智慧政务中的文本挖掘

2020 年 5 月 5 日

摘要

近年来，随着微信、微博、市长信箱、阳光热线等网络问政平台逐步成为政府了解民意、汇聚民智、凝聚民气的重要渠道，各类社情民意相关的文本数据量不断攀升，给以往主要依靠人工来进行留言划分和热点整理的相关部门的工作带来了极大挑战。另一方面，每时每刻都会有人来留言投稿，有的长篇大论，这就对人工带来了巨大的工作量，因此，人工处理群众的留言有着工作量大，效率低，且差错率大等弊端。

同时，随着大数据、云计算、人工智能等技术的发展，建立基于自然语言处理技术的智慧政务系统已经是社会治理创新发展的新趋势，对提升政府的管理水平和施政效率具有极大的推动作用。而且，大数据的运用能够对随时随地的市民反馈甚至是紧急情况进行及时的应对措施，这对处理事件的速度有了极大的提升。

我们的论文主要针对市民的留言详情对其建立一级标签分类的模型，在此模型中我们主要用到了 RNN 循环神经网络中词嵌入的方法，和 Keras 库，在定义损失函数时用了分类交叉熵的方法。

因为热点问题挖掘需要收集在一段时间对某一地区的问题的反映，所以我们采用命名实体识别的方法将我们所需要的地点名词区分出来，我们再用 sort 函数将词汇出现的次数进行排序，以便我们可以进行热点问题的挖掘和处理分析。

关于政府的相关答复，我们从文本相似性、文本完整性和文本可解释性等多方面进行分析。

关键词：自然语言处理，RNN 循环神经网络，keras 模型，命名实体识别

目录

一 问题重述

1.1 问题背景

随着科技的发展，网络平台的运用日益增长，市民或顾客反应心声不再像以前一样需要亲自将匿名信投递到真实的信箱或具体部门那样麻烦低效，网络发达也使民情民意更方便快捷地汇集到政府部门手里。

然而相应的问题也随之而来，留言数量不断攀升，给以往主要依靠人工的留言划分和热点整理的相关部门带来了极大的挑战。与此同时，随着大数据、云计算、人工智能等领域在近年来的崛起，相应的基于自然语言文本处理技术的智慧政务系统也将会使社会治理发展的新趋势，运用计算机的文本分类、机器翻译、量化处理，省时省力，以此更大程度上提高政府管理水平和施政效率。

1.2 问题提出（问题重述）

如果相关市政部门要高效处理市民留言，并有效解决市民提出的问题，需在分流留言信息、筛选热点问题以及提高答复意见质量等各个环节进行完善和提高效率。

因此，文本根据以下问题及相关资料，建立模型，设计不同的算法分别解决分类留言和筛选热点问题时对人力的节省，以及对答复意见的评价方案，并完善算法，提高准确率，为提升政务办事效率提供参考。

问题一：

在处理平台的群众留言时，工作人员首先按照一定体系将留言分类，以便后续能分派给不同的相应职能部门。目前，大部分电子信息仍是依靠人工处理，耗时耗力，且人工处理易出差错，为提高分流留言的效率，需建立关于留言内容的一级标签分类模型，编写合理的算法，让计算机来实现繁琐又多次的任务。

通常用$F\text{-}Score$对分类方法进行评价：

$$F_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2P_iR_i}{P_i+R_i},$$

其中P_i为第 i 类的查准率，R_i为第 i 类的查全率。

问题二：

在群众反映的诸多问题中，并不会是所有问题都是十分迫切、被关注度很高的，某一时段内群众集中反映的某一问题可称为热点问题。若能及时发现热点问题，优先有针对性地对热点问题进行处理，则能帮助相关部门提高工作效率。因此，我们需将某一时段内反映特定地点或特定人群问题的留言进行归类，定义合理的热度评价指标，并给出评价结果，罗列出热点问题的具体内容，以便相关职能部门快速了解到市民留言的热点，并针对解决。

问题三：

对于市民留言的答复意见通常是由人工完成的，但由于人工的劳累或不同的人完成，会出现质量不统一，因此需结合答复的相关性、完整性、可解释性等各角度，编写相应算法给答复意见的质量提供一套评价方案，并实现，从而有针对性地完善答复意见，以提高答复质量，更有效地服务市民。

二、算法与求解

2.1、群众留言分类

2.1.1、问题分析

群众的留言分类问题是一项针对中文文本的自然语言挖掘与处理的问题。在中文文本中，不同的两个字的组合就有可能会形成不用意义的词语，因此我们需要对中文文本进行分词处理。在一段中文文本中，会出现很多类似于“一会儿”，“我想”，“可能”等在语言表达中没有实际意义的词语，同时还有许多的标点符号，空格和换行符，因此我们需要对文本进行数据清洗的处理。一条文本的特征是否能转换为可视化的结果而被计算机所能接受和处理，因此我们需要对文本进行数字化特征的处理。而两条文本间的相关性度量是建立在文本数字化的基础上进行的，我们需要用一定的方法计算数字化之后的文本之间的相似度。最后，将整个过程训练为一个模型，并进行模型验证，调优。由此，模型得以建立及实现。

2.1.2、问题解决方案

2.1.2.1　方法探索

首先，我们拿到一条中文文本：“A3 区大道西行便道，未管所路口至加油站路段，人行道包括路灯杆，被圈西湖建筑集团燕子山安置房项目施工围墙内。每天尤其上下班期间这条路上人流车流极多，安全隐患非常大。强烈请求文明城市 A 市，尽快整改这个极不文明的路段。”我们要考虑怎样才能生成像我们人为读文章一样的断句与连读，怎样能够将我们拿到的文本合理的分成一个一个的词语。

于是，这里我们用到 jieba 中文分词库将文本分割成词语，我们用 jieba.lcut(“文本”)的方法对一段文字进行精确模式的分词。

```
text=jieba.lcut("A3 区大道西行便道，未管所路口至加油站路段，人行道包括路
```

灯杆，被圈西湖建筑集团燕子山安置房项目施工围墙内。每天尤其上下班期间这条路上人流车流极多，安全隐患非常大。强烈请求文明城市 A 市，尽快整改这个极不文明的路段。")

分词结果如下。我们可以看到像“燕子山”，“安置房”，“极不文明”等本应该是组合词的词语却被精确分词模式下的 lcut 切了开来，那么此时如果我们想得到组合词而不想把它们拆开来当成两个甚至三个词语去分析的话，我们可以用 jieba.add_word(“燕子山”)的方法将该组合词添加入 jieba 分词库，这样我们就能得到我们想要的“燕子山”的组合词了，其他所有词语都可以按照这种方法去添加。

```
print(text)
['A3', '区', '大道', '西行', '便', '道', ',  ', '未管', '所', '路口', '至', '加油站', '路段', ',
', '人行道', '包括', '路灯', '杆', ',  ', '被', '圈', '西湖', '建筑', '集团', '燕子', '山', '安
置', '房', '项目', '施工', '围墙', '内', '。', '每天', '尤其', '上下班', '期间', '这', '条', '
路上', '人流', '车流', '极', '多', ',  ', '安全隐患', '非常', '大', '。', '强烈', '请求', '文
明城市', 'A', '市', ',  ', '尽快', '整改', '这个', '极', '不', '文明', '的', '路段', '。']
```

