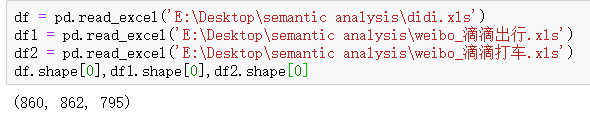
1. 百度贴吧、微博社区’滴滴出行’网约车评论文本信息展示，其中百度贴吧数据860条，微博社区滴滴出行862条，微博社区滴滴打车795条，共2517条评论数据进行分析

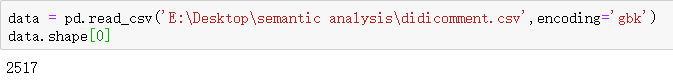
|  |  |
| --- | --- |
| 1 | 经常用，一款实用的APP，希望能再多点优惠 |
| 2 | 并且打车的发票在手机端就可以直接开出，而且我觉得打车的补贴还是比较多的，滴滴打车的优势是方便... |
| 3 | 滴滴出行是个不错的出行平台，最大限度的提高了出行者的权益，但在司机的准入上要求不严，以致连续... |
| 4 | 滴滴就是出了事情或者责任就会逃避 |
| 5 | 很想给滴滴好评的，因为它确实方便我出行了，而且滴滴司机给我的印象也不错。无奈最近它的大众形象... |

2.消费者关于网约车的评论文本数据预处理

（1）文本去重

剔除重复的文本数据，加强数据的可用性，利用python的drop\_duplicates（）函数对dataframe结构进行去重处理

文本数据去重之前数据为2517条



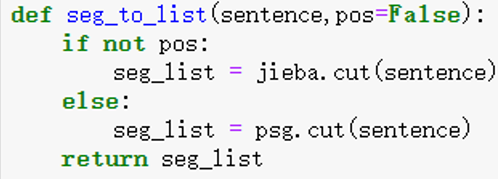
文本数据去重之后数据为2482条



（2）基于语义网络的消费者网约车评论分析分词处理

加载需要处理的文本数据，利用Python的jieba库进行分词处理，利用jieba库的cut方法进行分词处理，文本数据分词部分结果展示：

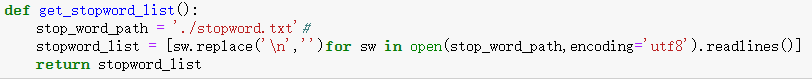
[['一款', '实用', 'APP', '希望', '多点', '优惠'],['打车','发票','手机',开出','打车','补贴','滴滴','打车','优势','方便快捷','一竿子','推翻','滴滴','社会','行业','贡献','网友','历史','出门在外','流量','变得','完善','支付','方式','一会','专车','来接','滴滴','女子','外出','遭受','侵害','例子','确实','出行','价格','明朗','十点','县城']]



（3）停用词剔除处理

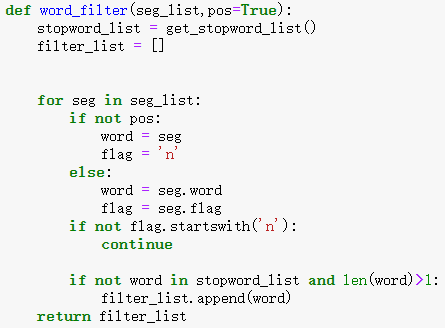
要对去重后的文本数据进行停用词剔除处理，加载库中的停用词表，对文本数据进行处理，剔除已经停用的词

停用词表部分词语展示：一时 一来 一样 一次 一片 一番 一直 一致 一般 一起 一转眼 一边 一面 七 万一 三 三天两头 三番两次 三番五次 上 上下 上升 上去 上来 上述 上面 下 下列 下去 下来 下面 不 不一 不久 不了 不亦乐乎 不仅 不仅...而且 不仅仅 不仅仅是 不会 不但 不但...而且 不光 不免 不再 不力 不单



（4）干扰词过滤处理

经常在数据中出现但是却没有多大用处的词汇过滤，过滤词中词以及长度<2的词



4.利用Ucient6软件对文本数据进行语义网络分析

采用语义网络分析对自然语言进行处理，分解输入文本数据中的句法关系，分析句子的深层格结构，记录语义关系，关键词提取及中心度分析是社会网络分析中十分重要的部分，因此对文本进行关键词提取进行中心度分析，而中心度主要三个指标为点度中心性（Degree Centrality）、中间中心性（Between Centrality）、特征向量中心性（Eigenvector Centrality）

1. Tf-idf关键词提取：

在一份给定的文件里，词频指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化，以防止它偏向长的文件。逆向文件频率是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。

TFIDF的主要思想是：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TFIDF实际上是：TF \* IDF，TF词频，IDF反文档频率。TF表示词条在文档d中出现的频率指的是某一个给定的词语在该文件中出现的次数。IDF的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大，则说明词条t具有很好的类别区分能力。如果某一类文档C中包含词条t的文档数为m，而其它类包含t的文档总数为k，显然所有包含t的文档数n=m+k，当m大的时候，n也大，按照IDF公式得到的IDF的值会小，就说明该词条t类别区分能力不强。是指果包含词条的文档越少，IDF越大，则说明词条具有很好的类别区分能力。

      在一份给定的文件里，词频指的是某一个给定的词语在该文件中出现的频率。这个数字是对词数的归一化，以防止它偏向长的文件。对于在某一特定文件里的词语  t_{i}  来说，它的重要性可表示为：

 \mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}

      以上式子中 n_{i,j} 是该词在文件d_{j}中的出现次数，而分母则是在文件d_{j}中所有字词的出现次数之和。

      逆向文件频率是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到：

 \mathrm{idf_{i}} =  \log \frac{|D|}{|\{j: t_{i} \in d_{j}\}|}

其中|D|：语料库中的文件总数 |\{ j: t_{i} \in d_{j}\}| ：包含词语 t_{i} 的文件数目（即 n_{i,j} \neq 0的文件数目）如果该词语不在语料库中，就会导致被除数为零，因此一般情况下使用1 + |\{j : t_{i} \in d_{j}\}|  \mathrm{tf{}idf_{i,j}} = \mathrm{tf_{i,j}} \times  \mathrm{idf_{i}} 

