

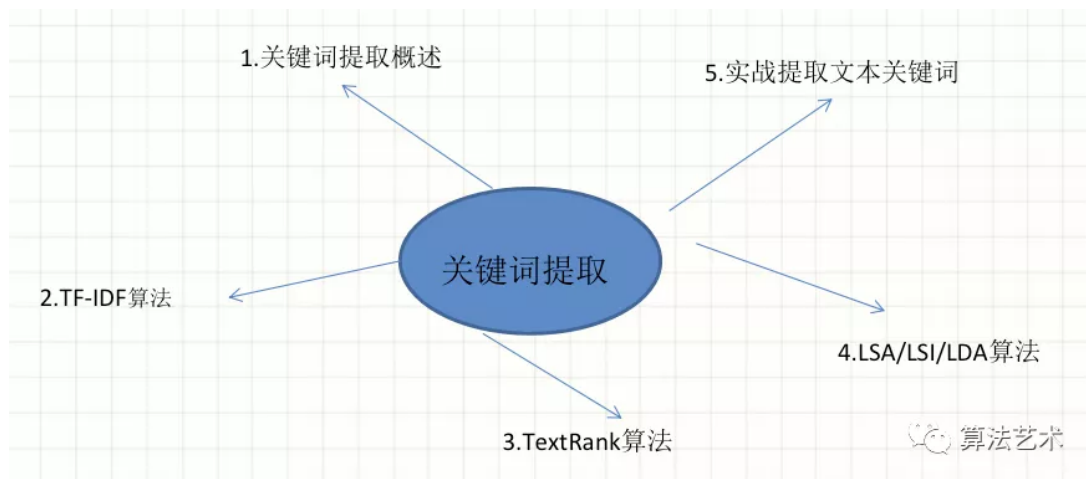
自然语言处理实战--关键词提取

原创 Paul 算法艺术 5月17日



前言

上篇讲了自然语言处理中词性标注和命名实体识别技术，这篇我们将讲解NLP中另外一个基础的技术--**关键词提取**。



注：本文约9800字，文字6000左右，代码270行，不喜欢看代码的可以忽略，没有复杂的公式推导，只有一下理论思想，阅读需要约20分钟，感谢关注。

概述

在信息爆炸的时代，很多信息无法全面接受，我们需要从中筛选出一些我们**感兴趣**的或者对我们**有用**的信息进行接收。

关键词提取就是解决这一类问题的很好的方法，该算法主要分为两类：

1.有监督方法：

主要是通过**分类**的方式进行的，通过构建一个较为丰富和完善的**词表**，然后通过判断每个**文档**与**词表**中每个词的**匹配程度**，以类似**打标签**的方式，达到关键词提取的效果

优点：可以获得**较高**的精度

缺点：1.需要大量标注数据，**人工成本高**

2.新增加的信息过多，**维护词表**人工成本也很高

2.无监督方法：

对数据要求较低，不需要词表，也不需要人工标注语料辅助训练

这类算法主要有：**TF-IDF TextRank LSA/LSI/LDA**

贰

算法

2.1 TF-IDF算法

TF-IDF算法（Term Frequency-Inverse Document Frequency,词频-逆文档词频），是一种基于统计的计算方法。

我们知道，一个词对一个文档的越重要，就越有可能是该文档的关键词。

该算法分两部分：1. TF算法 2. IDF算法

TF算法：是统计一个词在一篇文档中出现的频次(个数)，一个词如果在文档中出现的**次数**越多，对文档的**表达能力**也就越强。

例：我是中国人，热爱中国。

我(1),是(1),中国人(1),人(1),热爱(1),中国(2)

IDF算法：是统计一个词在文档集中，有多少个文档出现了这个词。如果一个词在**越少**的文档中出现，其对文档的**区分能力**就越强。

TF算法缺点：仅衡量词出现的频次，没有考虑词对文档的区分能力

IDF算法缺点：只强调词的区分能力，忽略了一个词在一篇文档中出现次数越多，能够更好的表现该文档的特征

经过许多学者对这两个算法的研究组合，将两种算法**相乘**，得到的效果比较好，

也就是从词频和逆文档词频两个角度对词重要性的衡量，就构成了**TF-IDF**

具体公式如下：

公式：

$$tf * idf(i, j) = \frac{n_{ij}}{\sum_k n_{kj}} * \log\left(\frac{|D|}{1 + |D_i|}\right)$$