这时，我们可以看到其中有很多词语，比如“的”，“便”，“每天”是对于我们进行分类判断所没有帮助的词语，因此，下一步我们将对这些没有实际意义的词语进行删除工作。

2.1.2.2　加载停用词

我们将采用去除停用词的方法，将特殊字符一同去除。首先我们用 open 的方法读取停用词(1).txt 文件，用 fr.read()读取文件中的所有内容，用 jieba.lcut 的方法以空格作为分割，将停用词(1)文件中的所有词进行精确模式分词，并用 append 的方法遍历分词后的元素，将其添加到 stop 列表中。

```
fr=open('E:/示例数据/C 题/停用词.txt',encoding='utf-8')
words=fr.read()
result=jieba.lcut(words)
stop=[]
for s in result:
    stop.append(s)
stop.extend([u'（',u'）','(',')','/','-','.','&','\t','\n',' ','　','　','\xa0','\u3000'])
fr.close()
stop=list(set(stop))
```

这时我们可以得到如下表所示的一个包含 1713 个元素的的停用词列表，这里，我们用 print 的方法展示其中的一些，这些词多是一些助词，虚词和副词，正是我们在分析文本时想要去掉的。

```
len(stop)
1713
print(stop)
['很少', '除', '不曾', '单单', '有所', '非得', '人家', '本地', '以免', ' / ', '腾', '存心', '千万', '谁料', '不会', '几番', '光是', '知道', '传闻', '」', '据', '直到', '近来', 'λ', '特点', '到底', '正是', '不怎么', '准备', '先后', '于是', '按说', '时候', '矣', '俺们', '表示', '可', '莫不', '采取', '怎', '绝对', '满', ':', ' B ', '"', '相信', '一方面', '趁早', '昉', '这种', '故意', '要不然', '高低', '那时', '仍旧',……]
```

接下来我们用一个列表来储存我们经过去停用词之后的词语列表。我们先定义了一个空列表 processed，用 for 循环遍历 text 中的每一个元素，并赋给 word，

用 if 语句判断 word 是否在停用词 stop 中，若不在，则用 append 的方法将 word 添加进 processed 列表中，等所有元素都遍历完之后，便生成了一个完整的 processed 列表。

```
processed=[]
for word in text:
    if word not in stop:
        processed.append(word)
print(processed)
```

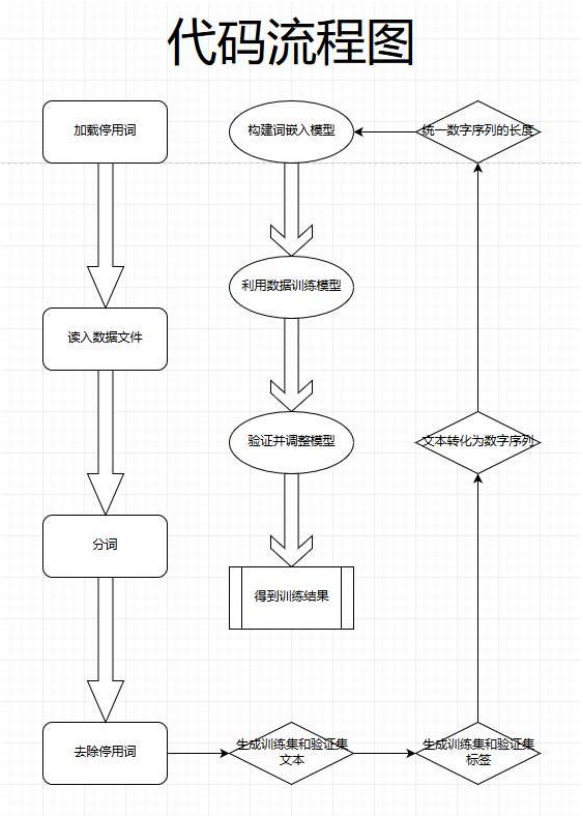
我们可以来看一下经过去除停用词后的 processed 列表的结果。这是一个含有 37 个元素的列表，当然这个列表是非常短的，可能有的会很长，这就需要我们在后面进行处理。

```
print(processed)
['A3', '区', '大道', '西行', '道', '未管', '路口', '加油站', '路段', '人行道', '包括', '路灯', '杆', '圈', '西湖', '建筑', '集团', '燕子', '山', '安置', '房', '项目', '施工', '围墙', '上下班', '期间', '条', '路上', '人流', '车流', '安全隐患', '请求', '文明城市', '市', '整改', '文明', '路段']
len(processed)
37
```

在对自然语言文本处理有了一个清晰而简短的认识后，我们要开始对群众留言进行分类判别的处理。

首先我们拿到的文件是一个 excel 文件，文件中一共有 6 列，第一列是留言编号，，第二列是留言用户，第三列是留言主题，第四列是留言时间，第五列是

留言详情，第六列是一级分类标签。对于群众留言的标签分类，我们所需要考虑的有用信息有留言详情和一级分类标签两列，留言详情和一级分类标签是一一对应的。



2.1.2.3　读取数据文件

在 python 中读取 excel 表格数据的第三方库是 xlrd，我们先将 xlrd 第三方库导入

```
Import xlrd
```

我们知道在字典中，每一个词语和它的解释都是一一对应的，如果是纸质版的字典，我们可以通过先寻找首字母，再寻找第二个字母，一次找寻的方式找到我们想要查找的单词；而在电子版的字典类型文件中，我们可以直接通过调用字典中的关键词 key 值，直接对关键词所对应的 value 值进行检索，这就大大缩短了我们的时间，提高了效率。因此，在这里我们选用字典类型的文件来保存我们

从 excel 表格中读入的数据。

接下来我们对附件 2 表格中的所有列所有行进行遍历循环，附件 2 中的第一行作为关键词，从第二行开始到最后一行的内容作为和关键词所对应的 value 值，建立字典类型的 train 和 test 文件。由此将 excel 文件中的所有信息都导入到 train 和 test 中。在此问题中，为了使我们建立模型的数据库有较多的数据，因此我们将全部数据作为训练集数据，而将示例数据作为测试集数据。于是我们得到了一个 len(train)=9210 的训练集和 len(test)=495 的测试集。

```
def read_train():                                    #定义读取文件函数 read_train
    book = xlrd.open_workbook(r"E:/示例数据/C 题/train/附件 2.xlsx")    #读取 excel 表格
    table = book.sheet_by_name("Sheet1")             #读取 excel 表格中的标签页
    row_Num = table.nrows                            #计算标签页中的有效行
    col_Num = table.ncols                            #计算标签页中的有效列
    u =[]                                            #建立一个空字典
    key =table.row_values(0)                         #表格中第一行为字典关键词
    j = 1                                            #循环初值
    for i in range(row_Num-1):
        v ={}                                        #建立一个空字典
        values = table.row_values(j)                 #读取第 i 行的全部内容
        for x in range(col_Num):
            v[key[x]]=values[x]                      #将关键词和 value 值一一对应
        j+=1
```

u.append(v) return u	#将字典内容储存到 list 中 #返回 list 值

def read_ test ():
book = xlrd.open_workbook(r"E:/示例数据/C 题/ test/附件 2.xlsx")
table = book.sheet_by_name("Sheet1")
row_Num = table.nrows
col_Num = table.ncols
u =[]
key =table.row_values(0)
j = 1
for i in range(row_Num-1):
v ={}
values = table.row_values(j)
for x in range(col_Num):
v[key[x]]=values[x]
j+=1
u.append(v)
return u
test=read_test()	

