**Gensim应用**

引用gensim包，gensim包中引用corpora,models, similarities，分别做语料库建立，模型库和相似度比较库from gensim import corpora, models, similarities

我调用了结巴分词做中文处理，所以同样import jieba

手工写个文本列表：sentences = ["我喜欢吃土豆","土豆是个百搭的东西","我不喜欢今天雾霾的北京"]

用结巴分词后待用，因为gensim包做主题模型，在意的是语料库，所以，中文英文，one-term，two-term都是无所谓的，如果有已经生成好的语料库，那么可以考虑直接跳到建模环节。官方提供的语料库范例是这样的：

corpus = [ [(0, 1.0), (1, 1.0), (2, 1.0)],

[(2, 1.0), (3, 1.0), (4, 1.0), (5, 1.0), (6, 1.0), (8, 1.0)],

[(1, 1.0), (3, 1.0), (4, 1.0), (7, 1.0)],

[(0, 1.0), (4, 2.0), (7, 1.0)]]

每个中括号代表一句话，用逗号隔开，（0，1.0）代表词典中编号为0的词出现了一次，以此类推，

回到过程中来，将范例的语句分词

words=[]  
 for doc in sentences:  
    words.append(list(jieba.cut(doc)))  
 print words

输出：

[[u'\u6211', u'\u559c\u6b22', u'\u5403', u'\u571f\u8c46'], [u'\u571f\u8c46', u'\u662f', u'\u4e2a', u'\u767e', u'\u642d', u'\u7684', u'\u4e1c\u897f'], [u'\u6211', u'\u4e0d', u'\u559c\u6b22', u'\u4eca\u5929', u'\u96fe', u'\u973e', u'\u7684', u'\u5317\u4eac']]

此时输出的格式为unicode，不影响后期运算，因此我保留不变，如果想看分词结果可以用循环输出jieba分词结果,得到的分词结果构造词典:

dic = corpora.Dictionary(words)  
 print dic  
 print dic.token2id

输出：

Dictionary(15 unique tokens)  
{'\xe5\x8c\x97\xe4\xba\xac': 12, '\xe6\x90\xad': 6, '\xe7\x9a\x84': 9, '\xe5\x96\x9c\xe6\xac\xa2': 1, '\xe4\xb8\x8d': 10, '\xe4\xb8\x9c\xe8\xa5\xbf': 4, '\xe5\x9c\x9f\xe8\xb1\x86': 2, '\xe9\x9c\xbe': 14, '\xe6\x98\xaf': 7, '\xe4\xb8\xaa': 5, '\xe9\x9b\xbe': 13, '\xe7\x99\xbe': 8, '\xe4\xbb\x8a\xe5\xa4\xa9': 11, '\xe6\x88\x91': 3, '\xe5\x90\x83': 0}  
可以看到各个词或词组在字典中的编号,为了方便看，我给了个循环输出：

for word,index in dic.token2id.iteritems():  
    print word +" 编号为:"+ str(index)

输出：

北京 编号为:12  
搭 编号为:6  
的 编号为:9  
喜欢 编号为:1  
不 编号为:10  
东西 编号为:4  
土豆 编号为:2  
霾 编号为:14  
是 编号为:7  
个 编号为:5  
雾 编号为:13  
百 编号为:8  
今天 编号为:11  
我 编号为:3  
吃 编号为:0

词典生成好之后，就开始生成语料库了:

corpus = [dic.doc2bow(text) for text in words]  
 print corpus

输出：

[[(0, 1), (1, 1), (2, 1), (3, 1)], [(2, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1)], [(1, 1), (3, 1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1)]]

语料库的官方描述写的是向量空间模型格式的语料库，Corpus is simply an object which, when iterated over, returns its documents represented as sparse vectors.

官方说明给了一个tips：

In this example, the whole corpus is stored in memory, as a Python list. However,the corpus interface only dictates that a corpus must support iteration over its constituent documents. For very large corpora, it is advantageous to keep the corpus on disk, and access its documents sequentially, one at a time. All the operations and transformations are implemented in such a way that makes them independent of the size of the corpus, memory-wise.

大概意思就是说范例中的语料库是存在内存中的一个列表格式，但是接口对语料库的要求只是，支持在构建文本文件中可循环即可，因此对于很大的语料库，最好还是存在硬盘上，然后依次访问文件。这样可以不用考虑语料库的size，也避免了内存占用太多.

