我爱自然语言处理

I Love Natural Language Processing

标签归档：**GENSIM**

维基百科语料中的词语相似度探索

之前写过《中英文维基百科语料上的Word2Vec实验》，近期有不少同学在这篇文章下留言提问，加上最近一些工作也与Word2Vec相关，于是又做了一些功课，包括重新过了一遍Word2Vec的相关资料，试了一下gensim的相关更新接口，google了一下"wikipedia word2vec" or "维基百科 word2vec" 相关的英中文资料，发现多数还是走得这篇文章的老路，既通过gensim提供的维基百科预处理脚本"gensim.corpora.WikiCorpus"提取维基语料，每篇文章一行文本存放，然后基于gensim的Word2Vec模块训练词向量模型。这里再提供另一个方法来处理维基百科的语料，训练词向量模型，计算词语相似度（Word Similarity)。关于Word2Vec, 如果英文不错，推荐从这篇文章入手读相关的资料: Getting started with Word2Vec 。

这次我们仅以英文维基百科语料为例，首先依然是下载维基百科的最新XML打包压缩数据，在这个英文最新更新的数据列表下：https://dumps.wikimedia.org/enwiki/latest/ ，找到 "enwiki-latest-pages-articles.xml.bz2" 下载，这份英文维基百科全量压缩数据的打包时间大概是2017年4月4号，大约13G，我通过家里的电脑wget下载大概花了3个小时，电信100M宽带，速度还不错。

接下来就是处理这份压缩的XML英文维基百科语料了，这次我们使用WikiExtractor:

> *WikiExtractor.py is a Python script that extracts and cleans text from a Wikipedia database dump.*
> *The tool is written in Python and requires Python 2.7 or Python 3.3+ but no additional library.*

WikiExtractor是一个Python 脚本，专门用于提取和清洗Wikipedia的dump数据，支持Python 2.7 或者Python 3.3+，无额外依赖，安装和使用都非常方便：

安装：

git clone https://github.com/attardi/wikiextractor.git

cd wikiextractor/

sudo python setup.py install

使用：

WikiExtractor.py -o enwiki enwiki-latest-pages-articles.xml.bz2

```
.....
INFO: 53665431  Pampapaul
INFO: 53665433  Charles Frederick Zimpel
INFO: Finished 11-process extraction of 5375019 articles in 8363.5s (642.7 art/s)
```

这个过程总计花了2个多小时，提取了大概537万多篇文章。关于我的机器配置，可参考：《深度学习主机攒机小记》

提取后的文件按一定顺序切分存储在多个子目录下：

```
textminer@textminer:/opt/wiki/data$ du -sh *
12G     enwiki
13G     enwiki-latest-pages-articles.xml.bz2
textminer@textminer:/opt/wiki/data$ cd enwiki/
textminer@textminer:/opt/wiki/data/enwiki$ ls
AA  AH  AO  AV  BC  BJ  BQ  BX  CE  CL  CS  CZ  DG  DN  DU  EB  EI
AB  AI  AP  AW  BD  BK  BR  BY  CF  CM  CT  DA  DH  DO  DV  EC  EJ
AC  AJ  AQ  AX  BE  BL  BS  BZ  CG  CN  CU  DB  DI  DP  DW  ED  EK
AD  AK  AR  AY  BF  BM  BT  CA  CH  CO  CV  DC  DJ  DQ  DX  EE  EL
AE  AL  AS  AZ  BG  BN  BU  CB  CI  CP  CW  DD  DK  DR  DY  EF  EM
AF  AM  AT  BA  BH  BO  BV  CC  CJ  CQ  CX  DE  DL  DS  DZ  EG
AG  AN  AU  BB  BI  BP  BW  CD  CK  CR  CY  DF  DM  DT  EA  EH
textminer@textminer:/opt/wiki/data/enwiki$
```

每个子目录下的又存放若干个以wiki_num命名的文件，每个大小在1M左右，这个大小可以通过参数 -b 控制:

-b n[KMG], --bytes n[KMG] maximum bytes per output file (default 1M)

```
textminer@textminer:/opt/wiki/data/enwiki$ cd AA/
textminer@textminer:/opt/wiki/data/enwiki/AA$ ls
wiki_00  wiki_13  wiki_26  wiki_39  wiki_52  wiki_65  wiki_78  wiki_91
```

```
wiki_01  wiki_14  wiki_27  wiki_40  wiki_53  wiki_66  wiki_79  wiki_92
wiki_02  wiki_15  wiki_28  wiki_41  wiki_54  wiki_67  wiki_80  wiki_93
wiki_03  wiki_16  wiki_29  wiki_42  wiki_55  wiki_68  wiki_81  wiki_94
wiki_04  wiki_17  wiki_30  wiki_43  wiki_56  wiki_69  wiki_82  wiki_95
wiki_05  wiki_18  wiki_31  wiki_44  wiki_57  wiki_70  wiki_83  wiki_96
wiki_06  wiki_19  wiki_32  wiki_45  wiki_58  wiki_71  wiki_84  wiki_97
wiki_07  wiki_20  wiki_33  wiki_46  wiki_59  wiki_72  wiki_85  wiki_98
wiki_08  wiki_21  wiki_34  wiki_47  wiki_60  wiki_73  wiki_86  wiki_99
wiki_09  wiki_22  wiki_35  wiki_48  wiki_61  wiki_74  wiki_87
wiki_10  wiki_23  wiki_36  wiki_49  wiki_62  wiki_75  wiki_88
wiki_11  wiki_24  wiki_37  wiki_50  wiki_63  wiki_76  wiki_89
wiki_12  wiki_25  wiki_38  wiki_51  wiki_64  wiki_77  wiki_90
textminer@textminer:/opt/wiki/data/enwiki/AA$
```

我们看一下wiki_00里的具体内容：

Anarchism

Anarchism is a political philosophy that advocates self-governed societies based on voluntary institutions. These are often described as stateless societies, although several authors have defined them more specifically as institutions based on non-hierarchical free associations. Anarchism holds the state to be undesirable, unnecessary, and harmful.
...
Criticisms of anarchism include moral criticisms and pragmatic criticisms. Anarchism is often evaluated as unfeasible or utopian by its critics.

Autism

Autism is a neurodevelopmental disorder characterized by impaired social interaction, verbal and non-verbal communication, and restricted and repetitive behavior. Parents usually notice signs in the first two years of their child's life. These signs often develop gradually, though some children with autism reach their developmental milestones at a normal pace and then regress. The diagnostic criteria require that symptoms become apparent in early childhood, typically before age three.
...

...

每个wiki_num文件里又存放若干个doc，每个doc都有相关的tag标记，包括id, url, title等，很好区分。

这里我们按照Gensim作者提供的word2vec tutorial里"memory-friendly iterator"方式来处理英文维基百科的数据。代码如下，也同步放到了github里：train_word2vec_with_gensim.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Author: Pan Yang (panyangnlp@gmail.com)
# Copyright 2017 @ Yu Zhen

import gensim
import logging
import multiprocessing
import os
import re
import sys

from pattern.en import tokenize
from time import time

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
                    level=logging.INFO)


def cleanhtml(raw_html):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', raw_html)
    return cleantext


class MySentences(object):
    def __init__(self, dirname):
        self.dirname = dirname

    def __iter__(self):
        for root, dirs, files in os.walk(self.dirname):
            for filename in files:
                file_path = root + '/' + filename
                for line in open(file_path):
                    sline = line.strip()
                    if sline == "":
                        continue
                    rline = cleanhtml(sline)
                    tokenized_line = ' '.join(tokenize(rline))
                    is_alpha_word_line = [word for word in
                                          tokenized_line.lower().split()
                                          if word.isalpha()]
                    yield is_alpha_word_line


if __name__ == '__main__':
    if len(sys.argv) != 2:
        print "Please use python train_with_gensim.py data_path"
        exit()
    data_path = sys.argv[1]
```

```
begin = time()

sentences = MySentences(data_path)
model = gensim.models.Word2Vec(sentences,
                        size=200,
                        window=10,
                        min_count=10,
                        workers=multiprocessing.cpu_count())
model.save("data/model/word2vec_gensim")
model.wv.save_word2vec_format("data/model/word2vec_org",
                        "data/model/vocabulary",
                        binary=False)

end = time()
print "Total procesing time: %d seconds" % (end - begin)
```

注意其中的word tokenize使用了pattern里的英文tokenize模块，当然，也可以使用nltk里的word_tokenize模块，做一点修改即可，不过nltk对于句尾的一些词的work tokenize处理的不太好。另外我们设定词向量维度为200， 窗口长度为10， 最小出现次数为10，通过 is_alpha() 函数过滤掉标点和非英文词。现在可以用这个脚本来训练英文维基百科的Word2Vec模型了：

python train_word2vec_with_gensim.py enwiki