我们可以取 train 的前两个值来看一下字典文件的情况。xlrd 库很好的将 excel 表格中的所有内容都导入到了 python 环境中，train 是一个 list 类型的数据

文件，而 train 中的每一个元素都是长度为 6 的字典类型文件。此时，我们如果直接检索 train[0][“留言时间”]，则会得到 train 列表中第 0 位置的元素的留言时间所对应的 value 值 '2020/1/6 12:09:38'。

```
train[0:2]
[{'留言编号': 24.0,
  '留言用户': 'A00074011',
  '留言主题': 'A 市西湖建筑集团占道施工有安全隐患',
  '留言时间': '2020/1/6 12:09:38',
  '留言详情': '\n\t\t\t\t\t\n\t\t\t\tA3 区大道西行便道，未管所路口至加油站路段，人行道包括路灯杆，被圈西湖建筑集团燕子山安置房项目施工围墙内。每天尤其上下班期间这条路上人流车流极多，安全隐患非常大。强烈请求文明城市 A 市，尽快整改这个极不文明的路段。\n\t\t\t\t\t\n\t\t\t\t\t\n\t\t\t\t',
  '一级标签': '城乡建设'},
 {'留言编号': 37.0,
  '留言用户': 'U0008473',
  '留言主题': 'A 市在水一方大厦人为烂尾多年，安全隐患严重',
  '留言时间': '2020/1/4 11:17:46',
  '留言详情': '\n\t\t\t\t\t\n\t\t\t\t\t 位于书院路主干道的在水一方大厦一楼至四楼人为拆除水、电等设施后，烂尾多年，用护栏围着，不但占用人行道路，而且护栏锈迹斑斑，随时可能倒塌，危机过往行人和车辆安全。请求有关部门牵头处理。\n\t\t\t\t\t\n\t\t\t\t\t\n\t\t\t\t',
  '一级标签': '城乡建设'}]
```

2.1.2.4　jieba 分词并去除停用词生成文本

接下来我们对训练集和数据集的第 5 列留言详情列分别进行去除停用词和特殊符号处理，把训练集中每一条留言详情文本经过分词和去除停用词后的列表统一保存到 train_texts 列表中，把测试集中的留言详情文本经过处理后的列表保存到 test_texts 列表中。于是我们得到了一个 len(train_texts)=9210 的训练集数据和 len(test_texts)=495 的测试集数据。

```
train_texts=[]
for i in range(len(train)):
    train_process=jieba.lcut(train[i][list(train[i])[4]])    #将留言详情进行精确模式分词
    train_middle=[]
    for word in train_process:                               #去除停用词
        if word not in stop:
            train_middle.append(word)
    train_texts.append(train_middle)


test_texts=[]
for i in range(len(test)):
    test_process=jieba.lcut(test[i][list(test[i])[4]])
    test_middle=[]
    for word in test_process:
        if word not in stop:
            test_middle.append(word)
```

```
    test_texts.append(test_middle)
```

同时，我们将第五列一级分类标签列的分类内容分别保存为 list 类型的 train_classify 和 list 类型的 test_classify

```
train_classify=[]
for i in range(len(train)):
    train_classify.append(train[i][list(train[i])[5]])

test_classify=[]
for i in range(len(test)):
    test_classify.append(test[i][list(test[i])[5]])
```

我们可以输出 train_classify 的前五条数据看一下，整个 train_classify 是一个列表的形式，而其中的每一个元素都是字符串

```
train_classify[0:5]
['城乡建设', '城乡建设', '城乡建设', '城乡建设', '城乡建设']
```

接下来，我们建立每条留言详情所对应的一级分类的标签的数字列表对应

train_labesl 和 test_labels.

2.1.2.5　生成独热编码标签

这里我们用到了独热编码(one-hot)的方法，使每一条留言其对应分类位置的元素为 1,，即我们把城乡建设的类别的独热编码记为，[1,0,0,0,0,0,0]；环境保护的类别的独热编码记为，[0,1,0,0,0,0,0]，以此类推。在此，我们用到的是 list.count(str)的方法对所有 list 中值等于 str 的元素进行计数，从而统计出每一个类别的数量。这里我们一共有 7 中分类，因此我们对每一个分类标签建立的都是一个 7 维的独热编码向量，从而 train_labels 和 test_labels 的 shape 都是(9210,7)。

```
train_labels=([[1,0,0,0,0,0,0]]*train_classify.count('城乡建设')+
              [[0,1,0,0,0,0,0]]*train_classify.count('环境保护')+
              [[0,0,1,0,0,0,0]]*train_classify.count('交通运输')+
              [[0,0,0,1,0,0,0]]*train_classify.count('教育文体')+
              [[0,0,0,0,1,0,0]]*train_classify.count('劳动和社会保障')+
              [[0,0,0,0,0,1,0]]*train_classify.count('商贸旅游')+
              [[0,0,0,0,0,0,1]]*train_classify.count('卫生计生'))


test_labels=([[1,0,0,0,0,0,0]]*test_classify.count('城乡建设')+
              [[0,1,0,0,0,0,0]]*test_classify.count('环境保护')+
              [[0,0,1,0,0,0,0]]*test_classify.count('交通运输')+
              [[0,0,0,1,0,0,0]]*test_classify.count('教育文体')+
              [[0,0,0,0,1,0,0]]*test_classify.count('劳动和社会保障')+
              [[0,0,0,0,0,1,0]]*test_classify.count('商贸旅游')+
```

```
[[0,0,0,0,0,0,1]]*test_classify.count('卫生计生'))
```

我们可以看到经过独热编码转换后的 train_labels 的前 5 项的值如下。其中的每一项都是列表的形式，于是我们之后在预测的时候可以很方便的将独热编码转换为分类标签。

```
print(train_labels[0:5])
[[1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0]]
```

整个数据读取及处理完毕之后我们可以看到训练集的 texts，labels 和 classify 之间是严谨的一一对应的关系。

```
print(train_texts[0])
['A3', '区', '大道', '西行', '道', '未管', '路口', '加油站', '路段', '人行道', '包括', '路灯', '杆', '圈', '西湖', '建筑', '集团', '燕子', '山', '安置', '房', '项目', '施工', '围墙', '上下班', '期间', '条', '路上', '人流', '车流', '安全隐患', '请求', '文明城市', '市', '整改', '文明', '路段']
print(train_labels[0])
[1, 0, 0, 0, 0, 0, 0]
print(train_classify[0])
城乡建设
```

