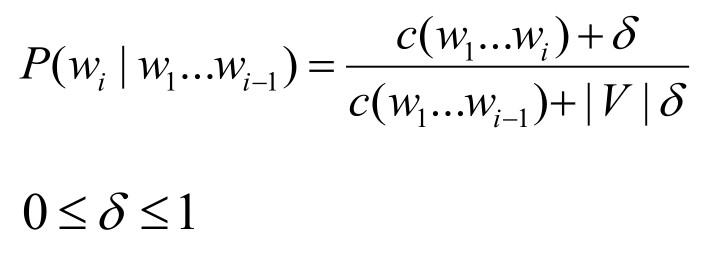


对于bigram,有



对于trigram,有

由以上P算出的word perplexity巨大，因为其中某个句子未出现过时，概率将会为0。故考虑使用add-Delta平滑稀疏数据。此时某个gram出现的概率变为：

其中V为语料库大小，即语料库中不同词的个数。

最后看一下word perplexity的定义：

可以理解为先计算平均信息量I，再对其取2的幂次，其中n是测试集大小，P（wi）为每一个测试集中元素的出现概率。

有了以上基础，可以描述计算word perplexity的算法如下(以trigram为例)：

for i in 测试集:

n += 1

tri = 该元素在seg\_trigram中出现频次，若i在trigram训练集中出现过

否则 0

bi = 该元素前两位在seg\_bigram出现频次，若i前两位在bigram训练集中出现过

否则 0

P = (tri+Delta)/(bi+V\*Delta)

I +=

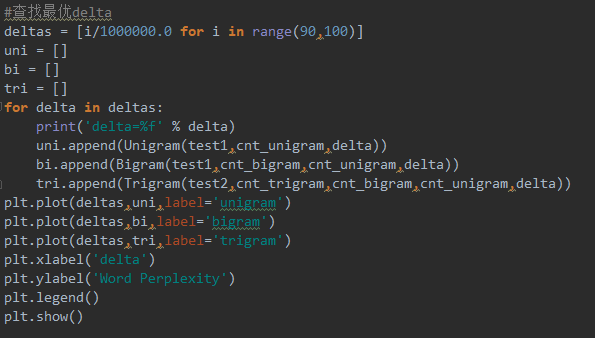
Exp = -I/n

wp = 2^exp

至此已求得wp，最后需要对不同的Delta下ngram的最优值进行查找。

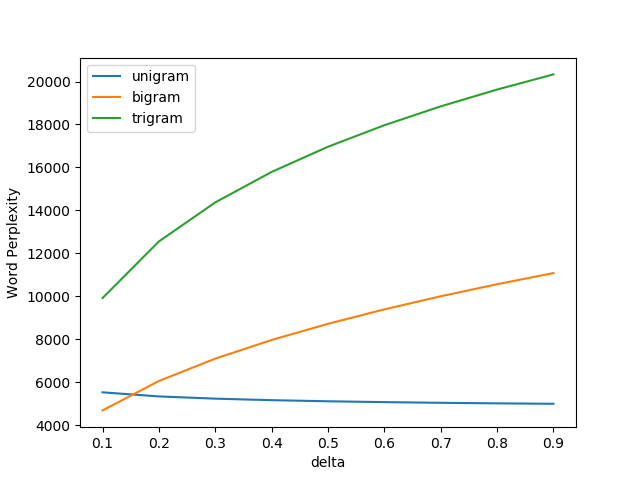
* 最优Delta的查找

选择一步步逼近的方法，即一开始从0~1，到0~0.1，0~0.01，直到找到最优值。以下为代码部分，需要导入matplotlib.pyplot。将deltas这个list的range不断改变，每次得到10个坐标点，绘图表示观察，来寻找到最优值。由于已找到Delta，故将这部分注释掉（可以解开注释运行观察，需要把import里那句注释也解开），过程在下一步描述。