分子： n_{ij} 表示词*i*在文档*j*中的出现频次
分母：统计文档中每次词出现次数的总和，也就是文档的总词数（对词频归一化）

$|D|$ 表示文档集总文档数
 $|D_i|$ 表示出现词*i*的文档数量
分母+1，是采用拉普拉斯算法艺术

我们举个例子，来解释这个公式

例：世界献血日，学校团体、献血服务志愿者等可到血液中心参观检验加工过程，我们会对检验结果进行公示，同时血液的价格也将进行公示。

TF: “献血”“血液”“进行”“公示”，出现次数均为2，tf值都是0.067

假设：有文档集1000篇，出现“献血”“血液”“进行”“公示”的文档数分别是10、15、100、50

$$\text{idf}(\text{献血}) = \log(1000 / 10) = 10$$

$$\text{idf}(\text{血液}) = \log(1000 / 15) = 4.20$$

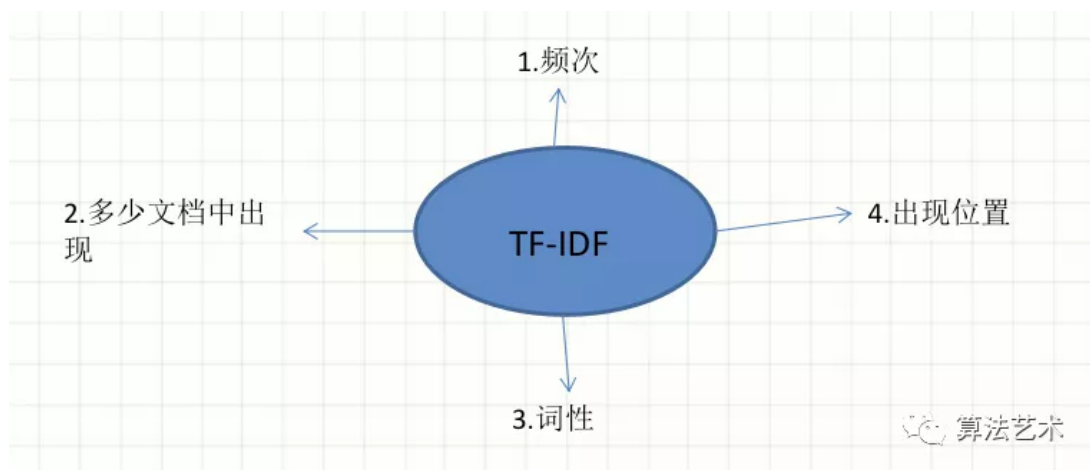
$$\text{idf}(\text{进行}) = \log(1000 / 100) = 2.30$$