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。

文本总体高频词汇（top36）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| words | frequency | words | frequency | words | frequency | words | frequency |
| DiDi | 5415 | master | 220 | online taxi-hailing | 169 | condition | 100 |
| driver | 1670 | fast ride | 216 | rubbish | 159 | Shuttle transfer | 98 |
| passenger | 584 | phone | 210 | Beijing | 148 | place | 98 |
| customer | 530 | player | 209 | video | 143 | world | 94 |
| servicer | 509 | Following wind | 207 | hour | 140 | Liu Qing | 87 |
| game | 507 | cellphone | 204 | software | 124 | user | 87 |
| company | 408 | Order form | 190 | get on | 120 | function | 87 |
| full text | 408 | time | 189 | Car owner | 118 | friend | 86 |
| platform | 227 | taxi | 172 | system | 111 | city | 76 |

1. 中心度分析及网络图生成

将文本数据处理成以分号分隔的格式，如：

滴滴;感觉;出租车;滴滴;价格;方式;滴滴;感觉;力度;感觉;滴滴;专职;司机

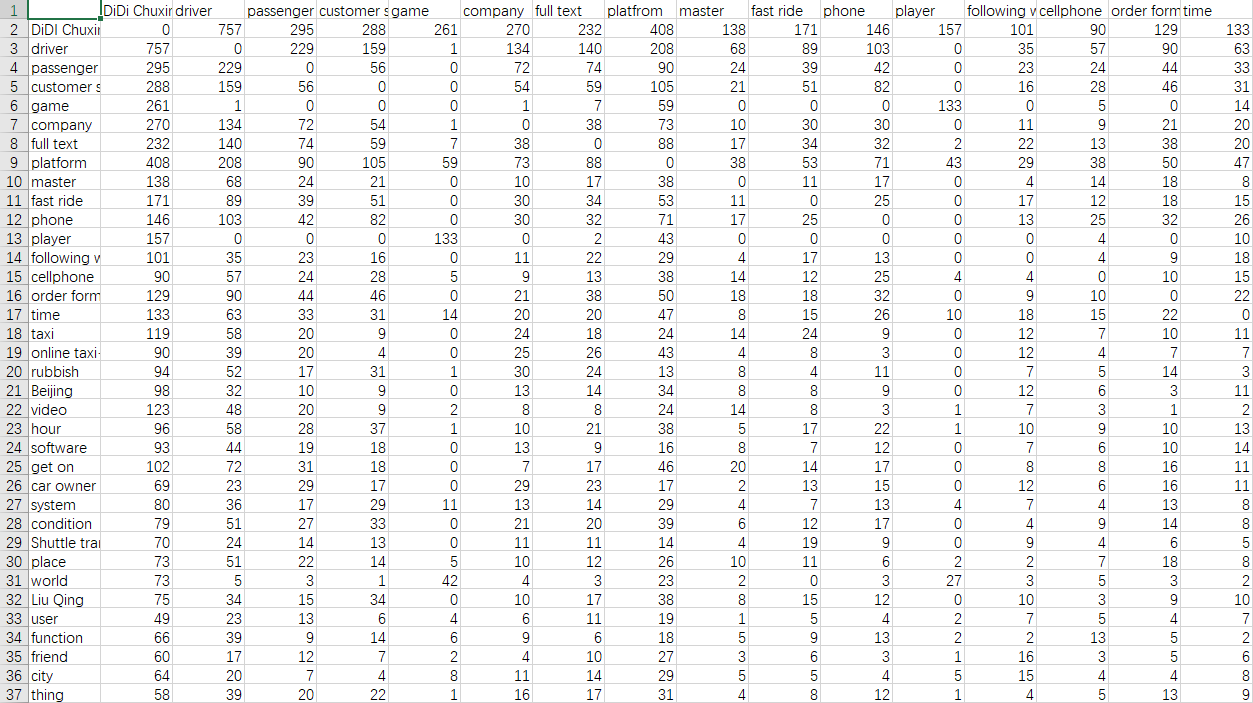
滴滴;行程;地点;地点;方案

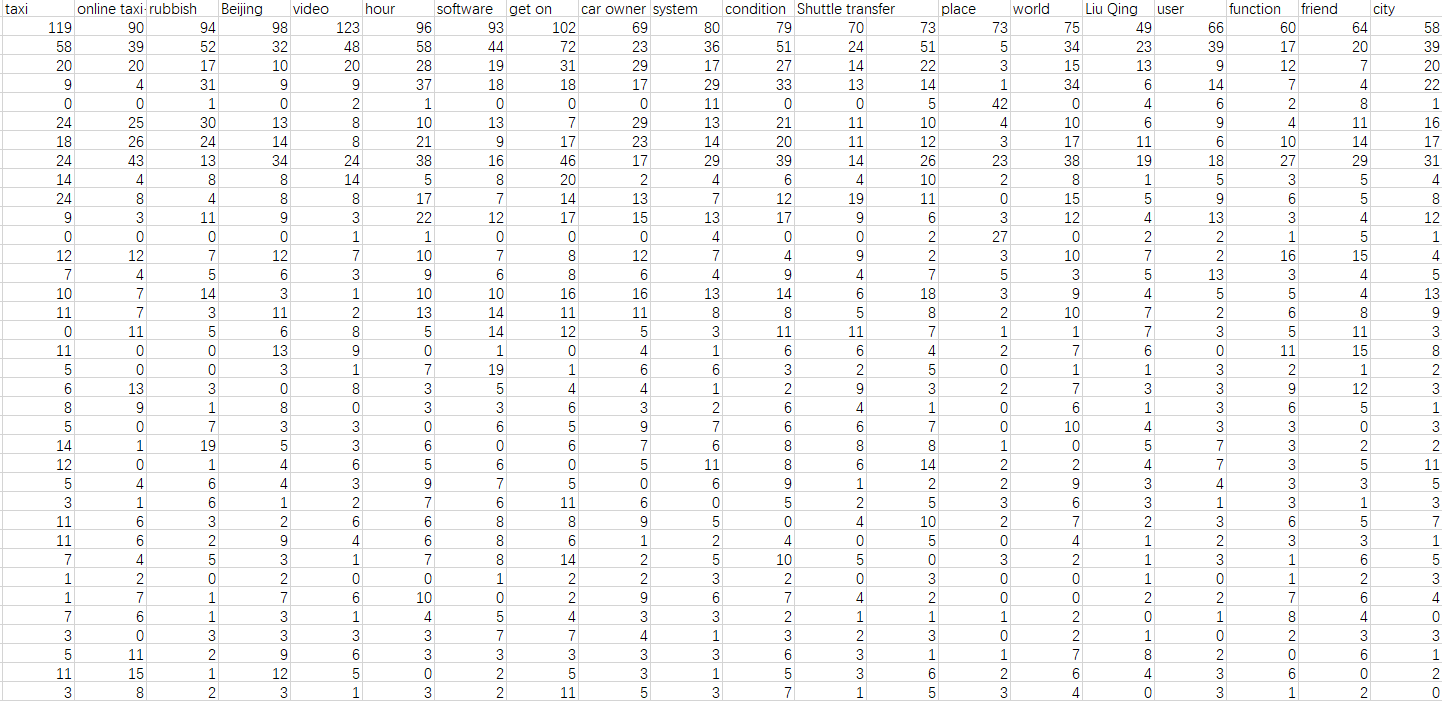
滴滴;消失;消失;习惯

滴滴;顺风;消费者;滴滴;快车;滴滴;快车;事件;滴滴

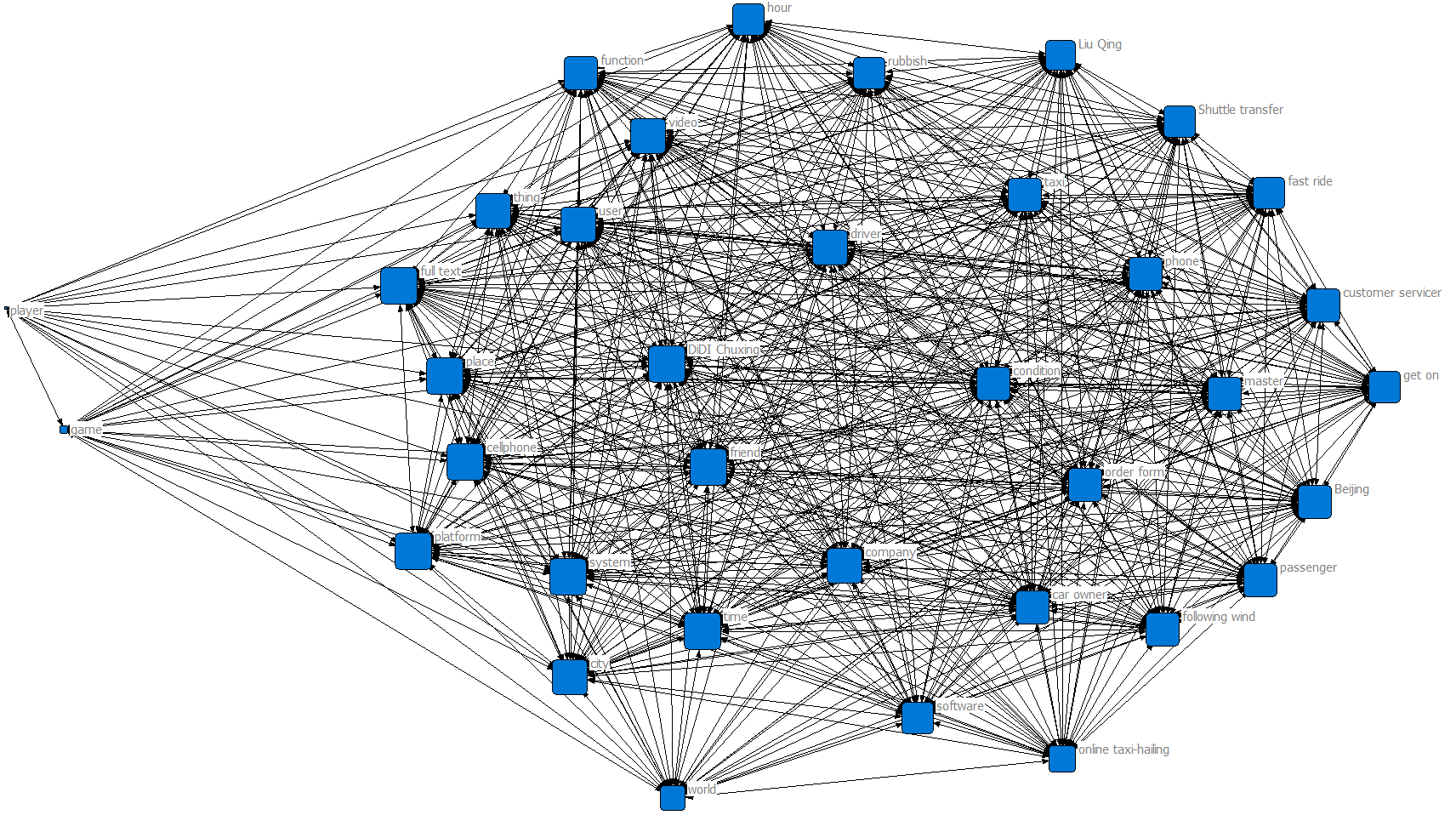
再用Ucient6软件导入数据生成词频数，高频词汇选取过多，绘制出来的网络图不利于观察，经过多次实验与尝试，取其中词频数top36生成36x36共现矩阵，利用共现矩阵绘制网络图

Top36关键词生成的36x36共现矩阵:

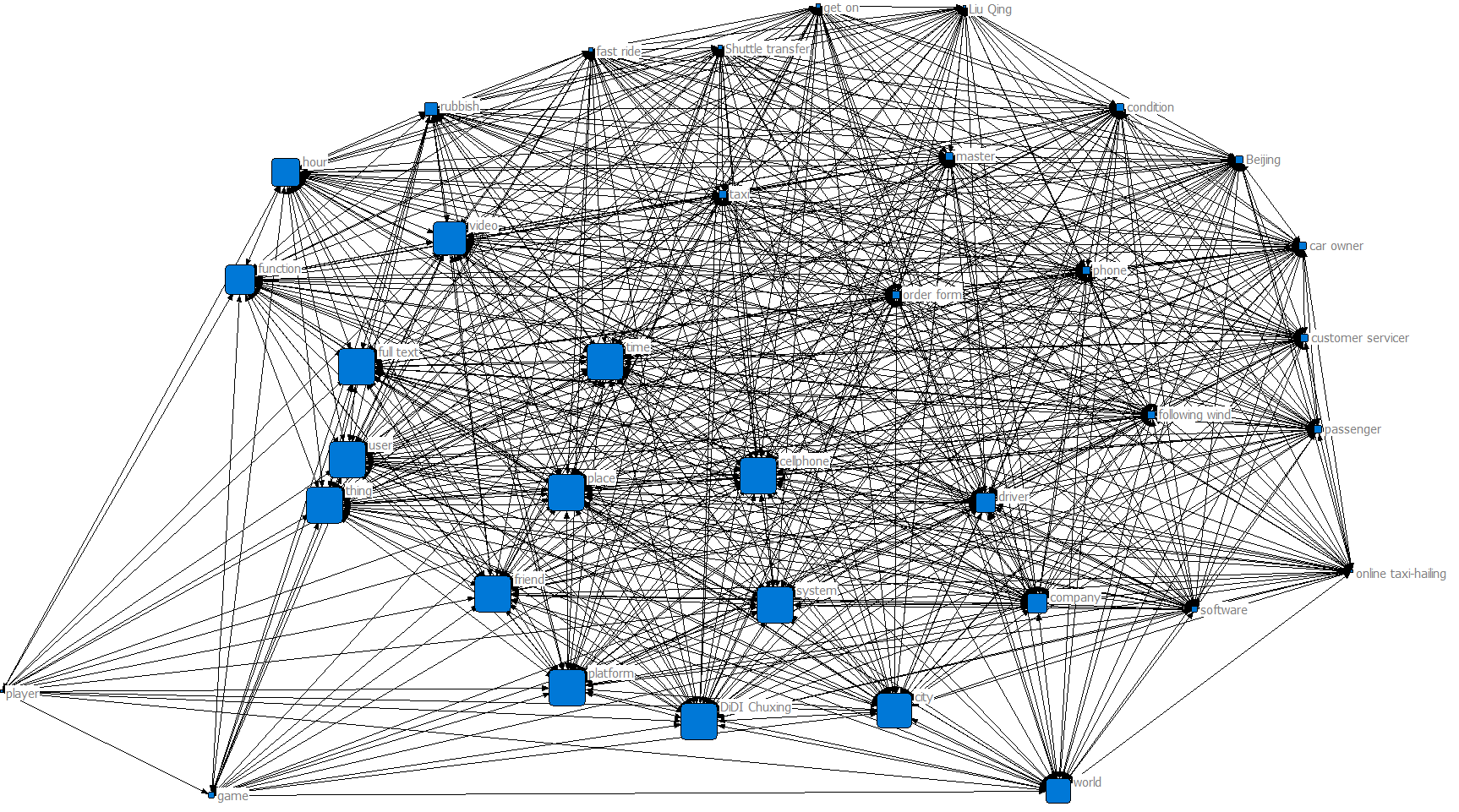




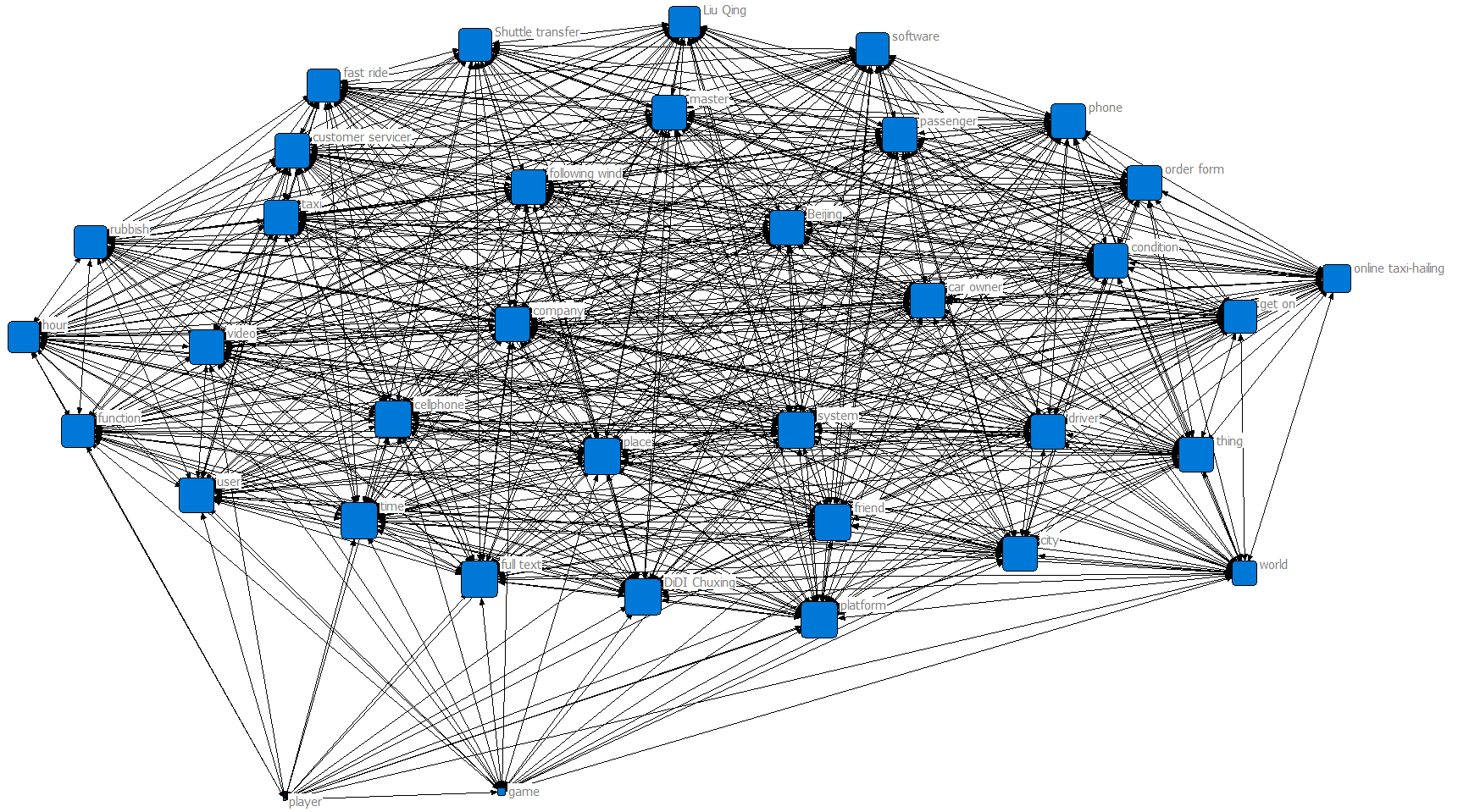
将Ucient6生成的36x36共现矩阵导入Netdraw中进行可视化，形成网约车在线评论语义网络图degree centrality分析图：