```
2017-04-22 14:31:04,703 : INFO : collecting all words and their counts
2017-04-22 14:31:04,704 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping 0 word ty
2017-04-22 14:31:06,442 : INFO : PROGRESS: at sentence #10000, processed 480546 words, keepir
2017-04-22 14:31:08,104 : INFO : PROGRESS: at sentence #20000, processed 983240 words, keepir
2017-04-22 14:31:09,685 : INFO : PROGRESS: at sentence #30000, processed 1455218 words, keep
2017-04-22 14:31:11,349 : INFO : PROGRESS: at sentence #40000, processed 1957479 words, keep
......
2017-04-23 02:50:59,844 : INFO : worker thread finished; awaiting finish of 2 more threads
2017-04-23 02:50:59,855 : INFO : saving Word2Vec object under data/model/word2vec_gensim, sep
2017-04-23 02:50:59,855 : INFO : not storing attribute syn0norm
2017-04-23 02:50:59,855 : INFO : storing np array 'syn0' to data/model/word2vec_gensim.wv.syn0.
2017-04-23 02:51:00,241 : INFO : storing np array 'syn1neg' to data/model/word2vec_gensim.syn1
2017-04-23 02:51:00,574 : INFO : not storing attribute cum_table
2017-04-23 02:51:13,886 : INFO : saved data/model/word2vec_gensim
2017-04-23 02:51:13,886 : INFO : storing vocabulary in data/model/vocabulary
2017-04-23 02:51:17,480 : INFO : storing 868777x200 projection weights into data/model/word2ve
Total procesing time: 44476 seconds
```

这个训练过程中大概花了12多小时，训练后的文件存放在data/model下：

```
-rw-rw-r-- 1 textminer textminer  10M 4月   23 02:51 vocabulary
-rw-rw-r-- 1 textminer textminer  56M 4月   23 02:51 word2vec_gensim
-rw-rw-r-- 1 textminer textminer 663M 4月   23 02:51 word2vec_gensim.syn1neg.npy
-rw-rw-r-- 1 textminer textminer 663M 4月   23 02:51 word2vec_gensim.wv.syn0.npy
-rw-rw-r-- 1 textminer textminer 1.6G 4月   23 02:52 word2vec_org
textminer@textminer:/opt/wiki/data/data/model$
```

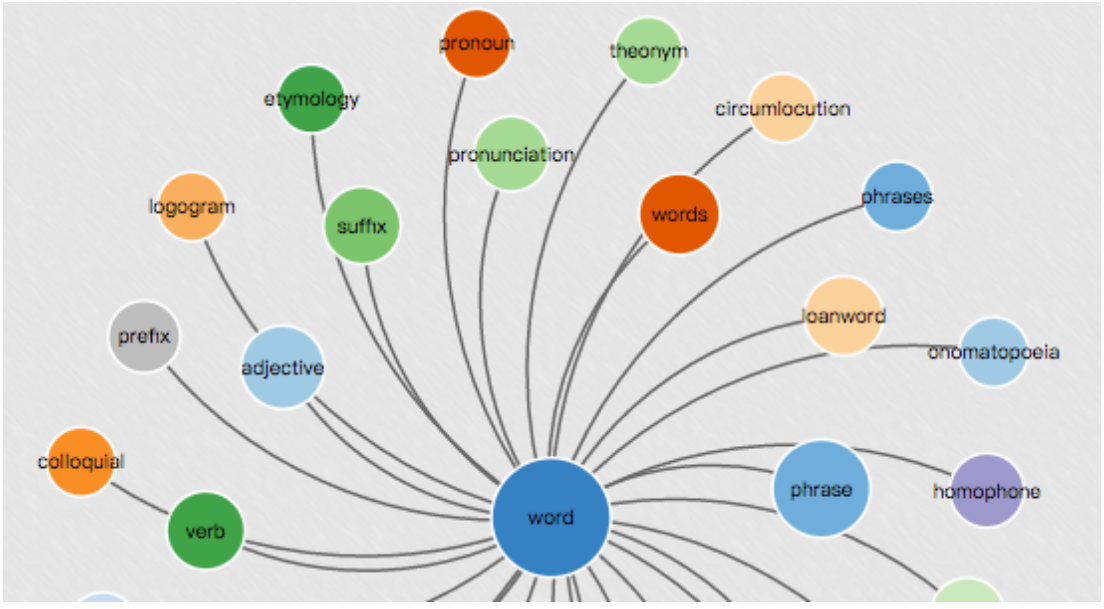
我们来测试一下这个英文维基百科的Word2Vec模型：

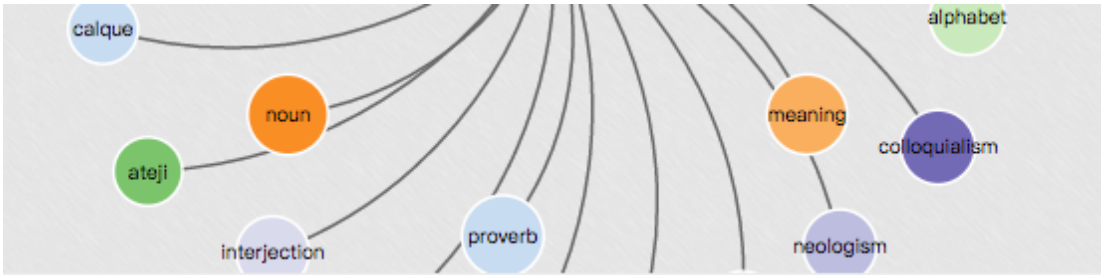
```
textminer@textminer:/opt/wiki/data$ ipython
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from gensim.models import Word2Vec

In [2]: en_wiki_word2vec_model = Word2Vec.load('data/model/word2vec_gensim')
```

首先来测试几个单词的相似单词(Word Similariy)：

word:

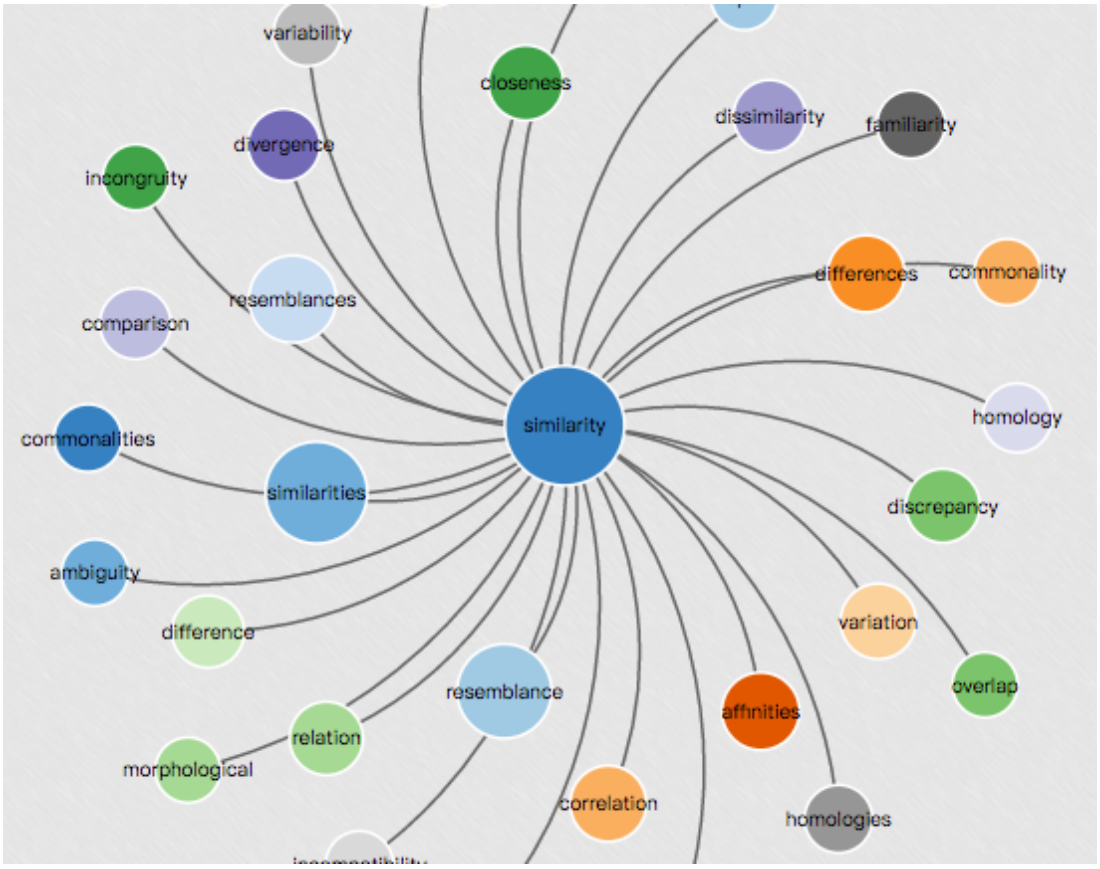
```
In [3]: en_wiki_word2vec_model.most_similar('word')
Out[3]:
[('phrase', 0.8129693269729614),
 ('meaning', 0.7311851978302002),
 ('words', 0.7010501623153687),
 ('adjective', 0.6805518865585327),
 ('noun', 0.6461974382400513),
 ('suffix', 0.6440576314926147),
 ('verb', 0.6319557428359985),
 ('loanword', 0.6262609958648682),
 ('proverb', 0.6240501403808594),
 ('pronunciation', 0.6105246543884277)]
```





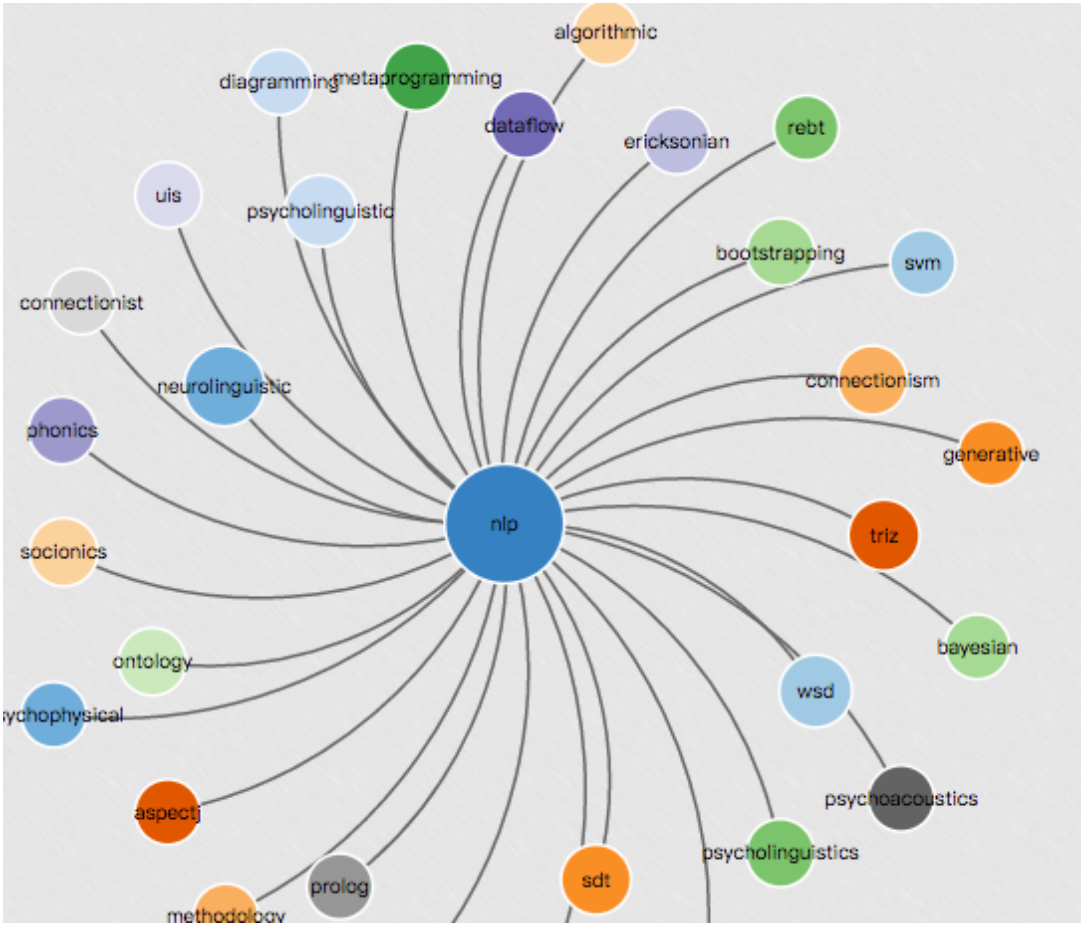
similarity:

```
In [4]: en_wiki_word2vec_model.most_similar('similarity')
Out[4]:
[('similarities', 0.8517599701881409),
 ('resemblance', 0.786037266254425),
 ('resemblances', 0.7496883869171143),
 ('affinities', 0.6571112275123596),
 ('differences', 0.6465682983398438),
 ('dissimilarities', 0.6212711930274963),
 ('correlation', 0.6071442365646362),
 ('dissimilarity', 0.6062943935394287),
 ('variation', 0.5970577001571655),
 ('difference', 0.5928016901016235)]
```



nlp:

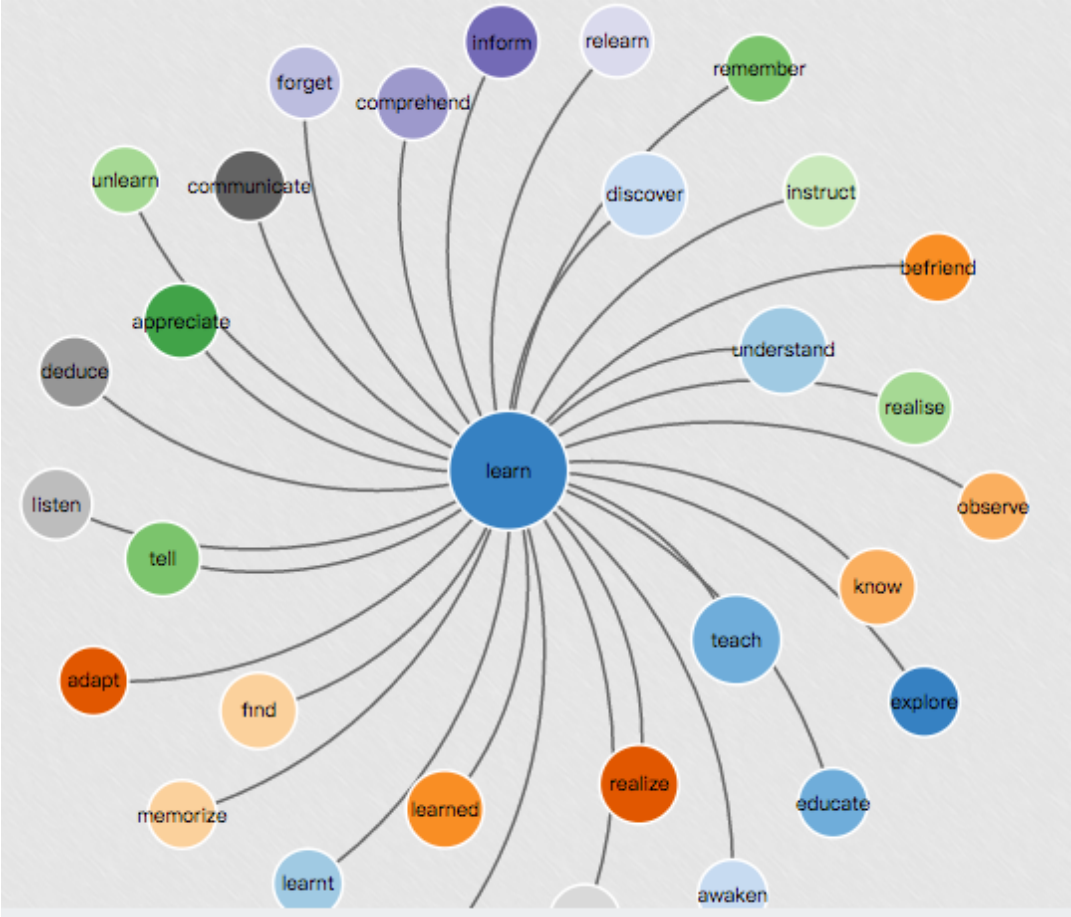
```
In [5]: en_wiki_word2vec_model.most_similar('nlp')
Out[5]:
[('neurolinguistic', 0.6698148250579834),
 ('psycholinguistic', 0.6388964056968689),
 ('connectionism', 0.6027182936668396),
 ('semantics', 0.5866401195526123),
 ('connectionist', 0.5865628719329834),
 ('bandler', 0.5837364196777344),
 ('phonics', 0.5733655691146851),
 ('psycholinguistics', 0.5613113641738892),
 ('bootstrapping', 0.559638261795044),
 ('psychometrics', 0.5555593967437744)]
```



learn:

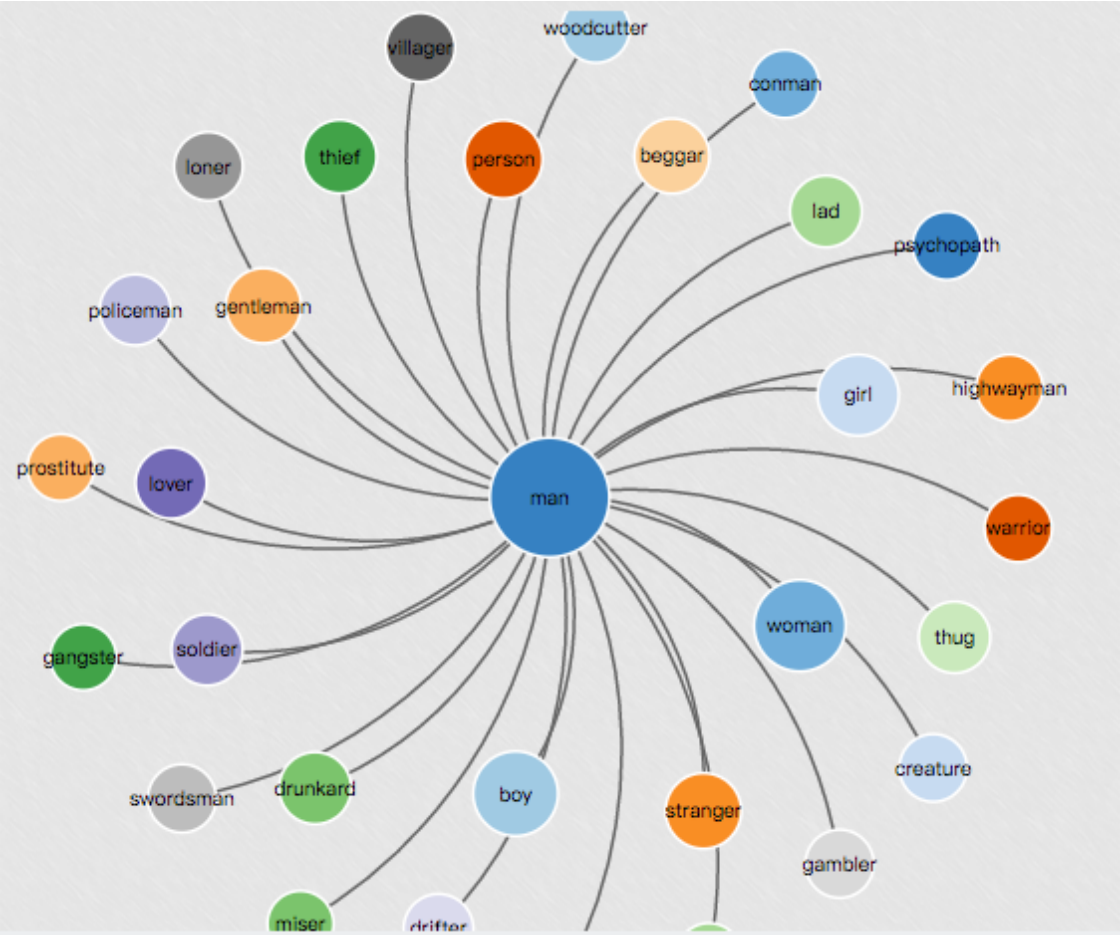
```
In [6]: en_wiki_word2vec_model.most_similar('learn')
Out[6]:
[('teach', 0.7533557415008545),
 ('understand', 0.71148681640625),
 ('discover', 0.6749690771102905),
```