$$\text{idf}(\text{公示}) = \log(1000 / 50) = 3.0$$

我们最后可以得到，“**献血**”的tf-idf值最高，最适合为这篇文档的**关键词**

当然关键词可以选择不止一个，可以同tf-idf值进行大到小排序，取k个关键词

TF-IDF只是对词频和逆文档词频进行了统计，而一篇文档中还有很多重要信息可以用于关键词的提取，如：



词性和**出现的位置**也是重要的信息，其中**名词**作为一种定义现实实体的词，带有更多的关键信息，给予更高**权重**

2.2 TextRank算法

与其他算法不同的一点是，其他算法的关键词提取都要基于一个现成的语料库

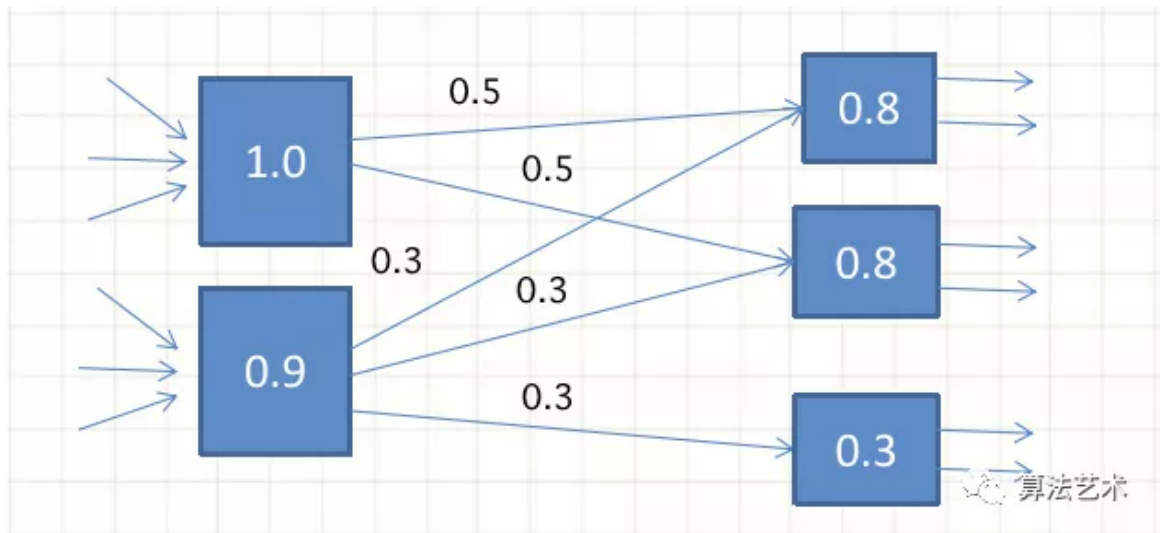
TF-IDF ----> 统计预料库中多少个文档出现过

主题模型 ----> 对大规模文档学习

而**TextRank**则**脱离**语料库的背景，仅对**单篇**文档进行分析就可以提取该文档的关键词。

基本思想来源于Google的**PageRank**算法

PageRank算法是Google创始人拉里-佩奇和谢尔盖-布林于1997年构建早期的搜索系统原型时提出的链接分析算法，用来评价搜索系统覆盖过网页重要性的一种方法。



PageRank算法是一种网页排名算法，基本思想有两条：

1. 链接数量：一个网页被**越多**的其他网页链接，说明这个网页越重要
2. 链接质量：一个网页被一个**越高权重**的网页链接，也表明这个网页越重要。

PageRank算法公式：

$In(V_i)$ 为 V_i 的入链集合

$Out(V_j)$ 为 V_j 的出链集合

$|Out(V_j)|$ 则是出链的数量

$S(V_i)$ 表示当前网页自身分数值

公式：

$$S(V_i) = \sum_{j \in In(V_i)} \left(\frac{1}{|Out(V_j)|} * S(V_j) \right)$$

算法艺术

上面会有个问题：

如上式计算导致一些**孤立**的网页（没有出链和入链的网页）的得分会为**0**，这样的话，这些网页就**不会**被访问到。

解决：

加入一个**阻尼系数d**，这样即使一个网页是孤立的，其自身也会有得分

改进公式：

$$S(V_i) = (1 - d) + d * \sum_{j \in (V_i)} \left(\frac{1}{|Out(V_j)|} * S(V_j) \right)$$

算法艺术

以上就是PageRank的理论，也是TextRank的理论基础

不同点是：

PageRank是**有向无权图**，

而TextRank进行自动摘要则是**有权图**，因为在计时除了考虑链接句的重要性外，还要考虑两个句子间的**相似性**。

计算每个句子给他链接句的贡献时，就不是通过平均分配的方式，而是通过计算**权重占总权重的比例**来分配（这里的权重就是两个句子之间的相似度，如编辑距离，余弦相似度等）。

公式：

$$WS(V_i) = (1 - d) + d * \sum_{j \in (V_i)} \left(\frac{w_{ji}}{|Out(V_j)|} * WS(V_j) \right)$$

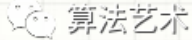
算法艺术

当TextRank应用到关键词提取时，与应用在自动摘要中时主要有两点不同

1.词与词之间关联没有权重

2.每个词不是与文档中所有词都有链接

由于第一个不同，就和PageRank一样，将得分平均贡献给每个链接的词

$$S(V_i) = (1 - d) + d * \sum_{j \in (V_i)} \left(\frac{1}{|Out(V_j)|} * S(V_j) \right)$$