2.1.2.6　文本转换为数字序列

我们把文本信息读取进来之后，要用自然语言处理的方法，将文本信息转换成为数字化的信息。首先我们要建立一个词汇词典 token，我们用到 Keras 中的一个分词器 Tokenizer 的方法。Tokenizer 是一个用于向量化文本，或将文本转换

为序列（即单词在字典中的下标构成的列表，从 1 算起）的类。Tokenizer 生成了一个字典，并且统计了词频等信息。

我们从一大堆文本中提取相关的词汇，对这些词我们要给出一个数字化的编号，而语言的词汇库是非常大的，所以并不是所有的词语我们都要考虑在内。于是参数 num_word 定义了我们准备最多处理的词语量，这些词语是在我们即将处理的文本中出现频率排名最高的 num_word 个，此时一些比较生僻的词语我们将不纳入计算范围。那么我们如何得到频率最高的 num_word 个词语呢？这里我们用了 fit_on_texts 的方法，将我们要处理的文本放入 fit_on_texts 函数中，程序将会自动进行计算和筛选。

```
token=keras.preprocessing.text.Tokenizer(num_words=4000)
token.fit_on_texts(train_texts)
```

我们可以取 token 的前 100 个元素来看一下。token 的一个属性 word_index 返回一个字典，字典中是每一个单词和它所出现频率排序的编号，比如‘年’出现频率最高，排名为 1；而‘文件’出现频率较低，排名 51。

```
print(token.word_index)
{'年': 1, '市': 2, '月': 3, '县': 4, '领导': 5, '公司': 6, '学校': 7, '工作': 8, '政府': 9, '部门': 10, '相关': 11, '西地省': 12, '国家': 13, '区': 14, '中': 15, '情况': 16, '小区': 17, '业主': 18, '学生': 19, '医院': 20, '希望': 21, '请': 22, '人员': 23, '单位': 24, '政策': 25, '解决': 26, '居民': 27, '时': 28, '教师': 29, '企业': 30, '号': 31, '生活': 32, '孩子': 33, '做': 34, '工资': 35, '教育': 36, '职工': 37, '尊敬': 38, '时间': 39, '老百姓': 40, '建设': 41, '老师': 42, '想': 43, '办理': 44, '管理': 45, '未': 46, '退休': 47, '请问': 48, '发展': 49, '影响': 50, '文件': 51, '您好': 52, '教育局': 53, '社会': 54, '开发商': 55,
```

```
'12': 56, '钱': 57, '村民': 58, '家长': 59, '项目': 60, '劳动': 61, '社保': 62, '费用': 63, '房屋': 64, '投诉': 65, '收费': 66, '小学': 67, '标准': 68, '电梯': 69, '工作人员': 70, '环境': 71, '规划': 72, '小孩': 73, '找': 74, '医生': 75, 'k': 76, '地方': 77, '导致': 78, '村': 79, '书记': 80, '污染': 81, '回复': 82, '城市': 83, '请求': 84, '群众': 85, '\r\n': 86, '有限公司': 87, '谢谢': 88, '里': 89, '服务': 90, '局长': 91, '关系': 92, '幼儿园': 93, '经济': 94, '农村': 95, '厅长': 96, '生产': 97, '物业': 98, '电话': 99, '通知': 100}
```

将每一个词语对应的数字索引确定好后，我们将整段的文本转换成数字列表，我们通过 texts_to_sequences 的方法将 train_texts 转换为数字列表 train_sequences，将 test_texts 转换成数字列表 test_sequences

```
train_sequences=token.texts_to_sequences(train_texts)
test_sequences=token.texts_to_sequences(test_texts)
```

我们看到原文本和数字序列的关系如下。每一个在 token 中的词语都对应着一个数字标记，如此生成 train_sequences 和 test_sequences。Token 中的文字和数字是以字典一一对应的形式存在的，因此 train_sequences 和 train_texts 之间可以互相对应的。同时可以说我们经过一系列的处理之后是将原本非常长且复杂的文本变成了一个数字序列，将其简化、降维成计算机非常容易处理的文本信息。

```
print(train_texts[0])
['A3', '区', '大道', '西行', '道', '未管', '路口', '加油站', '路段', '人行道', '包括', '路灯', '杆', '圈', '西湖', '建筑', '集团', '燕子', '山', '安置', '房', '项目', '施工', '围墙', '上下班', '期间', '条', '路上', '人流', '车流', '安全隐患', '请求', '文明城市', '市', '整改', '文明', '路段']
```

```
print(train_sequences[0])
[796, 14, 922, 476, 1577, 3387, 1578, 1579, 409, 1508, 215, 616, 1128, 272, 360,
60, 217, 1945, 3812, 192, 318, 1271, 450, 84, 2413, 2, 524, 805, 1578]
```

2.1.2.7　统一数字序列的长度

接下来我们还要解决一个问题，因为此时生成的每个序列的维数都不尽相同，Keras 也没法处理，因此要将所有的列表维数调整到相同，这里我们用的是 pad_sequences 的方法，其中 maxlen 定义每个列表的最大长度，超过最大长度的部分将被截断；padding='post'，后向填充，即将列表中不满 400 位的部分以 0 补齐；truncating='post'，后向截取，即将所有文本中超过 maxlen 的部分截断。

```
train_lists=keras.preprocessing.sequence.pad_sequences(
                                                    train_sequences,
                                                    padding='post',
                                                    truncating='post',
                                                    maxlen=400)
test_lists=keras.preprocessing.sequence.pad_sequences(
                                                    test_sequences,
                                                    padding='post',
                                                    truncating='post',
                                                    maxlen=400)
```

此时我们的 train_lists 和 test_lists 的 shape 就是很标准的，这样就便于我们在之后的模型处理。

```
train_lists.shape
```

```
(9210, 400)
```

接下来，我们要将训练集的数字序列的列表和训练集的标签数字列表分别转换为数组的形式，并命名为 x_train 和 y_train，以便于之后我们带入到模型中进行模型训练和在测试集上的试验。

```
x_train=np.array(list(train_lists))

y_train=np.array(train_labels)
```

在我们开始进行模型之前，我们还要将模型进行随机打乱顺序，以免会存在巧合的现象，这里我们用到 zip 的方法，对 x_train 和 y_train 进行压缩再解压。

```
order=list(zip(x_train,y_train))

random.shuffle(order)

x_train[:],y_train[:]=zip(*order)
```

2.1.2.8　构建词嵌入模型

接下来我们先介绍一下 RNN 循环神经网络的方法。

循环神经网络(Recurrent Neural Networks，RNNs)已经在众多自然语言处理(Natural Language Processing, NLP)中取得了巨大成功以及广泛应用。RNN 的目的是用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。

RNN 之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入

层的输出还包括上一时刻隐藏层的输出。

理论上，RNN 能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关，RNN 已经被在实践中证明对 NLP 是非常成功的。如词向量表达、语句合法性检查、词性标注等。[1]

我们在这里用 RNN 循环神经网络的方法建立我们的机器深度学习的模型。我们先建立一个模型的框架，用到了 keras 中的 layer.Embedding，即中文词嵌入的方法。

