

技术文档-研究热点的抽取和演变

一、 课题分析

- a) 关键词的抽取和识别:
 - i. 关键词的识别: 根据文本'关键词'对应文档中的每一行, 将对应的一行文本提取。
 - ii. 关键词的提取: Python 的 docx 库抽取每年的关键词, 并存储在文本文档中。
- b) 热点聚类, 一种是按照年度做聚类, 一种是不分年度做聚类; 聚类结果存储在文件中:
 - i. 按照年度的聚类: 使用机器学习库 scikit-learn 中的 BIRCH 方法进行聚类, 遍历文档后将结果存储。
 - ii. 不按照年度的聚类: 方法相同, 在之前需要将所有年度的关键词文档合并为一个总的文档, 聚类结果也会存在一个新的文本的文档中。
- c) 研究按照年度的热点演变轨迹分析:
 - i. 先将各个年度的关键词聚类结果输入到函数中, 选取年度的热点聚类 (也就是频率出现最多的类), 将其保存在列表里。
 - ii. 横坐标为 2005-2018 年度, 纵坐标为热点聚类出现的次数, 调用 matplotlib 库绘制折线图。

二、 运行环境

Windows10+python3.6.4

三、 实现步骤

- a) 提取关键词:
实现代码:

```
import docx

a = '关键词'
#文件路径
doc5 = docx.Document('E:\\spark_dataset\\2005.docx')
#2005
f5 = []
for i in range(len(doc5.paragraphs)):
    if a in doc5.paragraphs[i].text:
        key_C = doc5.paragraphs[i].text[4:]
        f5.append(key_C)
t1 = open('E:\\a\\2005.txt', 'w')
for item in f5:
    t1.write(item+'\n')
t1.close()
```

导入 python docx 库, 提取原文件: 位于 E 盘下 spark_dataset 文件夹下的.docx 文件, 设置变量 a 的值为“关键词”; 新建一个空的列表 f, 用于存储加载的符合标准的数据, 使用 for 循环和 docx 库中的 paragraphs 方法遍历文档中的每一行, 如果这行数据的文本中含有变量 a (“关键词”), 那么将其提取出来, text[4:]的作用是只取“关键词: ”后的

文本，用 append 方法加在列表 f 中；此时列表 f 中的每个元素为一行的关键词；最后将数据写入到新的文本文件中，使用 open 方法打开或新建文本文件，设置参数为 w（写入），使用 for 循环遍历列表 f 中的每一个元素，将其 write 进打开的文本文件中，并且在每一个元素写入后设置\n 换行，最后将文档关闭，完成关键词的提取。

2005.txt	2018/7/8 1:15	文本文档	3 KB
2006.txt	2018/7/8 1:15	文本文档	3 KB
2007.txt	2018/7/8 1:15	文本文档	3 KB
2008.txt	2018/7/8 1:15	文本文档	3 KB
2009.txt	2018/7/8 2:14	文本文档	4 KB
2010.txt	2018/7/8 1:27	文本文档	4 KB
2011.txt	2018/7/8 1:15	文本文档	3 KB
2012.txt	2018/7/8 1:27	文本文档	3 KB
2013.txt	2018/7/8 1:27	文本文档	4 KB
2014.txt	2018/7/8 1:27	文本文档	3 KB
2015.txt	2018/7/8 1:27	文本文档	3 KB
2016.txt	2018/7/8 1:27	文本文档	3 KB
2017.txt	2018/7/8 2:15	文本文档	3 KB
2018.txt	2018/7/8 1:19	文本文档	2 KB
all_year.txt	2018/7/8 2:15	文本文档	78 KB

all_year.txt 是我将所有年份的关键词提取成功后，在 windows 系统下使用 cmd 命令行模式将所有的 txt 文件合并为一个新文件，其中的数据是所有年份文档的关键词。

```
E:\a>copy *.txt all_year.txt
2005.txt
2006.txt
2007.txt
2008.txt
2009.txt
2010.txt
2011.txt
2012.txt
2013.txt
2014.txt
2015.txt
2016.txt
2017.txt
2018.txt
已复制          1 个文件。
E:\a>
```