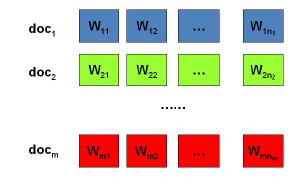
betweennesss centrality分析图如下:



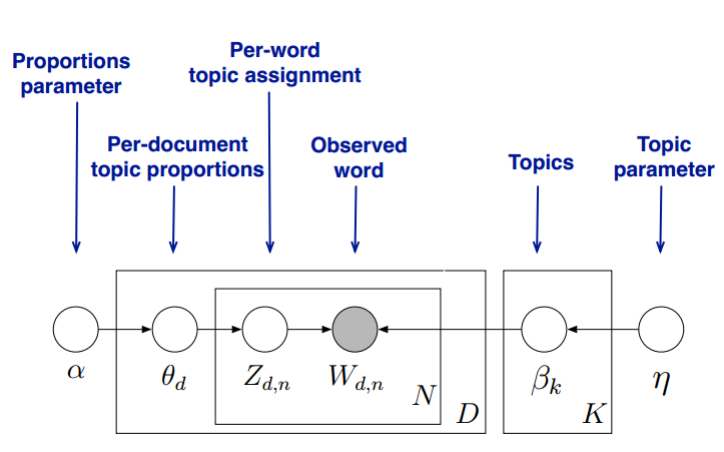
eigenvector centrality分析图如下:



5.基于LDA模型的网约车评论分析

展开定义一些字母的含义：文档集合D，主题（topic)集合T ，D中每个文档d看作一个单词序列<w1,w2,...,wn>，wi表示第i个单词，设d有n个单词。

D中涉及的所有不同单词组成一个大集合V，LDA以文档集合D作为输入，希望训练出的两个结果向量，对每个D中的文档d，对应到不同Topic的概θd<pt1,...,ptk>，其中，pti表示d对应T中第i个topic的概率。计算方法是直观的，pti=nti/n，其中nti表示d中对应第i个topic的词的数目，n是d中所有词的总数。对每个T中的topict，生成不同单词的概率φt<pw1,...,pwm>，其中，pwi表示t生成VOC中第i个单词的概率。计算方法同样很直观，pwi=Nwi/N，其中Nwi表示对应到topict的VOC中第i个单词的数目，N表示所有对应到topic的单词总数。LDA的公式：p(w|d)=p(w|t)\*p(t|d)直观的看这个公式，就是以Topic作为中间层，可以通过当前的θd和φt给出了文档d中出现单词w的概率。其中p(t|d)利用θd计算得到，p(w|t)利用φt计算得到。实际上，利用当前的θd和φt，我们可以为一个文档中的一个单词计算它对应任意一个Topic时的p(w|d)，然后根据这些结果来更新这个词应该对应的topic。然后，如果这个更新改变了这个单词所对应的Topic，就会反过来影响θd和φt。