由于第二个不同：

既然每个词不是与所有词相连，那么链接关系如何界定？

我们提出**滑动窗口**的概念，将文本切分成不同的模块

举个例子，解释一下滑动窗口的概念

例如：

世界献血日，学校团体、献血服务志愿者等可到血液中心参观检验加工过程，我们会对检验结果进行公示，同时血液的价格也将进行公示。

先做分词处理，分词后：

【世界，献血，日，学校，团体，服务，志愿者，等】，设置窗口长度为5

使用滑动窗口获取的结果：

- 1.【世界，献血，日，学校，团体】
- 2.【献血，日，学校，团体，服务】
- 3.【日，学校，团体，服务，志愿者】
- 4.【学校，团体，服务，志愿者，等】

2.3 LSA/LSI算法

一般来说TF-IDF和TextRank算法就能满足大部分关键词提取任务。但是有些文档关键词不一定是**显式**的出现在文档中

如：一篇讲解动物生成环境的科普文章，通篇解释狮子、大象、老虎等动物情况，文中没有出现**“动物”**，这样情况，前两种算法就无法提取动物这个**隐含**的主题信息。

就需要主题模型：LSA (LSI) 和LDA

LSA主要采用**SVD**方法进行暴力破解

LDA则是通过贝叶斯学派的方法对**分布信息**进行拟合

LSA (Latent Semantic Analysis, 潜在语义分析), 1988年, 美国贝尔通讯实验室的S.T.Ducmais等人了解决传统向量空间模型对文本的语义信息利用能力匮乏的问题, 提出了潜在语义分析概念

LSI (Latent Semantic Index, 潜在语义索引)

通常二者被认为是同一种算法

LSI只是在潜在语义分析后, 对结果进行建立相关**索引**

LSA主要步骤:

1.使用BOW模型将每个文档表示为向量

2.将所有的文档词向量拼接起来构成词-文档矩阵($m \times n$)

3.对词-文档矩阵进行SVD分解($[m \times r] \quad [r \times r] \quad [r \times n]$)

4.对SVD的结果, 将词-文档矩阵映射到一个更低纬度 k 的近似SVD结果, 每个词和文档都表示 k 个主题构成的空间中的一个点, 通过计算词和文档的相似度, 可以得到每个文档中对每个词的相识度结果, 取相似度最高的一个词即为文档的关键词

2.3 LDA算法

LDA (Latent Dirichlet Allocation, 隐含狄利克雷分布) 是由David Blei等人在2003年提出, 该方法理论基础是**贝叶斯**理论, LDA根据此的共现信息的分析, 拟合出 词-文档-主题的分布, 进而将词、文本都映射到一个语义空间中

LDA算法假设:

文档中主题的先验分布和主题中词的先验分布都服从

狄利克雷分布 (这也是LDA名字的由来)

而贝叶斯学派:

先验分布 + 数据 (似然) = 后验分布

所有就可以通过对数据集的统计, 得到每篇文档中主题和主题中词的多项式分布, 然后在利用贝叶斯学派, 推断出文档中主题和主题中词的后验分布

主要求解方法：吉布斯采样

结合吉布斯采样的LDA模型训练过程：

1. 随机初始化，对语料中每篇文档中的每个词 w ，随机地赋予一个topic编号 z
2. 重新扫描语料库，对每个词 w 按照吉布斯采样公式重新采样它的topic，在语料中进行更新
3. 重复以上语料库的重新采样过程直到吉布斯采样收敛
4. 统计语料库的topic=word共现频率矩阵，该矩阵就是LDA的模型

上面过程训练好了LDA模型，然后按照下面方式对新文档的topic进行预估：

1. 随机初始化，对文档中每篇文档中的每个词 w ，随机地赋予一个topic编号 z
2. 重新扫描当前文档，按照吉布斯采样公式，重新采样它的topic
3. 重复以上过程直到吉布斯采样收敛
4. 统计文档重点的topic分布即为预估的结果