b) 文本向量化
实现代码：

```

#encode = utf-8
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

# 读取数据, 进行TF-IDF计算
file1 = ['E:\\a\\2005.txt', 'E:\\a\\2006.txt', 'E:\\a\\2007.txt', 'E:\\a\\2008.txt', 'E:\\a\\2009.txt',
'E:\\a\\2010.txt', 'E:\\a\\2011.txt', 'E:\\a\\2012.txt', 'E:\\a\\2013.txt', 'E:\\a\\2014.txt',
'E:\\a\\2015.txt', 'E:\\a\\2016.txt', 'E:\\a\\2017.txt', 'E:\\a\\2018.txt']
# 存取数据的分词结果
file2 = ['E:\\b\\2005.txt', 'E:\\b\\2006.txt', 'E:\\b\\2007.txt', 'E:\\b\\2008.txt', 'E:\\b\\2009.txt',
'E:\\b\\2010.txt', 'E:\\b\\2011.txt', 'E:\\b\\2012.txt', 'E:\\b\\2013.txt', 'E:\\b\\2014.txt',
'E:\\b\\2015.txt', 'E:\\b\\2016.txt', 'E:\\b\\2017.txt', 'E:\\b\\2018.txt']

def idf1(i):
    corpus = []
    f = open(file1[i], 'r+')
    content = f.read()
    f.close()
    corpus.append(content)
    #将得到的词语转换为词频矩阵
    vectorizer = CountVectorizer()
    #统计每个单词的tf-idf权值
    transformer = TfidfTransformer()
    #计算出tf-idf并转换为tf-idf矩阵
    tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))
    # 所有文本的关键字
    word = vectorizer.get_feature_names()
    # 对应的tfidf矩阵
    weight = tfidf.toarray()
    return word, weight

```

在这里我使用了 python 的 scikit-learn 包进行 TF-IDF 分词权重计算, 主要使用了两个类: TfidfTransformer 和 CountVectorizer, 其中 CountVectorizer 是通过 fit_transform 函数将文本中的词语转换为词频矩阵, 通过 get_feature_names()可看到所有文本的关键字, 通过 toarray()可看到词频矩阵的结果; TfidfTransformer 是统计 vectorizer 中每个词语的 tf-idf 权值。

先新建一个列表, 用于存储读取到的关键词数据, 一定要写在最前面, 否则在运行时会出现 corpus 未清零的问题, 影响结果并使运行时间过长; 在这之前我将所有文件的路径写在了列表中, for 循环在最后的 main 函数 (见下图:); 先将所有读取的数据写入到列表 corpus 中, vectorizer = CountVectorizer()使用 CountVectorizer 函数将得到的关键词转换为词频矩阵, transformer = TfidfTransformer()计算出每个单词的 TF-IDF 权值, tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus)) 将刚刚得到的 transformer 转换为 TF-IDF 矩阵, word = vectorizer.get_feature_names()获取所有文本中的关键词, weight = tfidf.toarray(), 获取所有文本中的关键词对应的 TF-IDF 矩阵。这个函数的返回值为 word (文本中的所有关键词) 和 weight (所有关键词对应的 TF-IDF 权值)。

```

if __name__ == '__main__':
    for i in range(len(file1)):
        a = []
        if i != 4:
            a,b = idf1(i)
            for j in range(len(file2)):
                if j != 4:
                    c = save(a,b)
        else:
            a, b = idf2(i)
            for j in range(len(file2)):
                if j == 4:
                    c = save(a, b)

    print('第' + str(i+1) + '个文件已保存')