词嵌入是自然语言处理中语言模型与表征学习技术的统称。概念上而言，它是指把一个维数为所有词的数量的高维空间嵌入到一个维数低得多的连续向量空间中，每个单词或词组被映射为实数域上的向量。这是一种对矩阵进行降维的方法。使用词嵌入来表示词组的方法极大提升了自然语言处理中语法分析器和文本情感分析等的效果。

model=keras.models.Sequential() model.add(keras.layers.Embedding(output_dim=64, input_dim=4000, input_length=400)) model.add(keras.layers.Flatten()) model.add(keras.layers.Dense(units=256, activation='relu'))	模型序列化 词嵌入维度 最大词汇数，与 token 的 num_word 一致 tensorflow 的长度，与 pad_sequences 的 maxlen 一致 模型平坦化 神经元的维数 激活函数=‘relu’

model.add(keras.layers.Dropout(0.3)) model.add(keras.layers.Dense(units=7, activation='softmax'))	丢弃值，防止过拟合 与分类标签类别数一致 激活函数=‘softmax’

我们看一下建立好的模型框架的概况，调用 summary 函数，可看到在模型建立时，我们所对模型更改和定义的所有参数都在 summary 中有所显示。

```
model.summary()
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 400, 64)           256000
_________________________________________________________________
flatten_2 (Flatten)          (None, 25600)             0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               6553856
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
```

dense_5 (Dense) (None, 7) 1799 === Total params: 6,811,655 Trainable params: 6,811,655 Non-trainable params: 0 _____

由此，我们将模型的框架建立完成。

这里的 dropout 是为了防止过拟合而设置的一个丢弃值。Dropout 正则化是最简单的神经网络正则化方法，其原理十分简单：任意丢弃神经网络层中的输入，该层可以是数据样本中的输入变量或来自先前层的激活。它能够模拟具有大量不同网络结构的神经网络。

如果模型出现准确率下降等情况，我们可以考虑调整 dropout 值来进行新一轮的模型拟合。

接下来我们对模型进行设置，调用 compile 函数。

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])	优化器='adam' 损失函数用分类的交叉熵 度量用准确率

2.1.2.9 训练模型

接下来我们用 fit 函数进行模型的训练，将 x_train, y_train 代入模型，观察其 accuracy 值。我们将 fit 函数中的参数验证集分割，训练周期，一次性批量样本数，训练过程可视化显示参数设置好了之后，便等待模型进行拟合。

history=model.fit(x_train,y_train, validation_split=0.2, epochs=10, batch_size=128, verbose=1)	代入模型 验证集划分率 训练周期 一次性批量样本数 训练过程可视化显示参数

在这里我们将 padding='post', truncating='post' 和 padding='pre', truncating='pre'两种方法进行运行比较。

下面两张表是 padding='post', truncating='post'的模型的训练结果及其在测试集上的准确率

```
Train on 7368 samples, validate on 1842 samples
Epoch 1/10
7368/7368 [==============================] - 2s 317us/sample
- loss: 1.6439 - accuracy: 0.3890 - val_loss: 1.3212 - val_accuracy: 0.5695
Epoch 2/10
7368/7368 [==============================] - 2s 241us/sample
- loss: 0.9753 - accuracy: 0.7013 - val_loss: 0.6702 - val_accuracy: 0.7959
Epoch 3/10
7368/7368 [==============================] - 2s 249us/sample
- loss: 0.5285 - accuracy: 0.8417 - val_loss: 0.3894 - val_accuracy: 0.8952
Epoch 4/10
7368/7368 [==============================] - 2s 245us/sample
- loss: 0.2982 - accuracy: 0.9258 - val_loss: 0.2443 - val_accuracy: 0.9435
```

```
Epoch 5/10
7368/7368 [==============================] - 2s 251us/sample
- loss: 0.1814 - accuracy: 0.9578 - val_loss: 0.1732 - val_accuracy: 0.9566
Epoch 6/10
7368/7368 [==============================] - 2s 245us/sample
- loss: 0.1169 - accuracy: 0.9735 - val_loss: 0.1413 - val_accuracy: 0.9642
Epoch 7/10
7368/7368 [==============================] - 2s 244us/sample
- loss: 0.0829 - accuracy: 0.9805 - val_loss: 0.1069 - val_accuracy: 0.9739
Epoch 8/10
7368/7368 [==============================] - 2s 322us/sample
- loss: 0.0604 - accuracy: 0.9862 - val_loss: 0.0933 - val_accuracy: 0.9772
Epoch 9/10
7368/7368 [==============================] - 3s 366us/sample
- loss: 0.0459 - accuracy: 0.9900 - val_loss: 0.0805 - val_accuracy: 0.9799
Epoch 10/10
7368/7368 [==============================] - 3s 357us/sample
- loss: 0.0359 - accuracy: 0.9914 - val_loss: 0.0740 - val_accuracy: 0.9826
```

经过 10 轮的模型拟合周期后，我们可以看到模型的损失值之后 0.0036，非常的小，而准确率达到了 0.9991，模型在验证集上的损失值是 0.0656，在验证集上的准确率是 0.9919，整体还是一个相当不错的结果。

```
test_loss,test_acc=model.evaluate(test_lists,np.array(test_labels),verbose=1)
print("test_accuracy:",test_acc)


495/495 [==============================] - 0s 137us/sample - loss: 0.9512 - accuracy: 0.8101
```

我们再来看一下另一种情况。下面两张表是 padding='pre', truncating='pre' 的模型运行结果及其在验证集上的准确率

经过 10 轮的模型拟合周期后，我们可以看到模型的损失值之后 0.0740，非常的小，而准确率达到了 0.9826，相较于前一种情况稍低，模型在验证集上的损失值是 0.0770，在验证集上的准确率是 0.9810，整体还是一个相当不错的结果。

从上述两个方法的比较中我们可以看到后向截取的准确率要比前向截取的准确率高，因此我们可以大致认为群众留言的主要信息应在其留言内容详情的前半部分。

接下来，我们将测试集中预测值分类的独热编码与真实值分类的独热编码不同的测试集文本序号输出，并同时输出真实值分类的独热编码进行对照。在这之后，我们可以进行人工二次分类。

```
for i in range(len(prediction)):
    if list(map(lambda x: round(x), prediction[i]))!=test_labels[i]:
        print("the number",i,"th prediction is not correct!")
        print(prediction[i])
        print(test_labels[i])
```

得到如下结果。我们可以看到，在测试集的 495 条数据中，预测错误的还是

不少的，因此我们考虑可能是验证集中的有些文字是在训练集中没有出现过的，所以我们需要加大训练集的规模。以下仅列举 4 条作为参考。

```
the number 156 th prediction is not correct!
[9.99996424e-01 5.68570727e-07 1.61570540e-06 1.08364915e-07
 1.80354405e-07 9.03420414e-07 2.85562205e-08]
[0, 0, 1, 0, 0, 0, 0]

the number 167 th prediction is not correct!
[4.1122599e-05 3.5311453e-07 8.0346772e-03 1.3342916e-09 4.3014420e-08
 9.9192327e-01 5.1875327e-07]
[0, 0, 1, 0, 0, 0, 0]

the number 310 th prediction is not correct!
[9.8828155e-01 3.8474012e-05 1.1597826e-03 4.9300352e-04 9.6198656e-03
 3.7375471e-04 3.3624070e-05]
[0, 0, 0, 0, 1, 0, 0]

the number 323 th prediction is not correct!
[9.5781088e-06 3.0946557e-04 1.9700261e-10 9.9967861e-01 4.6056414e-07
 1.7172457e-06 2.2861828e-07]
[0, 0, 0, 0, 1, 0, 0]
```