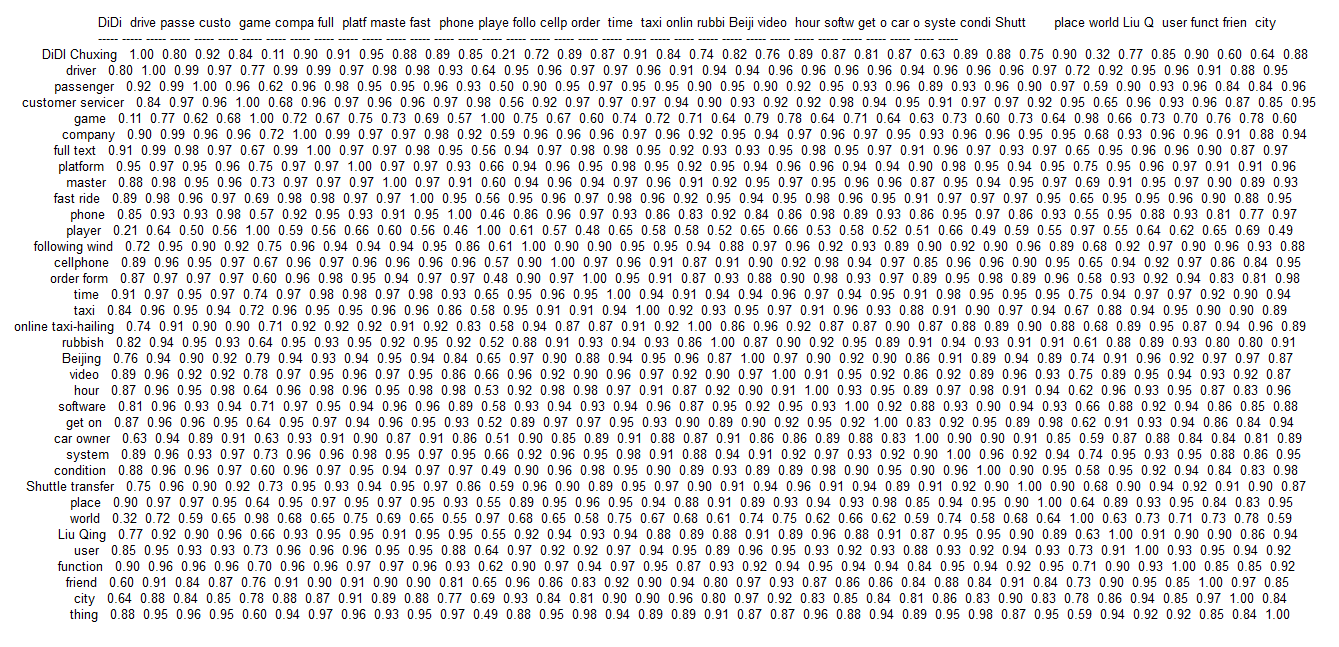
将文本数据导入模型，得到的topic结论为k=3结论最佳

|  |  |  |
| --- | --- | --- |
| Topic1 | Topic2 | Topic3 |
| DiDi Chuxing | driver | Player |
| platform | Software | game |
| Passenger | Car owner | place |
| Customer servicer | Master | friend |
| Order form | Shuttle transfer |  |
| phone | Taxi |  |
| rubbish | Liu Qing |  |
| cellphone | Video |  |
| time | Function |  |
| Company | Following wind |  |
| user | Online taxi-hailing |  |
| Fast ride | Beijing |  |
| Condition |  |  |
| Full text |  |  |
| World |  |  |
| City |  |  |
| Get on |  |  |
| System |  |  |
| Hour |  |  |

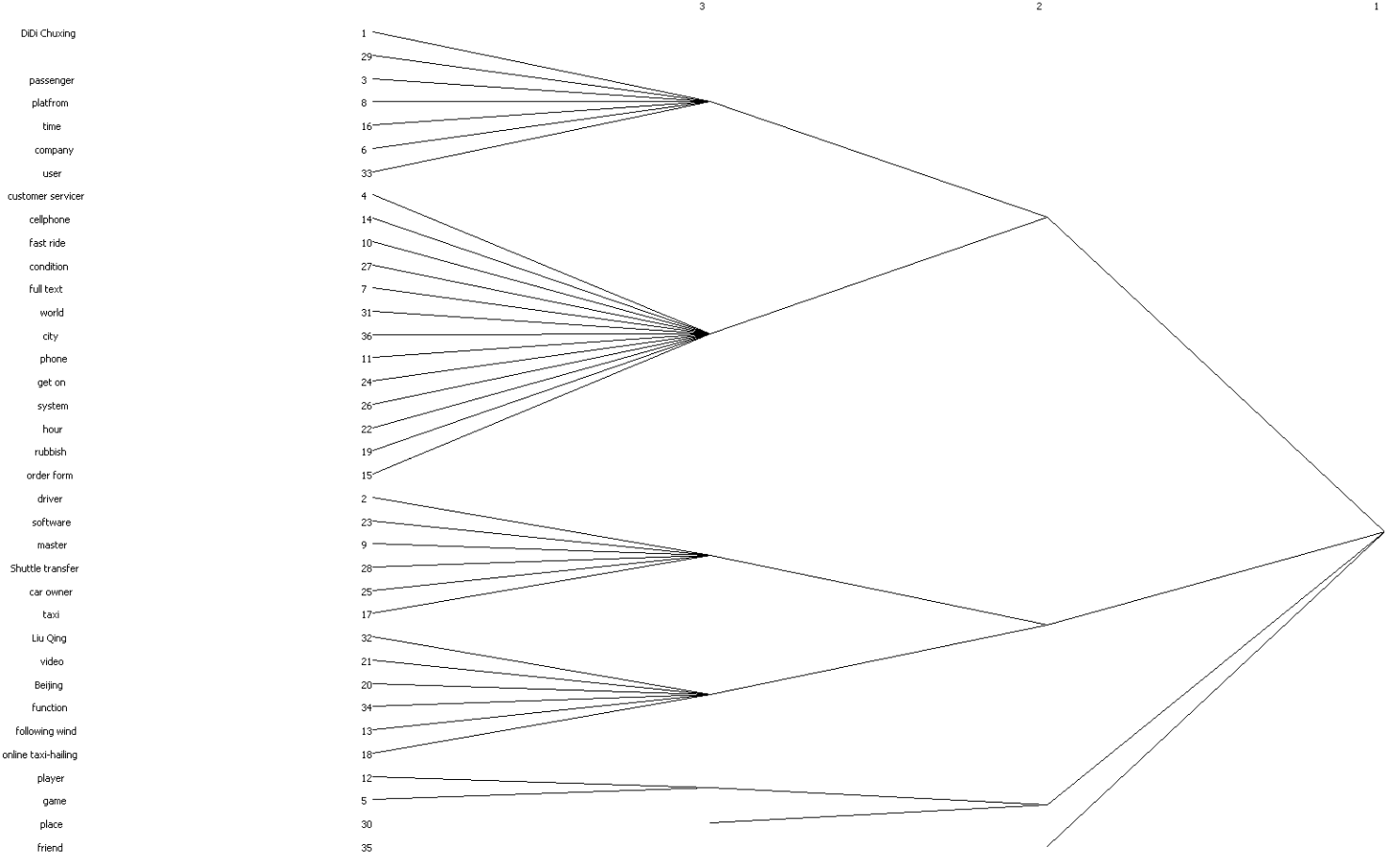
6.基于凝聚子群的滴滴网约车客户评论关键词CONCOR分析

CONCOR分析聚类结果使用Ucinet中的CONCOR计算可以对待分析的关键词进行聚类计算，同时计算出各大类的密度矩阵, CONCOR是一种迭代相关收敛法。它基于如下事实：如果对一个矩阵中的各个行之间的相关系数进行重复计算，最终产生的将是一个仅由1和-1组成的相关系数矩阵。进一步说我们可以据此把将要计算的一些项目分为两类：相关系数分别为1和-1的两类