```

由于我的数据文档中有一个文档存在编码错误，在进行断点操作分析错误之后，我找到了这个编码错误的文档，该文档需要在打开时加入一个参数：encoding='ANSI'（如下图），这也是我在上图的循环中加入了 if 判断语句的原因，当遍历到第 4 个文档时，需要用另一个函数 idf2（该函数与 idf1 的区别仅仅在于有无参数 encoding='ANSI'，之所以不将所有的文档读取都设置这个参数，是因为其他文档设置该参数后会出现'ANSI'编码错误），这两个函数的代码见下图：

```

def idf1(i):
    corpus = []
    f = open(file1[i], 'r+')
    content = f.read()
    f.close()
    corpus.append(content)
    #将得到的词语转换为词频矩阵
    vectorizer = CountVectorizer()
    #统计每个单词的tf-idf权值
    transformer = TfidfTransformer()
    #计算出tf-idf并转换为tf-idf矩阵
    tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))
    # 所有文本的关键字
    word = vectorizer.get_feature_names()
    # 对应的tfidf矩阵
    weight = tfidf.toarray()
    return word,weight

def idf2(i):
    corpus = []
    f = open(file1[i], 'r+',encoding='ANSI')
    content = f.read()
    f.close()
    corpus.append(content)
    #将得到的词语转换为词频矩阵
    vectorizer = CountVectorizer()
    #统计每个单词的tf-idf权值
    transformer = TfidfTransformer()
    #计算出tf-idf并转换为tf-idf矩阵
    tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))
    # 所有文本的关键字
    word = vectorizer.get_feature_names()
    # 对应的tfidf矩阵
    weight = tfidf.toarray()
    return word,weight

```


c) 关键词聚类

- i. 按照年度进行聚类并存储聚类结果至新文本文档:

实现代码:

```
from sklearn.cluster import Birch

#设置3个类并分类
def julei(word,weight):
    clusterer = Birch(n_clusters=3)
    y = clusterer.fit_predict(weight)
    # print(y)
    print(y.shape)

    for i in range(14):
        f2 = open(file3[i], 'w+')
        for j in range(len(y)):
            f2.write(word[j] + " " + str(y[j]) + "\n")
            # print(word[j] + " " + str(y[j]))
        f2.close()
    return y
```

使用 python scikit-learn 机器学习库中的 BIRCH 聚类算法对关键词进行聚类, 该函数的参数为 TF-IDF 计算出的 word 和 weight, 使用 Birch 方法设置参数 n_clusters 为 3, 该参数为类别数; 将其 fit 后得到 y, y 为分类结果 (如下图:); 后打开新的文本文档, 将每一个聚类结果与对应的关键词匹配, 输出到新的文本文档中。该函数的返回值为 y。

这是2013年的聚类结果:

```
[0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
 0 0 2 0 0 0 0 1 0 0 0 2 0]
(87,)
```

- ii. 不按照年度进行聚类分析并存储结果至新文本文档:

实现代码:

```

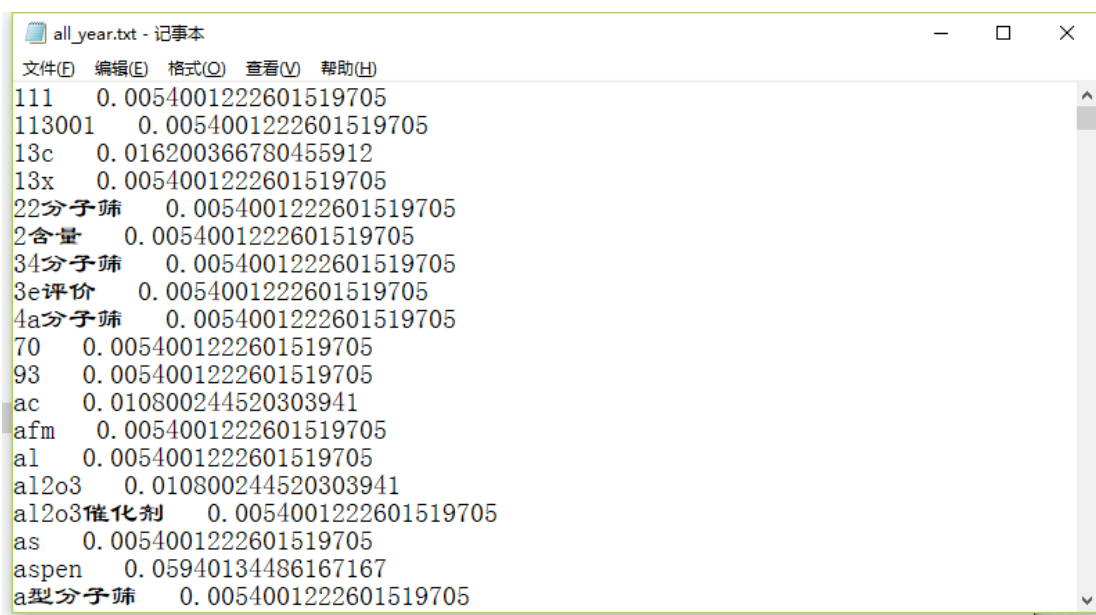
from sklearn.cluster import Birch
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

#新建列表
corpus = []
#读取文件
f = open('E:\\a\\all_year.txt', 'r+', encoding='ANSI')
#按行读取
lines = f.readlines()
for i in range(len(lines)):
    # print(lines[i])
    content = lines[i]
    #按行加入列表
    corpus.append(content)
f.close()
vectorizer = CountVectorizer()
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))
word = vectorizer.get_feature_names()
weight = tfidf.toarray()
# print(weight.shape)

#设置3个类
clusterer = Birch(n_clusters=3)
y = clusterer.fit_predict(weight)
# print(y)
# print(y.shape)

```

同样的，使用 python scikit-learn 机器学习库中的 BIRCH 聚类算法对关键词进行聚类，该函数的参数为 TF-IDF 计算出的 word 和 weight，将得到的 TF-IDF 结果存储在文件中（如下图：）