LDA具体流程看起来似乎并不复杂，但是里面有很多需要注意的地方：

1. 怎样确定共轭分布中的超参
2. 具体怎样实现吉布斯采样
3. 每个一环节都有许多复杂的数学推导过程



代码实现

上面我们介绍了很多关键词提取的算法，下面我们就具体用代码实现一下：
将使用jieba和Gensim开源工具

```
1  
2  
3 import math  
4
```

```
5 import jieba
6 import jieba.posseg as psg
7 from gensim import corpora, models
8 from jieba import analyse
9 import functools
10
11
12 # 停用词表加载方法
13 def get_stopword_list():
14     # 停用词表存储路径，每一行为一个词，按行读取进行加载
15     # 进行编码转换确保匹配准确率
16     stop_word_path = './stopword.txt'
17     stopwords_list = [sw.replace('\n', '') for sw in open(stop_word_path, encoding='utf-8')]
18     return stopwords_list
19
20
21 # 分词方法，调用结巴接口
22 def seg_to_list(sentence, pos=False):
23     if not pos:
24         # 不进行词性标注的分词方法
25         seg_list = jieba.cut(sentence)
26     else:
27         # 进行词性标注的分词方法
28         seg_list = psg.cut(sentence)
29     return seg_list
30
31
32 # 去除干扰词
33 def word_filter(seg_list, pos=False):
34     stopwords_list = get_stopword_list()
35     filter_list = []
36     # 根据POS参数选择是否词性过滤
37     ## 不进行词性过滤，则将词性都标记为n，表示全部保留
38     for seg in seg_list:
39         if not pos:
40             word = seg
41             flag = 'n'
42         else:
43             word = seg.word
44             flag = seg.flag
```

```
45         if not flag.startswith('n'):
46             continue
47         # 过滤停用词表中的词，以及长度为<2的词
48         if not word in stopword_list and len(word) > 1:
49             filter_list.append(word)
50
51     return filter_list
52
53
54 # 数据加载，pos为是否词性标注的参数，corpus_path为数据集路径
55 def load_data(pos=False, corpus_path='./corpus.txt'):
56     # 调用上面方式对数据集进行处理，处理后的每条数据仅保留非干扰词
57     doc_list = []
58     for line in open(corpus_path, 'r', encoding='utf-8'):
59         content = line.strip()
60         seg_list = seg_to_list(content, pos)
61         filter_list = word_filter(seg_list, pos)
62         doc_list.append(filter_list)
63
64     return doc_list
65
66
67 # idf值统计方法
68 def train_idf(doc_list):
69     idf_dic = {}
70     # 总文档数
71     tt_count = len(doc_list)
72
73     # 每个词出现的文档数
74     for doc in doc_list:
75         for word in set(doc):
76             idf_dic[word] = idf_dic.get(word, 0.0) + 1.0
77
78     # 按公式转换为idf值，分母加1进行平滑处理
79     for k, v in idf_dic.items():
80         idf_dic[k] = math.log(tt_count / (1.0 + v))
81
82     # 对于没有在字典中的词，默认其仅在一个文档出现，得到默认idf值
83     default_idf = math.log(tt_count / (1.0))
84     return idf_dic, default_idf
```

```
85
86
87 # 排序函数，用于topK关键词的按值排序
88 def cmp(e1, e2):
89     import numpy as np
90     res = np.sign(e1[1] - e2[1])
91     if res != 0:
92         return res
93     else:
94         a = e1[0] + e2[0]
95         b = e2[0] + e1[0]
96         if a > b:
97             return 1
98         elif a == b:
99             return 0
100     else:
101         return -1
102
103 # TF-IDF类
104 class TfIdf(object):
105     # 四个参数分别是：训练好的idf字典，默认idf值，处理后的待提取文本，关键词数量
106     def __init__(self, idf_dic, default_idf, word_list, keyword_num):
107         self.word_list = word_list
108         self.idf_dic, self.default_idf = idf_dic, default_idf
109         self.tf_dic = self.get_tf_dic()
110         self.keyword_num = keyword_num
111
112     # 统计tf值
113     def get_tf_dic(self):
114         tf_dic = {}
115         for word in self.word_list:
116             tf_dic[word] = tf_dic.get(word, 0.0) + 1.0
117
118         tt_count = len(self.word_list)
119         for k, v in tf_dic.items():
120             tf_dic[k] = float(v) / tt_count
121
122         return tf_dic
123
124     # 按公式计算tf-idf
```

```
125     def get_tfidf(self):
126         tfidf_dic = {}
127         for word in self.word_list:
128             idf = self.idf_dic.get(word, self.default_idf)
129             tf = self.tf_dic.get(word, 0)
130
131             tfidf = tf * idf
132             tfidf_dic[word] = tfidf
133
134         tfidf_dic.items()
135         # 根据tf-idf排序, 去排名前keyword_num的词作为关键词
136         for k, v in sorted(tfidf_dic.items(), key=functools.cmp_to_key(cmp)):
137             print(k + "/ ", end='')
138         print()
139
140
141 # 主题模型
142 class TopicModel(object):
143     # 三个传入参数: 处理后的数据集, 关键词数量, 具体模型 (LSI、LDA), 主题数量
144     def __init__(self, doc_list, keyword_num, model='LSI', num_topics=4):
145         # 使用gensim的接口, 将文本转为向量化表示
146         # 先构建词空间
147         self.dictionary = corpora.Dictionary(doc_list)
148         # 使用BOW模型向量化
149         corpus = [self.dictionary.doc2bow(doc) for doc in doc_list]
150         # 对每个词, 根据tf-idf进行加权, 得到加权后的向量表示