形成的相关系数矩阵如下：



现将从数据中提取的共现矩阵导入Ucient6软件进行CONCOR分析，形成聚类图表如下：



7.研究结论

滴滴出行自2015年推出，随着中国科技的发展和智能技术的运用，网约车出现在大众视野并得以推广，目前滴滴出行是中国大陆现在最大的网约车线上平台，而消费者对它的出现评论褒贬不一，因此对此爬取评论数据并做了一些研究。

一、对消费者整体文本评论进行关键词提取，选择其中top36,

DiDi-Chuxing, driver, passenger, customer-servicer, game, company, full-text, platform, master, fast-ride, phone, player, following-wind, cellphone, order-form, time, taxi, online-taxi-hailing, rubbish, Beijing,video,hour, software,get-on, car-owner, system, condition, Shuttle-transfer, place, world, Liu Qing,user,function, friend,city,thing

通过这些关键词以及出现的频数，可以看出消费者在使用滴滴出行时，对司机、客服、公司、平台评论较多，2500份左右的数据中频数都大于400，说明用户在使用滴滴出行时关心的因素是服务态度，以及司机配备的安全性和和公司、平台的信任与售后。

二、对文本评论进行LDA主题模型处理

在提取文本数据关键词后，继续进行主题模型分析，经过多次训练的数据topic=3，Topic1为滴滴出行平台，Topic为2为滴滴出行系统，Topic3为用户使用场景，在两千多份数据中，用户对滴滴出行的三大块进行评论，说明用户对滴滴出行平台、滴滴出行的系统以及使用过程中都有自己的想法，可以从这三个方面去进行反思。

三、对文本评论语义网络分析-中心度分析、CONCOR分析

度中心性只是衡量节点中心性的指标之一,指标中介中心性/中间中心性,以经过某个节点的最短路径数目来刻画节点重要性的指标,特征向量中心性,一个节点的重要性既取决于其邻居节点的数量（即该节点的度），也取决于其邻居节点的重要性。在生成的网络图中,可以看到player和game度较小,而platform,car owner,driver较大,频数较大的之间相关性较强，可以发现用户的大多数焦点在于平台、司机、车主。

对文本评论数据进行CONCOR分析，将与关键词相关的词汇进行聚类，子类为7类，分别为

DiDi-Chuxing, passenger, company, platform,time,user

full-text, master, fast-ride, phone, cellphone, order-form, rubbish, customer-servicer, condition, world, city, get-on,system, hour, thing

driver, software, car-owner, Shuttle-transfer, taxi

Liu Qing, video, Beijing, function, following-wind, online-taxi-hailing

player, game,

friend

place

#tf-idf LDA主题模型代码

import math

import jieba

import jieba.posseg as psg

from gensim import corpora,models

from jieba import analyse

import functools

#停用词表加载

def get\_stopword\_list():

stop\_word\_path = './stopword.txt'#停用词所在路径

stopword\_list = [sw.replace('\n','')for sw in open(stop\_word\_path,encoding='utf8').readlines()]

return stopword\_list

#定义分词方法

#pos为是否只保留名词的依据

def seg\_to\_list(sentence,pos=False):

if not pos:

seg\_list = jieba.cut(sentence)#不进行词性标注的分析

else:

seg\_list = psg.cut(sentence)#进行词性标注的分析

return seg\_list

#定义干扰词过滤方法

def word\_filter(seg\_list,pos=True):

stopword\_list = get\_stopword\_list() #获取停用词

filter\_list = [] #存储过滤后的词

#根据pos确定是否执行此行过滤

#不进行过滤则都标记为m并保留

for seg in seg\_list:

if not pos:

word = seg

flag = 'n'

else:

word = seg.word

flag = seg.flag

if not flag.startswith('n'):

continue

#过滤停用词中词以及长度<2的

if not word in stopword\_list and len(word)>1:

filter\_list.append(word)

return filter\_list

#加载数据集

def load\_data(pos=False,corpus\_path='./corpus.txt'):

doc\_list = []

for line in open(corpus\_path,'r',encoding='gbk'):

content=line.strip()

seg\_list = seg\_to\_list(content,pos)#对句子进行分词

filter\_list = word\_filter(seg\_list,pos)#对分词结果过滤干扰词

doc\_list.append(filter\_list)

return doc\_list

#topk关键词

def cmp(e1,e2):

import numpy as np

res = np.sign(e1[1]-e2[1])

if res != 0:

return res

else:

a = e1[0] + e2[0]

b = e2[0] + e1[0]

if a>b:

return 1

elif a==b:

return 0

else:

return -1

#idf统计方法

def train\_idf(doc\_list):

idf\_dic={}

#文档总数

tt\_count = len(doc\_list)

#每个词出现的文档数

for doc in doc\_list:

for word in set(doc):

idf\_dic[word]=idf\_dic.get(word,0.0)+1.0

#按工时转化为idf值，分母加1进行平滑处理

for k,v in idf\_dic.items():

idf\_dic[k] = math.log(tt\_count/(1.0+v))

#对没有出现的次数，默认出现一次

default\_idf = math.log(tt\_count/(1.0))

return idf\_dic,default\_idf

#YF-IDF类

class TfIdf(object):