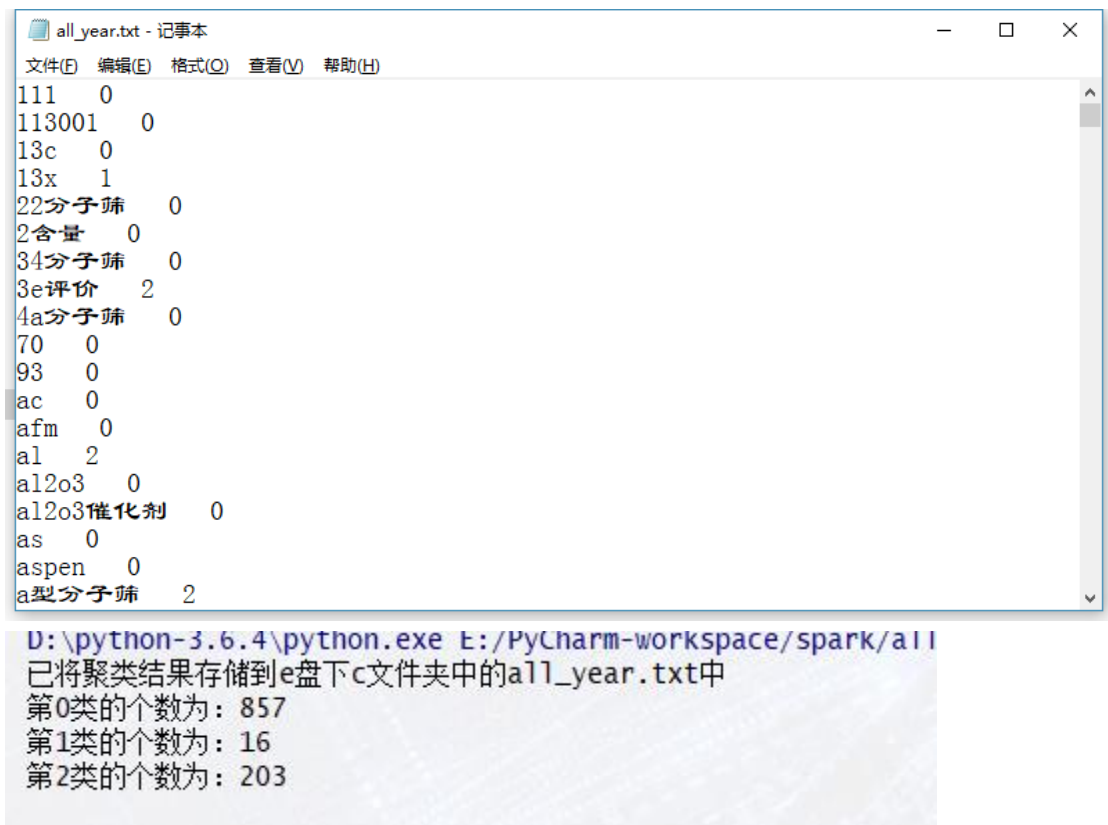
```

all_year.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
111 0.0054001222601519705
113001 0.0054001222601519705
13c 0.016200366780455912
13x 0.0054001222601519705
22分子筛 0.0054001222601519705
2含量 0.0054001222601519705
34分子筛 0.0054001222601519705
3e评价 0.0054001222601519705
4a分子筛 0.0054001222601519705
70 0.0054001222601519705
93 0.0054001222601519705
ac 0.010800244520303941
afm 0.0054001222601519705
al 0.0054001222601519705
al2o3 0.010800244520303941
al2o3催化剂 0.0054001222601519705
as 0.0054001222601519705
aspen 0.05940134486167167
a型分子筛 0.0054001222601519705

```

使用 Birch 方法设置参数 n_clusters 为 3，该参数为类别数；将其 fit 后得到 y，y 为聚类结果（如下图：）；后打开新的文本文档，将每一个聚类结果与对应的关键词匹配，输出到新的文本文档中（E 盘下的 c 文件夹中的 all_year.txt 中）。

打开该文档，可以看到各个关键词被分到不同的类中（如下图：）



The image shows a Notepad window titled 'all_year.txt - 记事本' with the following text:

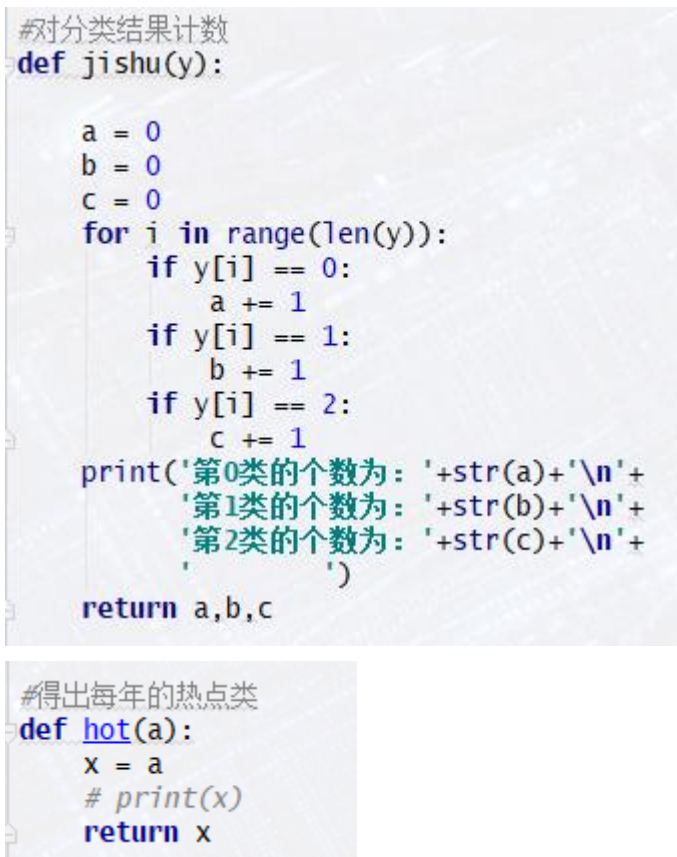
```
111 0
113001 0
13c 0
13x 1
22分子筛 0
2含量 0
34分子筛 0
3e评价 2
4a分子筛 0
70 0
93 0
ac 0
afm 0
al 2
al2o3 0
al2o3催化剂 0
as 0
aspen 0
a型分子筛 2
```