```
('learned', 0.6599283218383789),
('realize', 0.6390970349311829),
('find', 0.6308424472808838),
('know', 0.6171890497207642),
('tell', 0.6146825551986694),
('inform', 0.6008728742599487),
('instruct', 0.5998791456222534)]
```



man:

```
In [7]: en_wiki_word2vec_model.most_similar('man')
Out[7]:
[('woman', 0.7243080735206604),
 ('boy', 0.7029494047164917),
 ('girl', 0.6441491842269897),
 ('stranger', 0.63275545835495),
 ('drunkard', 0.6136815547943115),
 ('gentleman', 0.6122575998306274),
 ('lover', 0.6108279228210449),
 ('thief', 0.609005331993103),
 ('beggar', 0.6083744764328003),
 ('person', 0.597919225692749)]
```



再来看看其他几个相关接口：

```
In [8]: en_wiki_word2vec_model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
Out[8]: [('queen', 0.7752252817153931)]

In [9]: en_wiki_word2vec_model.similarity('woman', 'man')
Out[9]: 0.72430799548282099

In [10]: en_wiki_word2vec_model.doesnt_match("breakfast cereal dinner lunch".split())
Out[10]: 'cereal'
```

我把这篇文章的相关代码还有另一篇“中英文维基百科语料上的Word2Vec实验”的相关代码整理了一下，在github上建立了一个 Wikipedia_Word2vec 的项目，感兴趣的同学可以参考。

注：原创文章，转载请注明出处及保留链接“我爱自然语言处理”：http://www.52nlp.cn

本文链接地址：维基百科语料中的词语相似度探索 http://www.52nlp.cn/?p=9454

本条目发布于2017年04月24号 [http://www.52nlp.cn/%e7%bb%b4%e5%9f%ba%e7%99%be%e7%a7%91

%e8%af%ad%e6%96%99%e4%b8%ad%e7%9a%84%e8%af%8d%e8%af%ad%e7%9b%b8%e4%bc%bc%e5%ba%a6%e6%8e%a2%e7%b4%a2] 。属于深度学习、自然语言处理、语义相似度分类，被贴了 gensim、gensim word2vec、nltk、Pattern、pyhton word2vec、python、python gensim、WikiExtractor、Wikipedia、Wikipedia_Word2vec、Word Similarity、word tokenize、word2vec、Word2Vec Tutorial、维基百科、维基语料、词语相似度、语料 标签。作者是52nlp。

中英文维基百科语料上的Word2Vec实验

最近试了一下Word2Vec, GloVe 以及对应的python版本 gensim word2vec 和 python-glove，就有心在一个更大规模的语料上测试一下，自然而然维基百科的语料进入了视线。维基百科官方提供了一个很好的维基百科数据源：https://dumps.wikimedia.org，可以方便的下载多种语言多种格式的维基百科数据。此前通过gensim的玩过英文的维基百科语料并训练LSI，LDA模型来计算两个文档的相似度，所以想看看gensim有没有提供一种简便的方式来处理维基百科数据，训练word2vec模型，用于计算词语之间的语义相似度。感谢Google，在gensim的google group下，找到了一个很长的讨论帖：training word2vec on full Wikipedia ，这个帖子基本上把如何使用gensim在维基百科语料上训练word2vec模型的问题说清楚了，甚至参与讨论的gensim的作者Radim Řehůřek博士还在新的gensim版本里加了一点修正，而对于我来说，所做的工作就是做一下验证而已。虽然github上有一个wiki2vec的项目也是做得这个事，不过我更喜欢用python gensim的方式解决问题。

关于word2vec，这方面无论中英文的参考资料相当的多，英文方面既可以看官方推荐的论文，也可以看gensim作者Radim Řehůřek博士写得一些文章。而中文方面，推荐 @licstar的《Deep Learning in NLP（一）词向量和语言模型》，有道技术沙龙的《Deep Learning实战之word2vec》，@飞林沙 的《word2vec的学习思路》, falao_beiliu 的《深度学习word2vec笔记之基础篇》和《深度学习word2vec笔记之算法篇》等。
继续阅读 →

本条目发布于2015年03月12号 [http://www.52nlp.cn/%e4%b8%ad%e8%8b%b1%e6%96%87%e7%bb%b4%e5%9f%ba%e7%99%be%e7%a7%91%e8%af%ad%e6%96%99%e4%b8%8a%e7%9a%84word2vec%e5%ae%9e%e9%aa%8c] 。属于自然语言处理、语义相似度、语言模型分类，被贴了 gensim、gensim word2vec、glove、Mecab、python gensim、python glove、python word2vec、word2vec、word2vec实

验、word2vec应用、word2vec模型、word2vec相似度、word2vec相似度计算、word2vec词语相似度、中文分词、中文繁简转换、中文维基百科语料、中文编码转换、文档相似度、深度学习、相似度、维基百科、维基百科语料、英文维基百科语料、词语相似度、语义相似度、语言模型 标签。作者是52nlp。

Python 网页爬虫 & 文本处理 & 科学计算 & 机器学习 & 数据挖掘兵器谱

曾经因为NLTK的缘故开始学习Python，之后渐渐成为我工作中的第一辅助脚本语言，虽然开发语言是C/C++，但平时的很多文本数据处理任务都交给了Python。离开腾讯创业后，第一个作品课程图谱也是选择了Python系的Flask框架，渐渐的将自己的绝大部分工作交给了Python。这些年来，接触和使用了很多Python工具包，特别是在文本处理，科学计算，机器学习和数据挖掘领域，有很多很多优秀的Python工具包可供使用，所以作为Pythoner，也是相当幸福的。其实如果仔细留意微博，你会发现很多这方面的分享，自己也Google了一下，发现也有同学总结了“Python机器学习库”，不过总感觉缺少点什么。最近流行一个词，全栈工程师（full stack engineer），作为一个苦逼的创业者，天然的要把自己打造成一个full stack engineer，而这个过程中，这些Python工具包给自己提供了足够的火力，所以想起了这个系列。当然，这也仅仅是抛砖引玉，希望大家能提供更多的线索，来汇总整理一套Python网页爬虫，文本处理，科学计算，机器学习和数据挖掘的兵器谱。

一、Python网页爬虫工具集

一个真实的项目，一定是从获取数据开始的。无论文本处理，机器学习和数据挖掘，都需要数据，除了通过一些渠道购买或者下载的专业数据外，常常需要大家自己动手爬数据，这个时候，爬虫就显得格外重要了，幸好，Python提供了一批很不错的网页爬虫工具框架，既能爬取数据，也能获取和清洗数据，我们也就从这里开始了：

1. Scrapy

> *Scrapy, a fast high-level screen scraping and web crawling framework for Python.*

鼎鼎大名的Scrapy，相信不少同学都有耳闻，课程图谱中的很多课程都是依靠Scrapy抓去的，这方面的介绍文章有很多，推荐大牛pluskid早年的一篇文章：《Scrapy 轻松定制网络爬虫》，历久弥新。

官方主页：http://scrapy.org/
Github代码页: https://github.com/scrapy/scrapy

2. Beautiful Soup

> *You didn't write that awful page. You're just trying to get some data out of it. Beautiful Soup is here to help. Since 2004, it's been saving programmers hours or days of work on quick-turnaround screen scraping projects.*

读书的时候通过《集体智慧编程》这本书知道Beautiful Soup的，后来也偶尔会用用，非常棒的一套工具。客观的说，Beautifu Soup不完全是一套爬虫工具，需要配合urllib使用，而是一套HTML/XML数据分析，清洗和获取工具。

官方主页：http://www.crummy.com/software/BeautifulSoup/

3. Python-Goose

> *Html Content / Article Extractor, web scrapping lib in Python*

Goose最早是用Java写得，后来用Scala重写，是一个Scala项目。Python-Goose用Python重写，依赖了Beautiful Soup。前段时间用过，感觉很不错，给定一个文章的URL, 获取文章的标题和内容很方便。

Github主页：https://github.com/grangier/python-goose

二、Python文本处理工具集

从网页上获取文本数据之后，依据任务的不同，就需要进行基本的文本处理了，譬如对于英文来说，需要基本的tokenize，对于中文，则需要常见的中文分词，进一步的话，无论英文中文，还可以词性标注，句法分析，关键词提取，文本分类，情感分析等等。这个方面，特别是面向英文领域，有很多优秀的工具包，我们一一道来。
继续阅读 →

本条目发布于2014年07月24号 [http://www.52nlp.cn/python-%e7%bd%91%e9%a1%b5%e7%88%ac%e8%99%ab-%e6%96%87%e6%9c%ac%e5%a4%84%e7%90%86-%e7%a7%91%e5%ad%a6%e8%ae%a1%e7%ae%97-%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0-%e6%95%b0%e6%8d%ae%e6%8c%96%e6%8e%98] 。属于数据挖掘、机器学习、科学计算、自然语言处理分类，被贴了 gensim、Jieba、Jieba中文分词、LangId、MBSP、nltk、Pattern、python、Python开源工具包、Python数据挖掘、Python文本处理、Python文本挖掘、Python机器学习、Python爬虫、Python科学计算、Python网页爬虫、Scrapy、TextBlob、中文分词、数据挖掘、数据挖掘开源工具包、文本处理、文本处理开源工具包、文本挖掘、机器学习、机器学习开源工具包、爬虫、科学计算、网页爬虫、语言检测 标签。作者是52nlp。

如何计算两个文档的相似度（三）

上一节我们用了一个简单的例子过了一遍gensim的用法，这一节我们将用课程图谱的实际数据来做一些验证和改进，同时会用到NLTK来对课程的英文数据做预处理。

三、课程图谱相关实验

1、数据准备

为了方便大家一起来做验证，这里准备了一份Coursera的课程数据，可以在这里下载：coursera_corpus，（百度网盘链接: http://t.cn/RhjgPkv，密码: oppc）总共379个课程，每行包括3部分内容：课程名\t课程简介\t课程详情, 已经清除了其中的html tag, 下面所示的例子仅仅是其中的课程名：

Writing II: Rhetorical Composing
Genetics and Society: A Course for Educators
General Game Playing
Genes and the Human Condition (From Behavior to Biotechnology)
A Brief History of Humankind
New Models of Business in Society
Analyse Numérique pour Ingénieurs
Evolution: A Course for Educators

Coding the Matrix: Linear Algebra through Computer Science Applications
The Dynamic Earth: A Course for Educators
...

好了，首先让我们打开Python, 加载这份数据：