2.1.2.10　验证并调整模型

接下来，我们用混淆矩阵的方法来计算 F-Score 值。

混淆矩阵也称误差矩阵，是表示精度评价的一种标准格式，用 n 行 n 列的矩阵形式来表示。在自然语言分类中，混淆矩阵是可视化工具，特别用于监督学习。混淆矩阵的每一列代表了预测类别，每一列的总数表示预测为该类别的数据的数目，每一列中的数值表示真实数据被预测为该类的数目；每一行代表了数据的真实归属类别，每一行的数据总数表示该类别的数据实例的数目。

我们将预测值独热编码对应的列表找出其中最大值的元素，并把它们放在 y_pred 列表中，将真实值独热编码对应的列表找出其中最大值得元素，并把它们放在 y_true 列表中。

```
y_pred=[]
for i in range(len(prediction)):
    y_pred.append(list(prediction[i]).index(max(prediction[i])))
y_true=[]
for i in range(len(test_labels)):
    y_true.append(list(test_labels[i]).index(max(test_labels[i])))
```

然后用 sklearn.metrics 中的 confusion_matrix 函数对两个列表中元素进行一一对照比较，并将比较结果放入 matrix 矩阵中。

```
matrix=confusion_matrix(y_true,y_pred)
```

形成一个 7X7 的矩阵如下，我们可以看到在对角线以外的数字之和还是不少的的，但这和我们刚刚输出的 prediction is not correct 的数目是相同的。

```
matrix
  array([[42,  4,  5,  5,  16, 25,  4],
         [ 1, 31,  0,  0,  1,  0,  0],
         [ 2,  2, 40,  1,  2,  5,  3],
         [ 2,  0,  0, 86,  1,  3,  4],
         [ 0,  0,  1,  2, 98,  1,  2],
         [ 0,  0,  0,  1,  0, 46,  1],
         [ 0,  0,  0,  0,  0,  0, 58]], dtype=int64)
```

如此一来，我们便可用相应的公式计算 F-Score 的值。

F-Score 是一种非常常用的检验模型准确性的方法，其中要计算的两个因是 precision 和 recall。Precision 是查准率，recall 是查全率。用通俗一点的方法解释，就是假设我们有 100 个西瓜，一共有 60 个好西瓜和 40 个坏西瓜，而且我们已经有这 100 个西瓜好坏的标准答案。这是我们用模型训练预测出来有 72 个好西瓜，那么剩下的 28 个就是我们预测的坏西瓜。而这 72 个我们预测的好西瓜中预测对了 52 个，预测错了 20 个，那么在 28 个中，我们预测对了 20 个，预测错了 8 个，因此我们可以做出如下的表

		预测值		
		好西瓜	坏西瓜	
真实值	好西瓜	52	8	60
	坏西瓜	20	20	40
		72	28	100

在上面的例子中，我们的查准率 precision 是 52/72=0.7222，查全率 recall

是 52/60=0.86666，那么我们要计算的 F-Score 值就是

2*0.72*0.86/(0.72+0.86)=0.7837。在计算 F-Score 的式子中，我们不妨将它的算式进行化简：

$$F = \frac{2 * P * R}{P + R} = \frac{2 * P}{\frac{P}{R} + 1}$$

如果我们要使 F-Score 值越大的话，我们就要让查全率 recall 越大。

我们套用公式多维 F-Score 计算公式：

$$F = \frac{1}{n} \Sigma \frac{2 \times R_i \times P_i}{R_i + P_i}$$

对 F 值进行计算实现，这里我们遍历混淆矩阵 confusion_matrix 中的每一列，用 sum 求和的方法和 list 对应的方法取到矩阵中所需要的的元素。

```
f1=0
for i in range(len(matrix)):
    f1+=(2*(matrix[i][i]/sum(matrix[i]))*(matrix[i][i]/sum(matrix[:,i])))\
    /((matrix[i][i]/sum(matrix[i]))+(matrix[i][i]/sum(matrix[:,i])))
f_score=f1/len(matrix)
```

得出结果，我们看到模型的 F-Score 值并不是很高。因此需要进行模型的优化。

```
f_score
0.822954028078286
```

最后，我们对我们的模型进行保存，以便下次使用时能直接调用模型，而不用再运行一遍主程序

```
#模型保存
model.save('E:/示例数据/C 题/model.h5')
#模型读取
model=load_model('E:/示例数据/C 题/model.h5')
```

至此，我们的群众留言一级标签分类模型已建立完毕并能成功解决分类的问题。

接下来，我们将考虑模型优化的问题。

模型的建立会涉及到很多的参数，同时还和我们输入到模型中的数据有关，如果模型的结果不理想是，我们可以从这些可变的量去着手。

我们来看一下可以有哪些参数或数据可以修改：停用词库的大小，训练文本选择留言主题还是留言详情，最大计算词数的数量多少，词嵌入维数，最大词向量长度，神经元个数，验证集划分，训练周期。

停用词库越大，去除的文档中的无效信息越多，留下的有用信息越精准，这对于我们的模型来说会有极大的帮助，当然停用词库的更新是我们需要日积月累而去添加的。

训练文本的选择。如果选择留言主题作为我们分析的对象，那么我们所能提取到的有效信息仅仅只有群众在进行留言时的 短短十几个字的内容，这对我们进行模型训练时并没有起到很大的帮助；而将留言内容作为我们分析的对象时，我们将能够从文本中提取到许多有用的信息，能够创建一个庞大的数据库，这对于我们的分析和模型的训练时极为有利的。

计算最大词数的数量多少，在我们进行的模型训练中，我们的最大词数的数量是选择了 4000，那么出现频率在 4000 以外的词语我们是不会进行词语对应序

列的转换的，因此可能会存在以下两种情况。第一种是 4000 个词语仍不满足需求，我们还是将有效信息排除在外，这样导致我们在分析时会出现缺损有效信息的情况，从而导致模型准确率下降。第二种是 4000 个词超出了我们有用词的个数，导致我们将很多的无用词也考虑在内，这样也会导致我们模型的准确率下降。

词嵌入维数，这个参数是我们定义将我们的数字序列压缩成多少维的向量空间。如果我们增加词嵌入的维数，我们得到的向量的信息也会更详细，这对于我们的模型训练也是十分有利的。

最大词向量长度，这是针对我们在建立数字序列是所要考虑的因素，为了使每个文本对应的数字序列保持同样的维数，我们必须截长补短，那么这里出现的截长就是我们最大词向量的长度。这个参数的选择也是影响十分大的，如果我们的长度选取的太小，会导致将有效信息删除，而截取的太长，可能会带有很大的空白部分，因此也需要不断地尝试。