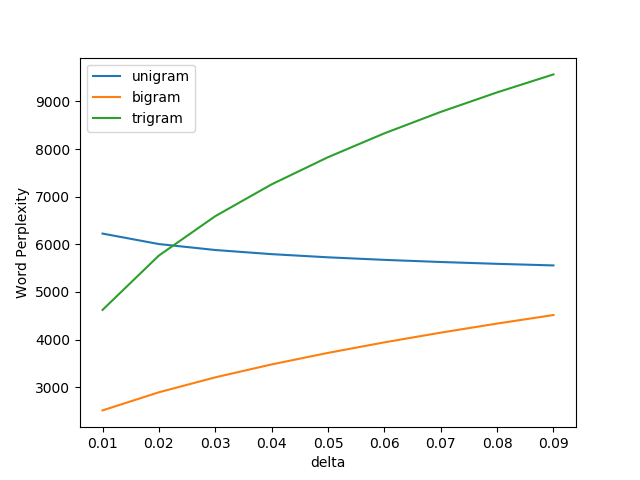
1. 结果分析

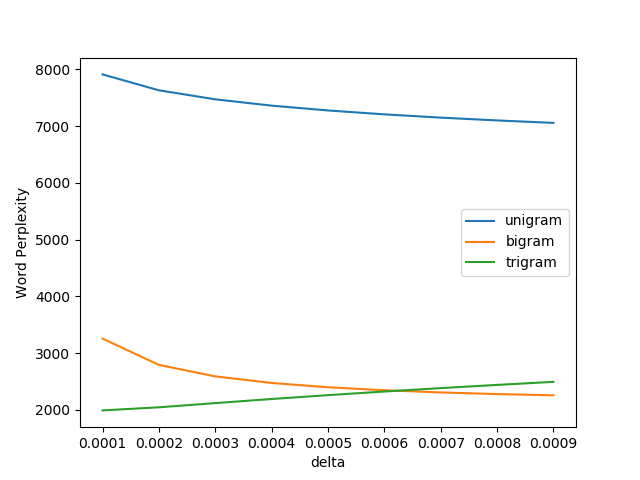
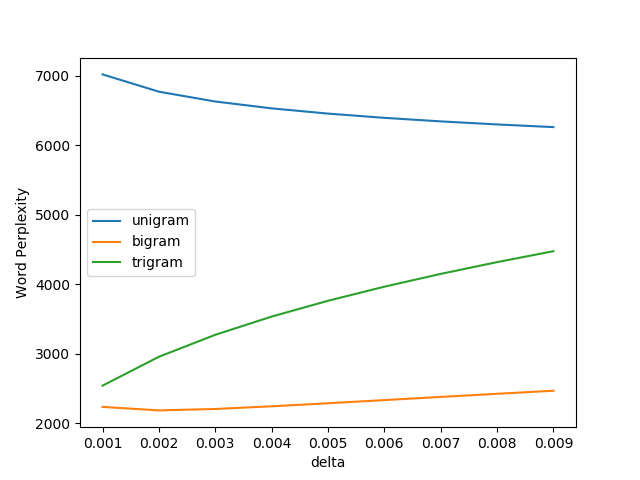
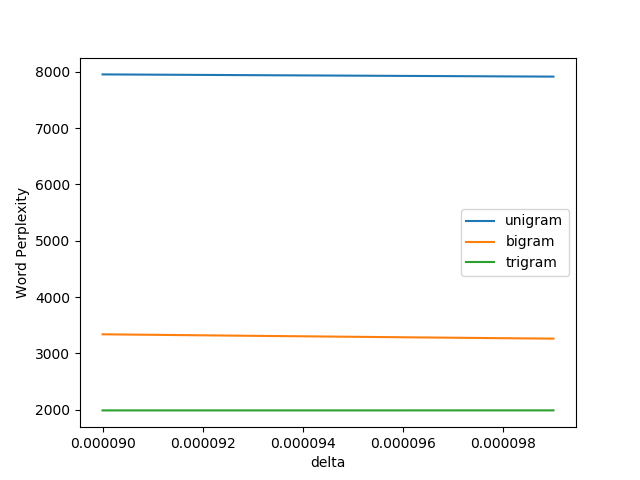
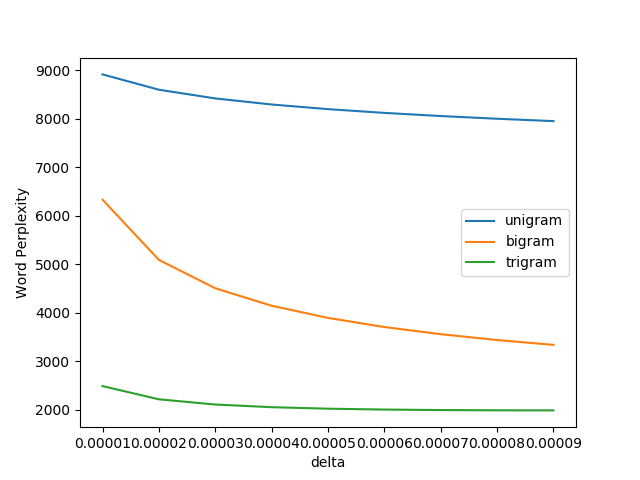
首先需要找出Lidstone方法平滑数据的Delta的最优值。

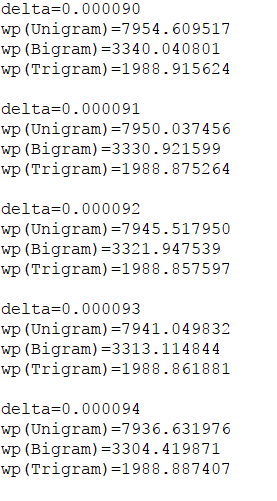
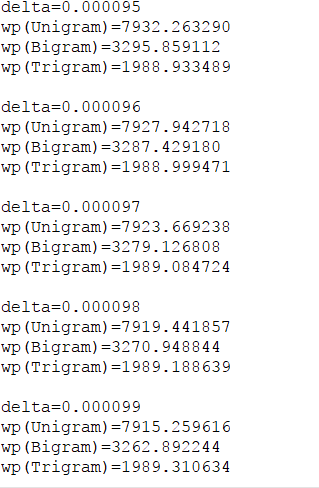
* 将Delta从(0,1)以0.1为间隔进行观察：

可以看到，在Delta大于大概0.15时，unigram模型的性能最优，而unigram模型在ngram模型中又是性能最差的，故直接考虑Delta在0~0.1时的情况。

* 再将Delta从(0,0.1)以0.01为间隔进行观察：

可以发现trigram模型的word perplexity可能在Delta小于0.01时，比bigram要好。于是不断逼近Delta。

* 下图依次是Delta从0~0.01，0~0.001，0~0.0001时，三个模型word perplexity值的变化，可以观察到trigram模型word perplexity值在0.00009~0.0001之间取到最小值，再以0.000001为精度，取这之间十个值进行观察：

将最后一张图的Delta打印出来观察，有

* N-gram模型在add-Delta平滑下的word-perplexity表现：

对于trigram模型，可以看出，Delta以0.000001为精度时，Delta取到0.000092时，trigram模型最优，word perplexity=1988.86。

对于bigram模型，可以看出，Delta 大约在 0.002左右时，bigram模型最优，word perplexity=2187.79。

对于unigram模型，可以看出，unigram模型对应的word perplexity函数在delta∈(0,1)之间一直递减，应该是在1处取得最优值，word perplexity = 4976.33。