151         self.tfidf_model = models.TfidfModel(corpus)
152         self.corpus_tfidf = self.tfidf_model[corpus]
153
154         self.keyword_num = keyword_num
155         self.num_topics = num_topics
156         # 选择加载的模型
157         if model == 'LSI':
158             self.model = self.train_lsi()
159         else:
160             self.model = self.train_lda()
161
162         # 得到数据集的主题-词分布
163         word_dic = self.word_dictionary(doc_list)
164         self.wordtopic_dic = self.get_wordtopic(word_dic)
```

```
165
166     def train_lsi(self):
167         lsi = models.LsiModel(self.corpus_tfidf, id2word=self.dictionary, num_topics=self.num_topics)
168         return lsi
169
170     def train_lda(self):
171         lda = models.LdaModel(self.corpus_tfidf, id2word=self.dictionary, num_topics=self.num_topics)
172         return lda
173
174     def get_wordtopic(self, word_dic):
175         wordtopic_dic = {}
176
177         for word in word_dic:
178             single_list = [word]
179             wordcorpus = self.tfidf_model[self.dictionary.doc2bow(single_list)]
180             wordtopic = self.model[wordcorpus]
181             wordtopic_dic[word] = wordtopic
182         return wordtopic_dic
183
184     # 计算词的分布和文档的分布的相似度，取相似度最高的keyword_num个词作为关键词
185     def get_simword(self, word_list):
186         sentcorpus = self.tfidf_model[self.dictionary.doc2bow(word_list)]
187         senttopic = self.model[sentcorpus]
188
189         # 余弦相似度计算
190         def calsim(l1, l2):
191             a, b, c = 0.0, 0.0, 0.0
192             for t1, t2 in zip(l1, l2):
193                 x1 = t1[1]
194                 x2 = t2[1]
195                 a += x1 * x1
196                 b += x1 * x2
197                 c += x2 * x2
198             sim = a / math.sqrt(b * c) if not (b * c) == 0.0 else 0.0
199             return sim
200
201         # 计算输入文本和每个词的主题分布相似度
202         sim_dic = {}
203         for k, v in self.wordtopic_dic.items():
204             if k not in word_list:
```

```
205         continue
206         sim = calsim(v, senttopic)
207         sim_dic[k] = sim
208
209     for k, v in sorted(sim_dic.items(), key=functools.cmp_to_key(cmp), r
210         print(k + "/" , end='')
211     print()
212
213     # 词空间构建方法和向量化方法, 在没有gensim接口时的一般处理方法
214     def word_dictionary(self, doc_list):
215         dictionary = []
216         for doc in doc_list:
217             dictionary.extend(doc)
218
219         dictionary = list(set(dictionary))
220
221         return dictionary
222
223     def doc2bowvec(self, word_list):
224         vec_list = [1 if word in word_list else 0 for word in self.dictionar
225         return vec_list
226
227
228     def tfidf_extract(word_list, pos=False, keyword_num=10):
229         doc_list = load_data(pos)
230         idf_dic, default_idf = train_idf(doc_list)
231         tfidf_model = TfIdf(idf_dic, default_idf, word_list, keyword_num)
232         tfidf_model.get_tfidf()
233
234
235     def textrank_extract(text, pos=False, keyword_num=10):
236         textrank = analyse.textrank
237         keywords = textrank(text, keyword_num)
238         # 输出抽取出的关键词
239         for keyword in keywords:
240             print(keyword + "/" , end='')
241         print()
242
243
244     def topic_extract(word_list, model, pos=False, keyword_num=10):
```



```

245     doc_list = load_data(pos)
246     topic_model = TopicModel(doc_list, keyword_num, model=model)
247     topic_model.get_simword(word_list)
248
249
250 if __name__ == '__main__':
251     text = '6月19日,《2012年度“中国爱心城市”公益活动新闻发布会》在京举行。' + \
252           '中华社会救助基金会理事长许嘉璐到会讲话。基金会高级顾问朱发忠,全国老龄' -
253           '办副主任朱勇,民政部社会救助司助理巡视员周萍,中华社会救助基金会副理事长耿
254           '重庆市民政局巡视员谭明政。晋江市人大常委会主任陈健倩,以及10余个省、市、
255           '领导及四十多家媒体参加了发布会。□中华社会救助基金会秘书长时正新介绍本年
256           '市”公益活动将以“爱心城市宣传、孤老关爱救助项目及第二届中国爱心城市大会”
257           '、呼和浩特市、长沙市、太原市、蚌埠市、南昌市、汕头市、沧州市、晋江市及遵
258           '这一公益活动。□中国雅虎副总编张银生和凤凰网城市频道总监赵耀分别以各自媒
259           '的宣传方案。□会上,中华社会救助基金会与“第二届中国爱心城市大会”承办方晋江
260           '事长接受晋江市参与“百万孤老关爱行动”向国家重点扶贫地区捐赠的价值400万元
261           '常委会主任陈健倩介绍了大会的筹备情况。'
262
263     pos = True
264     seg_list = seg_to_list(text, pos)
265     filter_list = word_filter(seg_list, pos)
266
267     print('TF-IDF模型结果: ')
268     tfidf_extract(filter_list)
269     print('TextRank模型结果: ')
270     textrank_extract(text)
271     print('LSI模型结果: ')
272     topic_extract(filter_list, 'LSI', pos)
273     print('LDA模型结果: ')
274     topic_extract(filter_list, 'LDA', pos)
275
276