神经元个数。神经元就相当于我们在输入和输出变量之间的媒介，这个媒介选取的越多，在模型处理中所进行的步骤就越详细，模型也就越可靠。

验证集划分。我们拿到一个庞大的文本数据文件，在训练模型时，不会将所有的文本都拿来做训练，而是会留下一部分的文本作为验证集，从而先检验模型在验证集上的准确率，在经过模型调优之后的最终模型才会拿来作为实验结果。因此我们这里的验证集划分参数是更改我们模型所接纳的训练集数量和用于验证的验证集数量。

2.1.2.11　结论

最后我在这里展示我们经过模型调优之后的模型的各项参数，以及在测试集上的准确率和 F-Score 值。

```
model.summary()

Model: "sequential_42"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

embedding_41 (Embedding)     (None, 400, 32)           128000

_________________________________________________________________

flatten_41 (Flatten)         (None, 12800)             0

_________________________________________________________________

dense_82 (Dense)             (None, 128)               1638528

_________________________________________________________________

dropout_41 (Dropout)         (None, 128)               0

_________________________________________________________________

dense_83 (Dense)             (None, 7)                 903

=================================================================

Total params: 1,767,431

Trainable params: 1,767,431

Non-trainable params: 0

_________________________________________________________________
```

```
test_loss,test_acc=model.evaluate(test_lists,np.array(test_labels),verbose=1)
```

```
print("test_accuracy:",test_acc)



495/495 [==============================] - 0s 323us/sample -

loss: 0.0770 - acc: 0.9414

      test accuracy: 0.941483
```

```
f_score

0.921437582916847
```

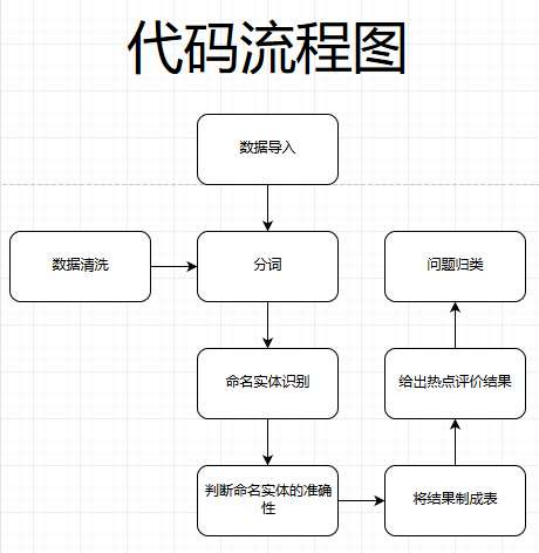
我们总体的模型结果还是很不错的。

2.2、热点问题挖掘

2.2.1、问题分析

群众的热点挖掘问题是一项针对中文文本的自然语言挖掘与处理的问题。我们需要解决的问题是一段时间内群众对某一地点的情况的放映数量的多少。我们需要从一条文本中提取关键的人名、地点等信息。而之后需要对文本进行归类。然后对每一个文本进行热度评价和排名。最后，将所有信息重新制成表。问题的难点在于地点、人群表达的多样化，也就是说需要有效的挖掘工具，还拥有相似计算的复杂性。

2.2.2、问题解决方案



我们首先读取文件，我们作为热度分析的数据是附件 3 中的留言主题列。信息是有不同编码格式的，我们通常将 codes 转化为 unicode 的格式来解决。

首先我们对文件进行每一句话用 jieba 库的 posseg 函数进行中文分词及词性标注，词性标注这里基本可以照搬分词的工作，在汉语中，大多数词语只有一个词性，或者出现频次最高的词性远远高于第二位的词性。据说单纯选取最高频词性，就能实现 80%准确率的中文词性标注程序。但其中有不少难点，一，相对于英文，中文缺少词形态变化，不能从词的形态来识别词性,；二，一词多词性很常见。统计发现，一词多词性的概率高达 22.5%。而且越常用的词，多词性现象越严重。比如“研究”既可以是名词（“基础性研究”），也可以是动词（“研究计算机科学”）；三，词性划分标准不统一。词类划分粒度和标记符号等，目前还没有一个广泛认可的统一的标准。比如 LDC 标注语料中，将汉语一级词性划分为 33 类，而北京大学语料库则将其划分为 26 类。词类划分标准和标记符号的不统一，以及分词规范的含糊，都给词性标注带来了很大的困难。jieba 分词采用了使用较为广泛的 ICTCLAS 汉语词性标注集规范；四、未登录词问题。和分词一样，未登录词的词性也是一个比较大的课题。未登录词不能通过查找字典的方式获取

词性，可以采用 HMM 隐马尔科夫模型[2]等基于统计的算法。知道这些难点后，我们进行命名实体识别，并将结果保存为一个 txt 格式文件。

成功识别以后，需要将文本的命名实体识别给提取出来，就需要把组成命名实体的词给单个提取出来且根据标记连接短语词组，其中去去的命名实体类型是地点、时间，我们以空格为切割标志，使每一个命名实体标注词组为列表的一个元素，后创造两个空列表，分别保存命名实体构成的词组的各个词，我们需要验证一下，即判断每个词是否是命名实体，我们主要使用正则表达式进行判断，之后为了进行连词处理我们判断多个词构成的命名实体，并进行连词处理，最后输出命名实体的结果。

在定义热度评价标准的时候，我们考虑的是

$$热点指数 = \frac{某时间段反映某地点的问题数}{时间段跨度的数学折算}$$

在这里，我们定义了我们自己的一套时间段跨度的数学折算。我们把某问题的所有反映的文本数全都挑选出来，按时间参数进行从早到晚的排序，将最早和最晚的时间挑选出来。接着，我们以月为时间单位，每月为 1，即一年为 12，将最晚的日期减去最早的日期，并通过转换得出月份数，再对月份数进行数学折算。假设我们有最早日期为 2016/05/03，最晚日期为 2019/04/06，此段时间内某问题反映的次数为 25 次，我们将两个日期相减，得到是 2 年 11 个月，按照时间跨度的数学折算，则为 35，因此我们通过我们的计算公式可得出