```
>>> courses = [line.strip() for line in file('coursera_corpus')]
>>> courses_name = [course.split('\t')[0] for course in courses]
>>> print courses_name[0:10]
['Writing II: Rhetorical Composing', 'Genetics and Society: A Course for Educators', 'General Game Playing', 'Genes and the Human Condition (From Behavior to Biotechnology)', 'A Brief History of Humankind', 'New Models of Business in Society', 'Analyse Num\xc3\xa9rique pour Ing\xc3\xa9nieurs', 'Evolution: A Course for Educators', 'Coding the Matrix: Linear Algebra through Computer Science Applications', 'The Dynamic Earth: A Course for Educators']
```

2、引入NLTK
NTLK是著名的Python自然语言处理工具包，但是主要针对的是英文处理，不过课程图谱目前处理的课程数据主要是英文，因此也足够了。NLTK配套有文档，有语料库，有书籍，甚至国内有同学无私的翻译了这本书: 用Python进行自然语言处理，有时候不得不感慨：做英文自然语言处理的同学真幸福。

首先仍然是安装NLTK，在NLTK的主页详细介绍了如何在Mac, Linux和Windows下安装NLTK：http://nltk.org/install.html ，最主要的还是要先装好依赖NumPy和PyYAML，其他没什么问题。安装NLTK完毕，可以import nltk测试一下，如果没有问题，还有一件非常重要的工作要做，下载NLTK官方提供的相关语料：

```
>>> import nltk
>>> nltk.download()
```

这个时候会弹出一个图形界面，会显示两份数据供你下载，分别是all-corpora和book，最好都选定下载了，这个过程需要一段时间，语料下载完毕后，NLTK在你的电脑上才真正达到可用的状态，可以测试一下布朗语料库：

```
>>> from nltk.corpus import brown
>>> brown.readme()
'BROWN CORPUS\n\nA Standard Corpus of Present-Day Edited American\nEnglish, for use with Digital Computers.\n\nby W. N. Francis and H. Kucera (1964)\nDepartment of Linguistics, Brown University\nProvidence, Rhode Island, USA\n\nRevised 1971, Revised and Amplified 1979\n\nhttp://www.hit.uib.no/icame/brown/bcm.html\n\nDistributed with the permission of the copyright
```

```
holder,\nredistribution permitted.\n'
>>> brown.words()[0:10]
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of']
>>> brown.tagged_words()[0:10]
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-TL'), ('Jury', 'NN-TL'), ('said', 'VBD'),
('Friday', 'NR'), ('an', 'AT'), ('investigation', 'NN'), ('of', 'IN')]
>>> len(brown.words())
1161192
```

现在我们就来处理刚才的课程数据，如果按此前的方法仅仅对文档的单词小写化的话，我们将得到如下的结果：

```
>>> texts_lower = [[word for word in document.lower().split()] for document in courses]
>>> print texts_lower[0]
['writing', 'ii:', 'rhetorical', 'composing', 'rhetorical', 'composing', 'engages', 'you', 'in', 'a', 'series', 'of',
'interactive', 'reading,', 'research,', 'and', 'composing', 'activities', 'along', 'with', 'assignments', 'designed',
'to', 'help', 'you', 'become', 'more', 'effective', 'consumers', 'and', 'producers', 'of', 'alphabetic,', 'visual', 'and',
'multimodal', 'texts.', 'join', 'us', 'to', 'become', 'more', 'effective', 'writers...', 'and', 'better', 'citizens.',
'rhetorical', 'composing', 'is', 'a', 'course', 'where', 'writers', 'exchange', 'words,', 'ideas,', 'talents,', 'and',
'support.', 'you', 'will', 'be', 'introduced', 'to', 'a', ...
```

注意其中很多标点符号和单词是没有分离的，所以我们引入nltk的word_tokenize函数，并处理相应的数据：

```
>>> from nltk.tokenize import word_tokenize
>>> texts_tokenized = [[word.lower() for word in word_tokenize(document.decode('utf-8'))] for document in
courses]
>>> print texts_tokenized[0]
['writing', 'ii', ':', 'rhetorical', 'composing', 'rhetorical', 'composing', 'engages', 'you', 'in', 'a', 'series', 'of',
'interactive', 'reading', ',', 'research', ',', 'and', 'composing', 'activities', 'along', 'with', 'assignments',
'designed', 'to', 'help', 'you', 'become', 'more', 'effective', 'consumers', 'and', 'producers', 'of', 'alphabetic', ',',
'visual', 'and', 'multimodal', 'texts.', 'join', 'us', 'to', 'become', 'more', 'effective', 'writers', '...', 'and', 'better',
'citizens.', 'rhetorical', 'composing', 'is', 'a', 'course', 'where', 'writers', 'exchange', 'words', ',', 'ideas', ',',
'talents', ',', 'and', 'support.', 'you', 'will', 'be', 'introduced', 'to', 'a', ...
```

对课程的英文数据进行tokenize之后，我们需要去停用词，幸好NLTK提供了一份英文停用词数据：

```
>>> from nltk.corpus import stopwords
>>> english_stopwords = stopwords.words('english')
```

```
>>> print english_stopwords
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while',
'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below',
'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when',
'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now']
>>> len(english_stopwords)
127
```

总计127个停用词，我们首先过滤课程语料中的停用词：

```
>>> texts_filtered_stopwords = [[word for word in document if not word in english_stopwords] for document
in texts_tokenized]
>>> print texts_filtered_stopwords[0]
['writing', 'ii', ':', 'rhetorical', 'composing', 'rhetorical', 'composing', 'engages', 'series', 'interactive', 'reading',
',', 'research', ',', 'composing', 'activities', 'along', 'assignments', 'designed', 'help', 'become', 'effective',
'consumers', 'producers', 'alphabetic', ',', 'visual', 'multimodal', 'texts.', 'join', 'us', 'become', 'effective',
'writers', '...', 'better', 'citizens.', 'rhetorical', 'composing', 'course', 'writers', 'exchange', 'words', ',', 'ideas', ',',
'talents', ',', 'support.', 'introduced', 'variety', 'rhetorical', 'concepts\xe2\x80\x94that', ',', 'ideas', 'techniques',
'inform', 'persuade', 'audiences\xe2\x80\x94that', 'help', 'become', 'effective', 'consumer', 'producer',
'written', ',', 'visual', ',', 'multimodal', 'texts.', 'class', 'includes', 'short', 'videos', ',', 'demonstrations', ',',
'activities.', 'envision', 'rhetorical', 'composing', 'learning', 'community', 'includes', 'enrolled', 'course',
'instructors.', 'bring', 'expertise', 'writing', ',', 'rhetoric', 'course', 'design', ',', 'designed', 'assignments',
'course', 'infrastructure', 'help', 'share', 'experiences', 'writers', ',', 'students', ',', 'professionals', 'us.',
'collaborations', 'facilitated', 'wex', ',', 'writers', 'exchange', ',', 'place', 'exchange', 'work', 'feedback']
```

停用词被过滤了，不过发现标点符号还在，这个好办，我们首先定义一个标点符号list:

```
>>> english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '&', '!', '*', '@', '#', '$', '%']
```

然后过滤这些标点符号：