Below the Notepad window is a terminal window showing the execution of a Python script:

```
D:\python-3.6.4\python.exe E:/PyCharm-workspace/spark/all
已将聚类结果存储到e盘下c文件夹中的all_year.txt中
第0类的个数为：857
第1类的个数为：16
第2类的个数为：203
```

d) 聚类结果分析

实现代码：



```
#对分类结果计数
def jishu(y):
    a = 0
    b = 0
    c = 0
    for i in range(len(y)):
        if y[i] == 0:
            a += 1
        if y[i] == 1:
            b += 1
        if y[i] == 2:
            c += 1
    print('第0类的个数为: '+str(a)+'\n'+
          '第1类的个数为: '+str(b)+'\n'+
          '第2类的个数为: '+str(c)+'\n'+
          ')')
    return a,b,c

#得出每年的热点类
def hot(a):
    x = a
    # print(x)
    return x
```

如代码所示，对按照年度的聚类结果进行分析，需要先对每次执行的聚类操作结果进行计数，因为我的聚类结果分为三类，所以对每一类都进行计数操作，新建变量 a, b, c 都为 0，对 y 遍历，遇到 a, b, c 对应的类，会加 1，最后得到 a, b, c 三个数，在 main 函数中做循环，得到每年的类别计数；再将每年最热的类别出现的频率存储在 x 中，为后面制作热点轨迹图打下基础。

e) 热点轨迹图绘制

实现代码：

```
Longitudinal = []
Transverse = [2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018]
for i in range(len(file1)):
    print('这是'+str(i+2005)+'年的聚类结果：')
    if i != 4:
        aaa = []
        aaa = read1(i)
        bbb,ccc = deal(aaa)
        ddd = juei(bbb,ccc)
        eee = jishu(ddd)
        Transverse.append(hot(eee[0]))
    else:
        aaa = read2(i)
        bbb,ccc = deal(aaa)
        ddd = juei(bbb,ccc)
        eee = jishu(ddd)
        Transverse.append(hot(eee[0]))
# print(Transverse)
# print(Longitudinal)
```