#参数为：训练好的idf字典，默认的idf值，处理后的待提取样本，关键词数量

def \_\_init\_\_(self,idf\_dic,default\_idf,word\_list,keyword\_num):

self.word\_list=word\_list

self.idf\_dic,self.default\_idf=idf\_dic,default\_idf

self.tf\_dic=self.get\_tf\_dic() #tf数据

self.keyword\_num=keyword\_num

#统计tf值

def get\_tf\_dic(self):

tf\_dic={}

for word in self.word\_list:

tf\_dic[word]=tf\_dic.get(word,0.0)+1.0

tt\_count=len(self.word\_list)

for k,v in tf\_dic.items():

tf\_dic[k]=float(v)/tt\_count

return tf\_dic

#计算tf-idf值

def get\_tfidf(self):

tfidf\_dic={}

for word in self.word\_list:

idf=self.idf\_dic.get(word,self.default\_idf)

tf=self.tf\_dic.get(word,0)

tfidf=tf\*idf

tfidf\_dic[word]=tfidf

#根据tf-idf的排序，keyword num个作为关键词

for k,v in sorted(tfidf\_dic.items(),key=functools.cmp\_to\_key(cmp),reverse=True)[:self.keyword\_num]:

print(k+'/',end='')

print()

#主题模型的类

class TopicModel(object):

#参数:处理后的数据集，关键词数量，具体模型（LSI,LDA），主题数量

def \_\_init\_\_(self,doc\_list,keyword\_num,model='LSI',num\_topics=4):

#停用gensim的接口将文本转化为向量表示

#构建词空间

self.dictionary=corpora.Dictionary(doc\_list)

#使用BOW模型向量化

corpus = [self.dictionary.doc2bow(doc) for doc in doc\_list]

#每个词根据tf-idf提取

self.tfidf\_model=models.TfidfModel(corpus)

self.corpus\_tfidf=self.tfidf\_model[corpus]

self.keyword\_num=keyword\_num

self.num\_topics=num\_topics

#选择加载的模型

if model=='LSI':

self.model=self.train\_lsi()

else:

self.model=self.train\_lda()

word\_dic = self.word\_dictionary(doc\_list)

self.wordtopic\_dic = self.get\_wordtopic(word\_dic)

#训练LSI

def train\_lsi(self):

lsi = models.LsiModel(self.corpus\_tfidf,id2word=self.dictionary,num\_topics=self.num\_topics)

return lsi

#训练lda

def train\_lda(self):

lda = models.LdaModel(self.corpus\_tfidf,id2word=self.dictionary,num\_topics=self.num\_topics)

return lda

#得到数据集的主题一词分布

word\_dic = self.word\_dictionary(doc\_list)

self.wordtopic\_dic=self.get\_wordtopic(word\_dic)

def get\_wordtopic(self,word\_dic):

word\_dic = {}

for word in word\_dic:

single\_list = [word]

wordcorpus=self.tfidf\_model[self.dictionary,doc2bow(single\_list)]

wordtopic=self.model[wordcorpus]

wordtopic\_dic[word] = wordtopic

return wordtopic\_dic

#计算词的分态与文本的相似度，取最高的几个作为关键词

def get\_simword(self,word\_list):

sentcorpus = self.tfidf\_model[self.dictionary.doc2bow(word\_list)]

senttopic = self.model[sentcorpus]

#余弦相似度计算

def calsim(l1,l2):

a,b,c =0.0,0.0,0.0

for t1,t2 in zip(l1,l2):

x1 = t1[1]

x2 = t2[1]

a += x1\*x1

b += x1\*x1

c += x2\*x2

sim = a/math.sqrt(b\*c) if not (b\*c) == 0.0 else 0.0

#计算输入文本和每个词的主题分布相似度

sim\_dic = {}

for k,v in self.wordtopic\_dic.items():

if k not in word\_list:

continue

sim = calsim(v,senttopic)

sim\_dic[k] = sim

for k,v in sorted(sim\_dic.items(),key=functools.cmp\_to\_key(cmp),reverse=True)[:self.keyword\_num]:

print(k+'/',end='')

print()

#词空间构建方法和向量化方法，在没有gensim接口时的一般处理放法

def word\_dictionary(self,doc\_list):

dictionary = {}

for doc in doc\_list:

dictionary.extend(doc)

dictionary = list(set(dictionary))

return dictionary

def doc2bowvec(self,word\_list):

vec\_list = [1 if word in word\_list else 0 for word in self.dictionary]

return vec\_list

if \_\_name\_\_ == '\_\_main\_\_':

corpus\_path='E:\Desktop\semantic analysis\didicomment1.txt'

pos = True

for line in open(corpus\_path,'r',encoding='gbk'):

content=line.strip()

seg\_list = seg\_to\_list(content,pos)#对句子进行分词

filter\_list = word\_filter(seg\_list,pos)

print('tf-idf的结果：')

#alist = [['行程'],['吴中区'],['权限'],['客服'],['前提'],['事实'],['苏州'],['费用'],['小时'],['司机'],['滴滴']]

doc\_list = load\_data(pos)

idf\_dic,default\_idf = train\_idf(doc\_list)

tfidf\_model = TfIdf(idf\_dic,default\_idf,filter\_list,keyword\_num=11)

print(tfidf\_model.get\_tfidf())

print(tfidf\_model.get\_tf\_dic())

idic = {}

adic = {'滴滴': 0.01710739783964967, '行程': 0.7356973209733757, '前提': 0.3789865755429755, '客服': 0.3931205445071967, '司机': 0.07126008072381139, '事实': 0.3470599740576011, '费用': 0.23719385421057773, '权限': 0.4223082743279721, '小时': 0.20069711117339314, '苏州': 0.3260304592687753, '吴中区': 0.44764984358473237}

for k,v in adic.items():

idic[k] = int(v\*count)

print(idic)

print('lda的结果：')

topic\_model = TopicModel(doc\_list,keyword\_num=10,model='LDA')

word\_dic = topic\_model.word\_dictionary(doc\_list)

wordtopic\_dic = topic\_model.get\_wordtopic(word\_dic)

print(wordtopic\_dic)

get\_list,get\_topic = topic\_model.get\_simword(filter\_list)

print(get\_list,get\_topic)

a = topic\_model.doc2bowvec(filter\_list)