```
>>> texts_filtered = [[word for word in document if not word in english_punctuations] for document in
texts_filtered_stopwords]
>>> print texts_filtered[0]
['writing', 'ii', 'rhetorical', 'composing', 'rhetorical', 'composing', 'engages', 'series', 'interactive', 'reading',
'research', 'composing', 'activities', 'along', 'assignments', 'designed', 'help', 'become', 'effective',
'consumers', 'producers', 'alphabetic', 'visual', 'multimodal', 'texts.', 'join', 'us', 'become', 'effective', 'writers',
```

'...', 'better', 'citizens.', 'rhetorical', 'composing', 'course', 'writers', 'exchange', 'words', 'ideas', 'talents', 'support.', 'introduced', 'variety', 'rhetorical', 'concepts\xe2\x80\x94that', 'ideas', 'techniques', 'inform', 'persuade', 'audiences\xe2\x80\x94that', 'help', 'become', 'effective', 'consumer', 'producer', 'written', 'visual', 'multimodal', 'texts.', 'class', 'includes', 'short', 'videos', 'demonstrations', 'activities.', 'envision', 'rhetorical', 'composing', 'learning', 'community', 'includes', 'enrolled', 'course', 'instructors.', 'bring', 'expertise', 'writing', 'rhetoric', 'course', 'design', 'designed', 'assignments', 'course', 'infrastructure', 'help', 'share', 'experiences', 'writers', 'students', 'professionals', 'us.', 'collaborations', 'facilitated', 'wex', 'writers', 'exchange', 'place', 'exchange', 'work', 'feedback']

更进一步，我们对这些英文单词词干化（Stemming)，NLTK提供了好几个相关工具接口可供选择，具体参考这个页面: http://nltk.org/api/nltk.stem.html，可选的工具包括Lancaster Stemmer, Porter Stemmer等知名的英文Stemmer。这里我们使用LancasterStemmer:

```
>>> from nltk.stem.lancaster import LancasterStemmer
>>> st = LancasterStemmer()
>>> st.stem('stemmed')
'stem'
>>> st.stem('stemming')
'stem'
>>> st.stem('stemmer')
'stem'
>>> st.stem('running')
'run'
>>> st.stem('maximum')
'maxim'
>>> st.stem('presumably')
'presum'
```

让我们调用这个接口来处理上面的课程数据:

```
>>> texts_stemmed = [[st.stem(word) for word in docment] for docment in texts_filtered]
>>> print texts_stemmed[0]
['writ', 'ii', 'rhet', 'compos', 'rhet', 'compos', 'eng', 'sery', 'interact', 'read', 'research', 'compos', 'act', 'along', 'assign', 'design', 'help', 'becom', 'effect', 'consum', 'produc', 'alphabet', 'vis', 'multimod', 'texts.', 'join', 'us', 'becom', 'effect', 'writ', '...', 'bet', 'citizens.', 'rhet', 'compos', 'cours', 'writ', 'exchang', 'word', 'idea', 'tal', 'support.', 'introduc', 'vary', 'rhet', 'concepts\xe2\x80\x94that', 'idea', 'techn', 'inform', 'persuad', 'audiences\xe2\x80\x94that', 'help', 'becom', 'effect', 'consum', 'produc', 'writ', 'vis', 'multimod', 'texts.', 'class', 'includ', 'short', 'video', 'demonst', 'activities.', 'envid', 'rhet', 'compos', 'learn', 'commun', 'includ', 'enrol', 'cours', 'instructors.', 'bring', 'expert', 'writ', 'rhet', 'cours', 'design', 'design', 'assign', 'cours',
```

'infrastruct', 'help', 'shar', 'expery', 'writ', 'stud', 'profess', 'us.', 'collab', 'facilit', 'wex', 'writ', 'exchang', 'plac', 'exchang', 'work', 'feedback']

在我们引入gensim之前，还有一件事要做，去掉在整个语料库中出现次数为1的低频词，测试了一下，不去掉的话对效果有些影响：

```
>>> all_stems = sum(texts_stemmed, [])
>>> stems_once = set(stem for stem in set(all_stems) if all_stems.count(stem) == 1)
>>> texts = [[stem for stem in text if stem not in stems_once] for text in texts_stemmed]
```

3、引入gensim

有了上述的预处理，我们就可以引入gensim，并快速的做课程相似度的实验了。以下会快速的过一遍流程，具体的可以参考上一节的详细描述。

```
>>> from gensim import corpora, models, similarities
>>> import logging
>>> logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

>>> dictionary = corpora.Dictionary(texts)
2013-06-07 21:37:07,120 : INFO : adding document #0 to Dictionary(0 unique tokens)
2013-06-07 21:37:07,263 : INFO : built Dictionary(3341 unique tokens) from 379 documents (total 46417 corpus positions)

>>> corpus = [dictionary.doc2bow(text) for text in texts]

>>> tfidf = models.TfidfModel(corpus)
2013-06-07 21:58:30,490 : INFO : collecting document frequencies
2013-06-07 21:58:30,490 : INFO : PROGRESS: processing document #0
2013-06-07 21:58:30,504 : INFO : calculating IDF weights for 379 documents and 3341 features (29166 matrix non-zeros)

>>> corpus_tfidf = tfidf[corpus]
```

这里我们拍脑门决定训练topic数量为10的LSI模型：

```
>>> lsi = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=10)

>>> index = similarities.MatrixSimilarity(lsi[corpus])
2013-06-07 22:04:55,443 : INFO : scanning corpus to determine the number of features
```

2013-06-07 22:04:55,510 : INFO : creating matrix for 379 documents and 10 features

基于LSI模型的课程索引建立完毕，我们以Andrew Ng教授的机器学习公开课为例，这门课程在我们的coursera_corpus文件的第211行，也就是：

```
>>> print courses_name[210]
Machine Learning
```

现在我们就可以通过lsi模型将这门课程映射到10个topic主题模型空间上，然后和其他课程计算相似度：

```
>>> ml_course = texts[210]
>>> ml_bow = dicionary.doc2bow(ml_course)
>>> ml_lsi = lsi[ml_bow]
>>> print ml_lsi
[(0, 8.3270084238788673), (1, 0.91295652151975082), (2, -0.28296075112669405), (3,
0.0011599008827843801), (4, -4.1820134980024255), (5, -0.37889856481054851), (6,
2.0446999575052125), (7, 2.3297944485200031), (8, -0.32875594265388536), (9,
-0.30389668455507612)]
>>> sims = index[ml_lsi]
>>> sort_sims = sorted(enumerate(sims), key=lambda item: -item[1])
```

取按相似度排序的前10门课程：

```
>>> print sort_sims[0:10]
[(210, 1.0), (174, 0.97812241), (238, 0.96428639), (203, 0.96283489), (63, 0.9605484), (189,
0.95390636), (141, 0.94975704), (184, 0.94269753), (111, 0.93654782), (236, 0.93601125)]
```

第一门课程是它自己:

```
>>> print courses_name[210]
Machine Learning
```

第二门课是Coursera上另一位大牛Pedro Domingos机器学习公开课

```
>>> print courses_name[174]
Machine Learning
```

第三门课是Coursera的另一位创始人，同样是大牛的Daphne Koller教授的概率图模型公开课：

```
>>> print courses_name[238]
Probabilistic Graphical Models
```

第四门课是另一位超级大牛Geoffrey Hinton的神经网络公开课，有同学评价是Deep Learning的必修课。

>>> print courses_name[203]
Neural Networks for Machine Learning

感觉效果还不错，如果觉得有趣的话，也可以动手试试。

好了，这个系列就到此为止了，原计划写一下在英文维基百科全量数据上的实验，因为课程图谱目前暂时不需要，所以就到此为止，感兴趣的同学可以直接阅读gensim上的相关文档，非常详细。之后我可能更关注将NLTK应用到中文信息处理上，欢迎关注。

注：原创文章，转载请注明出处“我爱自然语言处理”：www.52nlp.cn

本文链接地址：http://www.52nlp.cn/如何计算两个文档的相似度三

本条目发布于2013年06月7号 [http://www.52nlp.cn/%e5%a6%82%e4%bd%95%e8%ae%a1%e7%ae%97%e4%b8%a4%e4%b8%aa%e6%96%87%e6%a1%a3%e7%9a%84%e7%9b%b8%e4%bc%bc%e5%ba%a6%e4%b8%89] 。属于Topic Model、推荐系统、自然语言处理分类，被贴了 Deep Learning、Deep Learning公开课、gensim、nltk、NLTK中文信息处理、NLTK应用、python、Python自然语言处理、布朗语料库、文档相似度、机器学习、机器学习公开课、概率图模型、概率图模型公开课、神经网络公开课、自然语言处理、课程图谱 标签。作者是52nlp。

如何计算两个文档的相似度（二）

上一节我们介绍了一些背景知识以及gensim，相信很多同学已经尝试过了。这一节将从gensim最基本的安装讲起，然后举一个非常简单的例子用以说明如何使用gensim，下一节再介绍其在课程图谱上的应用。

二、gensim的安装和使用

1、安装
gensim依赖NumPy和SciPy这两大Python科学计算工具包，一种简单的安装方法是pip install，但是国内因为网络的缘故常常失败。所以我是下载了gensim的源代码包安装的。gensim的这个官方安装页面很详细的

列举了兼容的Python和NumPy, SciPy的版本号以及安装步骤，感兴趣的同学可以直接参考。下面我仅仅说明在Ubuntu和Mac OS下的安装：

1）我的VPS是64位的Ubuntu 12.04，所以安装numpy和scipy比较简单"sudo apt-get install python-numpy python-scipy", 之后解压gensim的安装包，直接“sudo python setup.py install"即可；

2）我的本是macbook pro，在mac os上安装numpy和scipy的源码包废了一下周折，特别是后者，一直提示fortran相关的东西没有，google了一下，发现很多人在mac上安装scipy的时候都遇到了这个问题，最后通过homebrew安装了gfortran才搞定：“brew install gfortran”,之后仍然是“sudo python setpy.py install" numpy 和 scipy即可；

2、使用
gensim的官方tutorial非常详细，英文ok的同学可以直接参考。以下我会按自己的理解举一个例子说明如何使用gensim，这个例子不同于gensim官方的例子，可以作为一个补充。上一节提到了一个文档：Latent Semantic Indexing (LSI) A Fast Track Tutorial , 这个例子的来源就是这个文档所举的3个一句话doc。首先让我们在命令行中打开python，做一些准备工作:

```
>>> from gensim import corpora, models, similarities
>>> import logging
>>> logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

然后将上面那个文档中的例子作为文档输入，在Python中用document list表示：

```
>>> documents = ["Shipment of gold damaged in a fire",
... "Delivery of silver arrived in a silver truck",
... "Shipment of gold arrived in a truck"]
```

正常情况下，需要对英文文本做一些预处理工作，譬如去停用词，对文本进行tokenize，stemming以及过滤掉低频的词，但是为了说明问题，也是为了和这篇"LSI Fast Track Tutorial"保持一致，以下的预处理仅仅是将英文单词小写化：

```
>>> texts = [[word for word in document.lower().split()] for document in documents]
>>> print texts
[['shipment', 'of', 'gold', 'damaged', 'in', 'a', 'fire'], ['delivery', 'of', 'silver', 'arrived', 'in', 'a', 'silver', 'truck'],
['shipment', 'of', 'gold', 'arrived', 'in', 'a', 'truck']]
```

我们可以通过这些文档抽取一个“词袋（bag-of-words)"，将文档的token映射为id：

```
>>> dictionary = corpora.Dictionary(texts)
>>> print dictionary
Dictionary(11 unique tokens)
>>> print dictionary.token2id
{'a': 0, 'damaged': 1, 'gold': 3, 'fire': 2, 'of': 5, 'delivery': 8, 'arrived': 7, 'shipment': 6, 'in': 4, 'truck': 10, 'silver': 9}
```

然后就可以将用字符串表示的文档转换为用id表示的文档向量：

```
>>> corpus = [dictionary.doc2bow(text) for text in texts]
>>> print corpus
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1)], [(0, 1), (4, 1), (5, 1), (7, 1), (8, 1), (9, 2), (10, 1)], [(0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (10, 1)]]
```

例如（9，2）这个元素代表第二篇文档中id为9的单词“silver”出现了2次。

有了这些信息，我们就可以基于这些“训练文档”计算一个TF-IDF“模型”：

```
>>> tfidf = models.TfidfModel(corpus)
2013-05-27 18:58:15,831 : INFO : collecting document frequencies
2013-05-27 18:58:15,881 : INFO : PROGRESS: processing document #0
2013-05-27 18:58:15,881 : INFO : calculating IDF weights for 3 documents and 11 features (21 matrix non-zeros)
```

基于这个TF-IDF模型，我们可以将上述用词频表示文档向量表示为一个用tf-idf值表示的文档向量：

```
>>> corpus_tfidf = tfidf[corpus]
>>> for doc in corpus_tfidf:
... print doc
...
[(1, 0.6633689723434505), (2, 0.6633689723434505), (3, 0.2448297500958463), (6, 0.2448297500958463)]
[(7, 0.16073253746956623), (8, 0.4355066251613605), (9, 0.871013250322721), (10, 0.16073253746956623)]
[(3, 0.5), (6, 0.5), (7, 0.5), (10, 0.5)]
```

发现一些token貌似丢失了，我们打印一下tfidf模型中的信息：

```
>>> print tfidf.dfs
{0: 3, 1: 1, 2: 1, 3: 2, 4: 3, 5: 3, 6: 2, 7: 2, 8: 1, 9: 1, 10: 2}
>>> print tfidf.idfs
{0: 0.0, 1: 1.5849625007211563, 2: 1.5849625007211563, 3: 0.5849625007211562, 4: 0.0, 5: 0.0, 6:
0.5849625007211562, 7: 0.5849625007211562, 8: 1.5849625007211563, 9: 1.5849625007211563, 10:
0.5849625007211562}
```

我们发现由于包含id为0， 4， 5这3个单词的文档数（df)为3，而文档总数也为3，所以idf被计算为0了，看来gensim没有对分子加1，做一个平滑。不过我们同时也发现这3个单词分别为a, in, of这样的介词，完全可以在预处理时作为停用词干掉，这也从另一个方面说明TF-IDF的有效性。

有了tf-idf值表示的文档向量，我们就可以训练一个LSI模型，和Latent Semantic Indexing (LSI) A Fast Track Tutorial中的例子相似，我们设置topic数为2：

```
>>> lsi = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=2)
>>> lsi.print_topics(2)
2013-05-27 19:15:26,467 : INFO : topic #0(1.137): 0.438*"gold" + 0.438*"shipment" + 0.366*"truck" +
0.366*"arrived" + 0.345*"damaged" + 0.345*"fire" + 0.297*"silver" + 0.149*"delivery" + 0.000*"in" +
0.000*"a"
2013-05-27 19:15:26,468 : INFO : topic #1(1.000): 0.728*"silver" + 0.364*"delivery" + -0.364*"fire" +
-0.364*"damaged" + 0.134*"truck" + 0.134*"arrived" + -0.134*"shipment" + -0.134*"gold" + -0.000*"a" +
-0.000*"in"
```

lsi的物理意义不太好解释，不过最核心的意义是将训练文档向量组成的矩阵SVD分解，并做了一个秩为2的近似SVD分解，可以参考那篇英文tutorail。有了这个lsi模型，我们就可以将文档映射到一个二维的topic空间中：

```
>>> corpus_lsi = lsi[corpus_tfidf]
>>> for doc in corpus_lsi:
... print doc
...
[(0, 0.67211468809878649), (1, -0.54880682119355917)]
[(0, 0.44124825208697727), (1, 0.83594920480339041)]
[(0, 0.80401378963792647)]
```

可以看出，文档1，3和topic1更相关，文档2和topic2更相关；

我们也可以顺手跑一个LDA模型：

```
>>> lda = models.LdaModel(copurs_tfidf, id2word=dictionary, num_topics=2)
>>> lda.print_topics(2)
2013-05-27 19:44:40,026 : INFO : topic #0: 0.119*silver + 0.107*shipment + 0.104*truck + 0.103*gold +
0.102*fire + 0.101*arrived + 0.097*damaged + 0.085*delivery + 0.061*of + 0.061*in
2013-05-27 19:44:40,026 : INFO : topic #1: 0.110*gold + 0.109*silver + 0.105*shipment + 0.105*damaged
+ 0.101*arrived + 0.101*fire + 0.098*truck + 0.090*delivery + 0.061*of + 0.061*in
```

lda模型中的每个主题单词都有概率意义，其加和为1，值越大权重越大，物理意义比较明确，不过反过来再看这三篇文档训练的2个主题的LDA模型太平均了，没有说服力。

好了，我们回到LSI模型，有了LSI模型，我们如何来计算文档直接的相思度，或者换个角度，给定一个查询Query，如何找到最相关的文档？当然首先是建索引了：

```
>>> index = similarities.MatrixSimilarity(lsi[corpus])
2013-05-27 19:50:30,282 : INFO : scanning corpus to determine the number of features
2013-05-27 19:50:30,282 : INFO : creating matrix for 3 documents and 2 features
```

还是以这篇英文tutorial中的查询Query为例：gold silver truck。首先将其向量化：

```
>>> query = "gold silver truck"
>>> query_bow = dictionary.doc2bow(query.lower().split())
>>> print query_bow
[(3, 1), (9, 1), (10, 1)]
```

再用之前训练好的LSI模型将其映射到二维的topic空间：

```
>>> query_lsi = lsi[query_bow]
>>> print query_lsi
[(0, 1.1012835748628467), (1, 0.72812283398049593)]
```

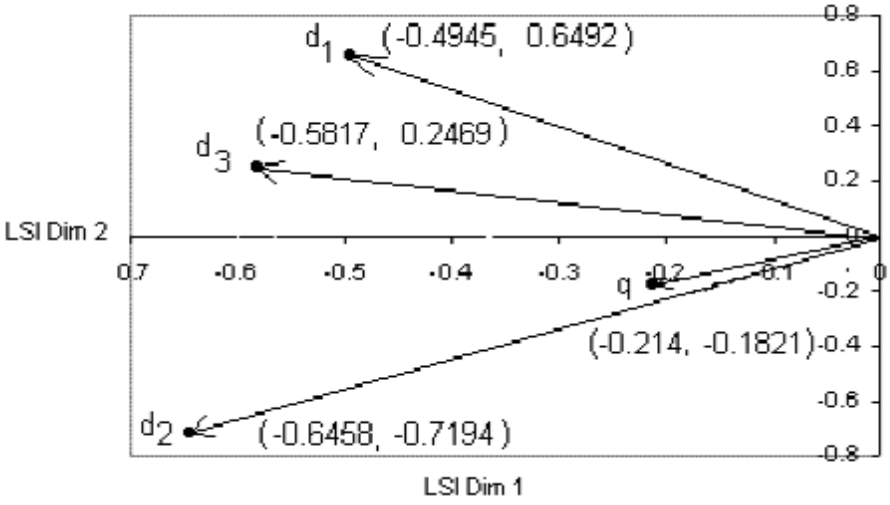
最后就是计算其和index中doc的余弦相似度了：

```
>>> sims = index[query_lsi]
>>> print list(enumerate(sims))
[(0, 0.40757114), (1, 0.93163693), (2, 0.83416492)]
```

当然，我们也可以按相似度进行排序：

```
>>> sort_sims = sorted(enumerate(sims), key=lambda item: -item[1])
>>> print sort_sims
[(1, 0.93163693), (2, 0.83416492), (0, 0.40757114)]
```

可以看出，这个查询的结果是doc2 > doc3 > doc1，和fast tutorial是一致的，虽然数值上有一些差别：



好了，这个例子就到此为止，下一节我们将主要说明如何基于gensim计算课程图谱上课程之间的主题相似度，同时考虑一些改进方法，包括借助英文的自然语言处理工具包NLTK以及用更大的维基百科的语料来看看效果。

未完待续...