此时，得到了语料库，接下来做一个TF-IDF变换,可以理解成 将用词频向量表示一句话 变换成为用 词的重要性向量表示一句话（TF-IDF变换：评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在[语料库](http://baike.baidu.com/view/686705.htm" \t "http://blog.csdn.net/whzhcahzxh/article/details/_blank)中出现的频率成反比下降。）

tfidf = models.TfidfModel(corpus)

vec = [(0, 1), (4, 1)]  
print tfidf[vec]  
corpus\_tfidf = tfidf[corpus]  
for doc in corpus\_tfidf:  
    print doc

输出：

[(0, 0.7071067811865475), (4, 0.7071067811865475)]  
[(0, 0.8425587958192721), (1, 0.3109633824035548), (2, 0.3109633824035548), (3, 0.3109633824035548)]  
[(2, 0.16073253746956623), (4, 0.4355066251613605), (5, 0.4355066251613605), (6, 0.4355066251613605), (7, 0.4355066251613605), (8, 0.4355066251613605), (9, 0.16073253746956623)]  
[(1, 0.1586956620869655), (3, 0.1586956620869655), (9, 0.1586956620869655), (10, 0.42998768831312806), (11, 0.42998768831312806), (12, 0.42998768831312806), (13, 0.42998768831312806), (14, 0.42998768831312806)]  
vec是查询文本向量，比较vec和训练中的三句话相似度

index = similarities.SparseMatrixSimilarity(tfidf[corpus], num\_features=14)  
sims = index[tfidf[vec]]  
print list(enumerate(sims))

输出：

[(0, 0.59577906), (1, 0.30794966), (2, 0.0)]

表示和第1句话相似度为59.578%，和第二句话的相似度位30.79%，第三句没有相似度，

我们看看vec这句话是什么：0为吃，4为东西，所以vec这句话可以是["吃东西"]或者["东西吃"]

而第一句话"我喜欢吃土豆","土豆是个百搭的东西"明显有相似度，而第三句话"我不喜欢今天雾霾的北京"，相似度几乎为0，至于为什么第一句比第二句更相似，就需要考虑TfIdf document representation和cosine similarity measure了

回到tfidf转换，接着训练LSI模型，假定三句话属于2个主题，

lsi = models.LsiModel(corpus\_tfidf, id2word=dic, num\_topics=2)  
lsiout=lsi.print\_topics(2)  
print lsiout[0]  
print lsiout[1]

输出：

0.532\*"吃" + 0.290\*"喜欢" + 0.290\*"我" + 0.258\*"土豆" + 0.253\*"霾" + 0.253\*"雾" + 0.253\*"北京" + 0.253\*"今天" + 0.253\*"不" + 0.166\*"东西"  
0.393\*"百" + 0.393\*"搭" + 0.393\*"东西" + 0.393\*"是" + 0.393\*"个" + -0.184\*"霾" + -0.184\*"雾" + -0.184\*"北京" + -0.184\*"今天" + -0.184\*"不"

这就是基于SVD建立的两个主题模型内容

将文章投影到主题空间中

corpus\_lsi = lsi[corpus\_tfidf]  
for doc in corpus\_lsi:  
    print doc

输出：

[(0, -0.70861576320682107), (1, 0.1431958007198823)]  
[(0, -0.42764142348481798), (1, -0.88527674470703799)]  
[(0, -0.66124862582594512), (1, 0.4190711252114323)]

因此第一三两句和主题一相似，第二句和主题二相似

同理做个LDA

lda = models.LdaModel(corpus\_tfidf, id2word=dic, num\_topics=2)  
ldaOut=lda.print\_topics(2)  
print ldaOut[0]  
print ldaOut[1]  
corpus\_lda = lda[corpus\_tfidf]  
for doc in corpus\_lda:  
    print doc

得到的结果每次都变，给一次的输出：

0.077\*吃 + 0.075\*北京 + 0.075\*雾 + 0.074\*今天 + 0.073\*不 + 0.072\*霾 + 0.070\*喜欢 + 0.068\*我 + 0.062\*的 + 0.061\*土豆  
0.091\*吃 + 0.073\*搭 + 0.073\*土豆 + 0.073\*个 + 0.073\*是 + 0.072\*百 + 0.071\*东西 + 0.066\*我 + 0.065\*喜欢 + 0.059\*霾  
[(0, 0.31271095988105352), (1, 0.68728904011894654)]  
[(0, 0.19957991735916861), (1, 0.80042008264083142)]  
[(0, 0.80940337254233863), (1, 0.19059662745766134)]

第一二句和主题二相似，第三句和主题一相似

结论和LSI不一样，我估计这和样本数目太少，区别度不高有关，毕竟让我来区分把第一句和哪一句分在一个主题，我也不确定

输入一句话，查询属于LSI得到的哪个主题类型，先建立索引：

index = similarities.MatrixSimilarity(lsi[corpus])  
query = "雾霾"  
query\_bow = dic.doc2bow(list(jieba.cut(query)))  
print query\_bow  
query\_lsi = lsi[query\_bow]  
print query\_lsi

输出:

[(13, 1), (14, 1)]  
[(0, 0.50670602027401368), (1, -0.3678056037187441)]

与第一个主题相似

比较和第几句话相似，用LSI得到的索引接着做，并排序输出

sims = index[query\_lsi]  
print list(enumerate(sims))  
sort\_sims = sorted(enumerate(sims), key=lambda item: -item[1])  
print sort\_sims

输出：

[(0, 0.90161765), (1, -0.10271341), (2, 0.99058259)]  
[(2, 0.99058259), (0, 0.90161765), (1, -0.10271341)]

可见和第二句话相似度很高，因为只有第二句话出现了雾霾两个词，可是惊讶的是和第一句话的相似度也很高，这得益于LSI模型的算法：**在A和C共现，B和C共现的同时，可以找到A和B的相似度**