LDA的perplexity计算

**评估LDA主题模型-perflexity**

LDA主题模型好坏的评估，判断改进的参数或者算法的建模能力。

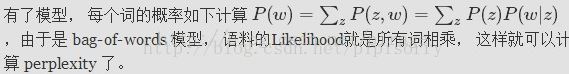
**perplexity is only a crude measure, it's helpful (when using LDA) to get 'close' to the appropriate number of topics in a corpus.**

Blei先生在论文《Latent Dirichlet Allocation》实验中用的是Perplexity值作为评判标准，并在论文里只列出了perplexity的计算公式。



**Note**:M代表测试语料集的文本数量，Nd代表第d篇文本的大小（即单词的个数），P(Wd)代表文本的概率

**文本的概率的计算：**



p(z)表示的是文本d在该主题z上的分布，应该是p(z|d)

**Note：**

1. **Blei是从每篇文本的角度来计算perplexity的，而上面是从单词的角度计算perplexity。**

2. 测试文本集中有M篇文本，对词袋模型里的任意一个单词w，P(w)=∑z p(z|d)\*p(w|z)，即该词在所有主题分布值和该词所在文本的主题分布乘积。

3. 模型的perplexity就是exp^{ - (∑log(p(w))) / (N) }，∑log(p(w))是对所有单词取log（直接相乘一般都转化成指数和对数的计算形式），N的测试集的单词数量（不排重）

4. P(w)=∑z p(z|d)\*p(w|z)这个w是测试集上的词汇

困惑度一般在自然语言处理中用来衡量训练出的语言模型的好坏。在用LDA做主题和词聚类时，原作者D.Blei就是采用了困惑度来确定主题数量。文章中的公式为：

perplexity=exp^{ - (∑log(p(w))) / (N) }

其中，P(W)是指的测试集中出现的每一个词的概率，具体到LDA的模型中就是P(w)=∑z p(z|d)\*p(w|z)【z,d分别指训练过的主题和测试集的各篇文档】。分母的N是测试集中出现的所有词，或者说是测试集的总长度，不排重。

因而python程序代码块需要包括几个方面：

1.对训练的LDA模型，将Topic-word分布文档转换成字典，方便查询概率，即计算perplexity的分子

2.统计测试集长度，即计算perplexity的分母

3.计算困惑度

4.对于不同的Topic数量的模型，计算的困惑度，画折线图。

python代码如下：

# -\*- coding: UTF-8-\*-

import numpy

import math

import string

import matplotlib.pyplot as plt

import re

def dictionary\_found(wordlist): #对模型训练出来的词转换成一个词为KEY,概率为值的字典。

word\_dictionary1={}

for i in xrange(len(wordlist)):

if i%2==0:

if word\_dictionary1.has\_key(wordlist[i])==True:

word\_probability=word\_dictionary1.get(wordlist[i])

word\_probability=float(word\_probability)+float(wordlist[i+1])

word\_dictionary1.update({wordlist[i]:word\_probability})

else:

word\_dictionary1.update({wordlist[i]:wordlist[i+1]})

else:

pass

return word\_dictionary1

def look\_into\_dic(dictionary,testset): #对于测试集的每一个词，在字典中查找其概率。

'''Calculates the TF-list for perplexity'''

frequency=[]

letter\_list=[]

a=0.0

for letter in testset.split():

if letter not in letter\_list:

letter\_list.append(letter)

letter\_frequency=(dictionary.get(letter))

frequency.append(letter\_frequency)

else:

pass

for each in frequency:

if each!=None:

a+=float(each)

else:

pass

return a

def f\_testset\_word\_count(testset): #测试集的词数统计

'''reture the sum of words in testset which is the denominator of the formula of Perplexity'''

testset\_clean=testset.split()

return (len(testset\_clean)-testset.count("\n"))

def f\_perplexity(word\_frequency,word\_count): #计算困惑度

'''Search the probability of each word in dictionary

Calculates the perplexity of the LDA model for every parameter T'''

duishu=-math.log(word\_frequency)

kuohaoli=duishu/word\_count

perplexity=math.exp(kuohaoli)

return perplexity

def graph\_draw(topic,perplexity): #做主题数与困惑度的折线图

x=topic

y=perplexity

plt.plot(x,y,color="red",linewidth=2)

plt.xlabel("Number of Topic")

plt.ylabel("Perplexity")

plt.show()

topic=[]

perplexity\_list=[]

f1=open('/home/alber/lda/GibbsLDA/jd/test.txt','r') #测试集目录

testset=f1.read()

testset\_word\_count=f\_testset\_word\_count(testset) #call the function to count the sum-words in testset

for i in xrange(14):

dictionary={}

topic.append(5\*(3i+1)) #模型文件名的迭代公式

trace="/home/alber/lda/GibbsLDA/jd/stats/model-final-"+str(5\*(i+1))+".txt" #模型目录

f=open(trace,'r')

text=f.readlines()

word\_list=[]

for line in text:

if "Topic" not in line:

line\_clean=line.split()

word\_list.extend(line\_clean)

else:

pass

word\_dictionary=dictionary\_found(word\_list)

frequency=look\_into\_dic(word\_dictionary,testset)

perplexity=f\_perplexity(frequency,testset\_word\_count)

perplexity\_list.append(perplexity)

graph\_draw(topic,perplexity\_list)

下面是画出的折线图，在拐点附近再调整参数（当然与测试集有关，有图为证～～），寻找最优的主题数。实验证明，只要Topic选取数量在其附近，主题抽取一般比较理想。