```

结果对比

1: 不选择词性过滤, 各种词性的词都可能被选为关键词

TF-IDF模型结果:

晋江市 / 救助 / 城市 / 大会 / 爱心 / 中华 / 基金会 / 陈健倩 / 重庆市 / 许嘉璐 /

TextRank模型结果：

城市 / 爱心 / 救助 / 中国 / 社会 / 晋江市 / 基金会 / 大会 / 介绍 / 公益活动/

LSI模型结果：

中华 / 活动 / 宣传 / 爱心 / 行动 / 中国 / 基金会 / 项目 / 救助 / 社会/

LDA模型结果：

晋江市 / 签约 / 民政部 / 大会 / 第二届 / 许嘉璐 / 重庆市 / 人大常委会 / 巡视员 / 孤老/

2: 选择词性过滤，仅选用名词都作为关键词的结果

TF-IDF模型结果：

晋江市 / 城市 / 大会 / 爱心 / 中华 / 基金会 / 陈健倩 / 重庆市 / 许嘉璐/ 巡视员 /

TextRank模型结果：

城市 / 爱心 / 中国 / 社会 / 晋江市 / 基金会 / 大会 / 公益活动/

LSI模型结果：

中国 / 中华 / 爱心 / 基金会 / 项目 / 社会 / 城市/ 公益活动/ 全国/ 国家

LDA模型结果：

晋江市 / 公益活动 / 年度 / 民政局 / 大会 / 新闻 / 许嘉璐 / 巡视员 / 陈健倩 / 人大常委会 /

————| The End |————

参考资料：

1:《统计学习方法》--- 李航

2:《Python自然语言处理实战》-- 涂铭，刘祥，刘树春

3: 代码和数据：来自《Python自然语言处理实战》

大家可自行下载：<https://github.com/nlpinaction/learning-nlp>

大家加油！