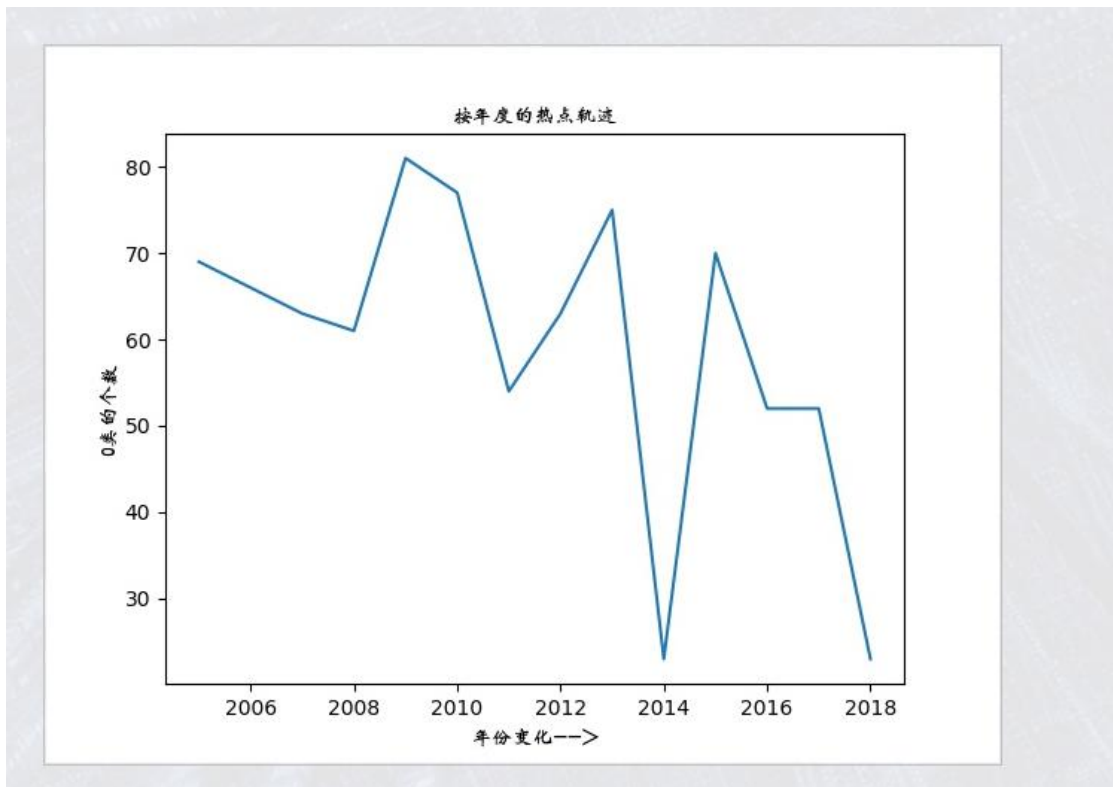
新建两个列表 Transverse 和 Longitudinal, Transverse 为绘图时的横坐标, 数据为年份, Longitudinal 为绘图时的纵坐标, 数据为每年热点聚类出现的次数, Transverse 的获取方式为调用之前计算每年的热点类的参数 a, 将其写在 Transverse 的列表中

```
plt.figure(1, dpi=100)
plt.plot(Transverse,Longitudinal)
plt.title(u'按年度的热点轨迹',fontproperties=myfont)
plt.xlabel('年份变化-->',fontproperties=myfont)
plt.ylabel('0类的个数',fontproperties=myfont)
plt.savefig('hot.jpg')
# plt.show()
```

以 Transverse 和 Longitudinal 为两个坐标绘图, 展示 2005-2018 年的最热类别折线图, 在设置名称时, 发现绘图后会有中文乱码, 查阅资料后增加参数 fontproperties=myfont, myfont 为 windows 系统的字体, 需要导入 (如下图:), 并将生成后的图片保存在项目文件夹下。

```
myfont = matplotlib.font_manager.FontProperties(fname=r'C:\\Windows\\Fonts\\汉仪魏碑简.ttf')
```

完成所有后绘制出的折线图为：



四、结果分析

至此，所有步骤已经完成，根据热点轨迹图我们可以看出，2005 年至 2013 年，聚类所得的第 0 类普遍被大多数人研究，在 2014 年却只有 20 多人的论文方向是根据这一类，后几年又恢复正常。

五、总结归纳

聚类时我用了 BIRCH 聚类的方法，还有其它很多聚类方法如：K-Means、Gaussian mixtures、Mean-shift 等方法，因为去年做过 K-Means 方法，今年想有不同的尝试，所以我选择了 BIRCH 方法，BIRCH 方法通过计算点之间的欧式距离来进行聚类。但 BIRCH 方法不能很好的扩展到高维数据，不过处理这次的问题还是足够了。