注：原创文章，转载请注明出处“我爱自然语言处理”：www.52nlp.cn

本文链接地址：http://www.52nlp.cn/如何计算两个文档的相似度二

本条目发布于2013年05月27号 [http://www.52nlp.cn/%e5%a6%82%e4%bd%95%e8%ae%a1%e7%ae%97%e4%b8%a4%e4%b8%aa%e6%96%87%e6%a1%a3%e7%9a%84%e7%9b%b8%e4%bc%bc%e5%ba%a6%e4%ba%8c] 。属于Topic Model、推荐系统、自然语言处理分类，被贴了 gensim、LDA、LDA主题模型、LSA、LSI、nltk、NLTK应用、numpy、scipy、TF-IDF、Topic Model、主题模型、余弦相似度、向量空间模型、推荐系统、文本分析、文本相似度、文档相似度、浅层语义分析、浅层语义索引、维基百科语料、课程图谱 标签。作者是52nlp。

如何计算两个文档的相似度（一）

前几天，我发布了一个和在线教育相关的网站：课程图谱，这个网站的目的通过对公开课的导航、推荐和点评等功能方便大家找到感兴趣的公开课，特别是目前最火的Coursera，Udacity等公开课平台上的课程。在发布之前，遇到的一个问题是如何找到两个相关的公开课，最早的计划是通过用户对课程的关注和用户对用户的关注来做推荐，譬如“你关注的朋友也关注这些课程”，但是问题是网站发布之前，我还没有积累用户关注的数据。另外一个想法是提前给课程打好标签，通过标签来计算它门之间的相似度，不过这是一个人工标注的过程，需要一定的时间。当然，另一个很自然的想法是通过课程的文本内容来计算课程之间的相似度，公开课相对来说有很多的文本描述信息，从文本分析的角度来处理这种推荐系统的冷启动问题应该不失为一个好的处理方法。通过一些调研和之前的一些工作经验，最终考虑采用Topic model来解决这个问题，其实方案很简单，就是将两个公开课的文本内容映射到topic的维度，然后再计算其相似度。然后的然后就通过google发现了gensim这个强大的Python工具包，它的简介只有一句话：topic modelling for humans, 用过之后，只能由衷的说一句：感谢上帝，感谢Google，感谢开源！

当前课程图谱中所有课程之间的相似度全部基于gensim计算，自己写的调用代码不到一百行，topic模型采用LSI(Latent semantic indexing, 中文译为浅层语义索引），LSI和LSA（Latent semantic analysis，中文译为浅层语义分析）这两个名词常常混在一起，事实上，在维基百科上，有建议将这两个名词合二为一。以下是课程图谱的一个效果图，课程为著名的机器学习专家Andrew Ng教授在Coursera的机器学习公开课，图片显示的是主题模型计算后排名前10的相关课程，Andrew Ng教授同时也是Coursera的创始人之一：

主题相关的课程

Machine Learning (Coursera) 1 个评论 关注

Neural Networks for Machine Learning (Coursera) 2 个评论 关注

Fundamentals of Online Education: Planning and Application (Coursera) 0 个评论 关注

Introduction to Engineering Mechanics (Coursera) 0 个评论 关注

Natural Language Processing (Coursera) 2 个评论 关注

Artificial Intelligence Planning (Coursera) 1 个评论 关注

Metadata: Organizing and Discovering Information (Coursera) 0 个评论 关注

The Brain-Targeted Teaching® Model for 21st Century Schools (Coursera) 0 个评论 关注

Probabilistic Graphical Models (Coursera) 3 个评论 关注

First Year Teaching - Success from the Start (Coursera) 0 个评论 关注

最后回到这篇文章的主题，我将会分3个部分介绍，首先介绍一些相关知识点，不过不会详细介绍每个知识点的细节，主要是简要的描述一下同时提供一些互联网上现有的不错的参考资料，如果读者已经很熟悉，可以直接跳过去；第二部分我会介绍gensim的安装和使用，特别是如何计算课程图谱上课程之间的相似度的；第三部分包括如何基于全量的英文维基百科（400多万文章，压缩后9个多G的语料）在一个4g内存的macbook上训练LSI模型和LDA模型，以及如何将其应用到课程图谱上来改进课程之前的相似度的效果，注意课程图谱的课程内容主要是英文，目前的效果还是第二部分的结果，第三部分我们一起来实现。如果你的英文没问题，第二，第三部分可以直接阅读gensim的tutorail，我所做的事情主要是基于这个tutorail在课程图谱上做了一些验证。

一、相关的知识点及参考资料

这篇文章不会写很长，但是涉及的知识点蛮多，所以首先会在这里介绍相关的知识点，了解的同学可以一笑而过，不了解的同学最好能做一些预习，这对于你了解topic model以及gensim更有好处。如果以后时间允许，我可能会基于其中的某几个点写一篇比较详细的介绍性的文章。不过任何知识点首推维基百科，然后才是下面我所罗列的参考资料。

1）TF-IDF，余弦相似度，向量空间模型
这几个知识点在信息检索中是最基本的，入门级的参考资料可以看看吴军老师在《数学之美》中第11章“如何确定网页和查询的相关性”和第14章“余弦定理和新闻的分类”中的通俗介绍或者阮一峰老师写的两篇科普文章“TF-IDF与余弦相似性的应用（一）：自动提取关键词”和“TF-IDF与余弦相似性的应用（二）：找出相似文章”。

专业一点的参考资料推荐王斌老师在中科院所授的研究生课程“现代信息检索(Modern Information Retrieval)”的课件，其中“第六讲向量模型及权重计算”和该主题相关。或者更详细的可参考王斌老师翻译的经典的《信息检索导论》第6章或者其它相关的信息检索书籍。

2）SVD和LSI
想了解LSI一定要知道SVD（Singular value decomposition, 中文译为奇异值分解），而SVD的作用不仅仅局限于LSI，在很多地方都能见到其身影，SVD自诞生之后，其应用领域不断被发掘，可以不夸张的说如果学了线性代数而不明白SVD，基本上等于没学。想快速了解或复习SVD的同学可以参考这个英文tutorial: Singular Value Decomposition Tutorial，当然更推荐MIT教授Gilbert Strang的线性代数公开课和相关书籍，你可以直接在网易公开课看相关章节的视频。

关于LSI，简单说两句，一种情况下我们考察两个词的关系常常考虑的是它们在一个窗口长度（譬如一句话，一段话或一个文章）里的共现情况，在语料库语言学里有个专业点叫法叫Collocation，中文译为搭配或词语搭配。而LSI所做的是挖掘如下这层词语关系：**A**和**C**共现，**B**和**C**共现，目标是找到**A**和**B**的隐含关系，学术一点的叫法是**second-order co-ocurrence**。以下引用百度空间上一篇介绍相关参考资料时的简要描述：

LSI本质上识别了以文档为单位的second-order co-ocurrence的单词并归入同一个子空间。因此：
1）落在同一子空间的单词不一定是同义词，甚至不一定是在同情景下出现的单词，对于长篇文档尤其如是。
2）LSI根本无法处理一词多义的单词（多义词），多义词会导致LSI效果变差。

A persistent myth in search marketing circles is that LSI grants contextuality; i.e., terms occurring in the same context. This is not always the case. Consider two documents X and Y and three terms A, B and C and wherein:

A and B do not co-occur.
X mentions terms A and C
Y mentions terms B and C.

:. A---C---B

The common denominator is C, so we define this relation as an in-transit co-occurrence since both A and B occur while in transit with C. This is called second-order co-occurrence and is a special case of high-order co-occurrence.

其实我也推荐国外这篇由Dr. E. Garcia所写的SVD与LSI的通俗教程，这个系列最早是微博上有朋友推荐，不过发现英文原始网站上内容已经被其主人下架了，原因不得而知。幸好还有Google,在CSDN上我找到了这个系列“SVD与LSI教程系列”，不过很可惜很多图片都看不见了，如果哪位同学发现更好的版本或有原始的完整版本，可以告诉我，不甚感激！

不过幸好原文作者写了两个简要的PDF Tutorial版本：

Singular Value Decomposition (SVD)- A Fast Track Tutorial

Latent Semantic Indexing (LSI) A Fast Track Tutorial

这两个简明版本主要是通过简单的例子直观告诉你什么是SVD，什么是LSI，非常不错。

这几个版本的pdf文件我在微盘上上传了一个打包文件，也可以从这里下载：svd-lsi-doc.tar.gz

3) LDA
这个啥也不说了，隆重推荐我曾经在腾讯工作时的leader rickjin的"LDA数学八卦"系列，通俗易懂，娓娓道

来，另外rick的其他系列也是非常值得一读的。

未完待续...

注：原创文章，转载请注明出处“我爱自然语言处理”：www.52nlp.cn

本文链接地址：http://www.52nlp.cn/如何计算两个文档的相似度一

本条目发布于2013年05月18号 [http://www.52nlp.cn/%e5%a6%82%e4%bd%95%e8%ae%a1%e7%ae%97%e4%b8%a4%e4%b8%aa%e6%96%87%e6%a1%a3%e7%9a%84%e7%9b%b8%e4%bc%bc%e5%ba%a6%e4%b8%80] 。属于Topic Model、推荐系统、自然语言处理分类，被贴了 gensim、LDA、LSA、LSI、TF-IDF、Topic Model、主题模型、余弦相似度、向量空间模型、推荐系统、文本分析、文本相似度、文档相似度、浅层语义分析、浅层语义索引、课程图谱 标签。作者是52nlp。