$$热点指数 = \frac{25}{35} = 0.714$$

在得到所有问题的热点指数后，我们将所有的热点指数进行由大到小的排序，从而挑选出热点指数最高的 5 条作为热点问题

这里我们展示一下我们的运行代码

```
sort_token=sorted(token.word_counts.items(),key=lambda x:x[1],reverse=True)
sort_token=[('街道', 199), ('建议', 167), ('噪音', 147), ('施工', 140),
                ('社区', 132), ('建设', 125), ('违规', 114), ('居民', 112),
                ('解决', 110), ('规划', 91), ('物业', 90),
                ('幼儿园', 87), ('有限公司', 75), ('房屋', 74), ('车位', 73),
                ('公司', 69), ('影响', 68), ('安置', 67), ('公交车', 66),
                ('项目', 66), ('购房', 64), ('开发商', 64),
                ('拖欠', 62), ('小学', 61), ('苑', 61), ('公园', 61),
                ('安全隐患', 60), ('拆迁', 59), ('中学', 59), ('非法', 58),
                ('改造', 57), ('质量', 57), ('违法', 54), ('销售', 53),
                ('学生', 49), ('收费', 47), ('违建', 47),
                ('道路', 44), ('设置', 44), ('学校', 44), ('经营', 44),
                ('工地', 44), ('医院', 44), ('生活', 42),
                ('学院', 42), ('县星', 41)]
take=[]
for m in range(len(sort_token)):
    for i in range(len(train_texts)):
        for j in range(len(train_texts[i])):
            if train_texts[i][j]==sort_token[m][0]:
                take.append(train_texts[i][j-1:j+2])
count_take=dict()
for i in range(len(take)):
```

```
    if len(take[i])>0:
        if take[i][0] in count_take:
            count_take[take[i][0]]+=1
        else:
            count_take[take[i][0]]=1
count_take=sorted(count_take.items(),key=lambda x:x[1],reverse=True)
ihave=[]
for i in range(len(sort_token)):
    ihave.append(sort_token[i][0])
uha=[]
for i in range(len(count_take)):
    if count_take[i][0] not in ihave:
        uha.append(count_take[i])
```

最后的这个 uha 是我们进行热点问题搜索的参考

以下是我们展示一部分我们找出的前 5 大热点问题的表 1

	A	B	C	D	E	F	G	H
1	热度排名	问题ID	热度指数	时间范围	地点人群	问题描述		
2	1	1	10.26	2019/07/07到2019/09/01	A市伊景园滨河苑	伊景园滨河苑捆绑销售车位		
3	2	2	7.06	2018/10/27到2020/01/07	A市地铁建设	地铁线路的建设与运营		
4	3	3	5.88	2019/04/10到2020/01/26	A市丽发新城	搅拌站噪音和污染扰民		
5	4	4	1.58	2019/01/07到2019/12/11	A7县楚龙街道	旧房整治和改造		
6	5	5	1.1	2019/02/22到2019/12/28	A7县星沙街道	公共区域整改		
7								
8								
9								
10								

以下是我们将附件 3 中所有此问题的留言都存为表 2

	A	B	C	D	E	F	G	H
1	问题ID	留言编号	留言用户	留言主题	留言时间	留言详情	反对数	点赞数
2	1	190337	A0009051	关于伊景园滨河苑捆绑销售车位的维权投诉	2019/8/23 12:22	投诉伊景园滨河苑开发商捆绑销售车位！A	0	0
3	1	191001	A909171	A市伊景园滨河苑协商要求购房同时必须购买车位	2019/8/16 9:21	商品房伊景园滨河苑项目是由A市政府办牵	1	12
4	1	195995	A909199	关于广铁集团铁路职工定向商品房伊景园滨河苑项目的问题	2019/8/10 18:15	尊敬的市政府领导，您好！我是广铁集团基	0	0
5	1	196264	A0009508	投诉A市伊景园滨河苑捆绑车位销售	2019/8/7 19:52	A市伊景园滨河苑现强制要求购房者	0	0
6	1	205277	A909234	伊景园滨河苑捆绑车位销售合法吗？！	2019/8/14 9:28	广铁集团强制要求职工购买伊景园滨河苑楼	0	1
7	1	205982	A909168	坚决反对伊景园滨河苑强制捆绑销售车位	2019/8/3 10:03	我坚决反对伊景园滨河苑捆绑销售车位！原	0	2
8	1	207243	A909175	伊景园滨河苑强行捆绑车位销售给业主	2019/8/23 12:16	您好！A市武广新城片区的伊景园滨河苑是	0	0
9	1	209571	A909200	伊景园滨河苑项目绑定车位出售是否合法合规	2019/8/28 19:32	广铁集团铁路职工定向商品房伊景园滨河苑	0	0
10	1	213584	A909172	投诉A市伊景园滨河苑定向限价商品房违规涨价	2019/7/28 13:09	投诉A市伊景园滨河苑定向限价商品房项目	0	0
11	1	214975	A909182	关于房伊景园滨河苑销售若干问题的投诉	2019/8/22 0:00	尊敬的领导，您好！感谢您在百忙之中抽出	0	3
12	1	234633	A909194	无视消费者权益的A市伊景园滨河苑车位捆绑销售行为	2019/8/20 12:34	伊景园滨河苑项目商品房，广铁集团下发文	0	0
13	1	236301	A909197	和谐社会背景下的A市伊景园滨河苑车位捆绑销售	2019/8/30 16:32	广铁集团与A市政府及A市政工程有限公司签	0	0
14	1	239032	A909169	请维护铁路职工权益取消伊景园滨河苑捆绑销售车位的要求	2019/9/1 10:03	广铁集团和A市政开发有限公司协商，将伊	0	1
15	1	244243	A909198	关于伊景园滨河苑捆绑销售车位的投诉	2019/8/24 18:23	广铁集团铁路职工定向商品房伊景园滨河苑	0	0
16	1	244342	A0008948	投诉A市伊景园滨河苑定向限价商品房违规涨价	2019/7/28 10:36	A市伊景园滨河苑定向限价商品房项	0	0
17	1	244528	A909235	伊景园滨河苑开发商强买强卖！	2019/8/21 19:05	A市广铁集团伊景园滨河苑商品房本来是政	0	2
18	1	246407	A0009959	举报广铁集团在伊景园滨河苑项目非法绑定车位出售	2019/9/1 14:20	我要举报广铁集团在伊景园滨河苑项目非法	0	0
19	1	258037	A909190	投诉伊景园滨河苑捆绑销售车位问题	2019/8/23 11:46	尊敬的领导：我是广铁集团铁路职工，本来武	0	0
20	1	258286	A909185	A市伊景园滨河苑数压百姓	2019/8/28 0:00	A市伊景园滨河苑强制要求购房者捆绑购买	0	0

2.3、答复意见的评价

2.3.1、问题分析

答复意见的评价问题是一个开放的问题，目前并没有完整的解决方法，一项针对中文文本的自然语言挖掘与处理的问题。相关性指的是答复意见的内容是否与问题相关；完整性指的是是否满足某种语法规范；可解释性指的是答复意见中内容的相关解释。一条文本的特征是否能转换为可视化的结果而被计算机所能接受和处理，因此我们需要对文本进行数字化特征的处理。而两条文本间的相关性度量是建立在文本数字化的基础上进行的，我们需要用一定的方法计算数字化之后的文本之间的相似度。最后，通过相似度量进行分析的。

2.3.2、问题解决方案

关键词提取的关键在于对于分词后文本的提取重要信息，根据现有的信息得到最终需要的文本，基于于统计特征的关键词抽取算法的思想是利用文档中词语的统计信息抽取文档的关键词。通常将文本经过预处理得到候选词语的集合，然后采用特征值量化的方式从候选集合中得到关键词。基于统计特征的关键词抽取方法的关键是采用什么样的特征值量化指标的方式，目前常用的有三类，一，基于词权重的特征量化；二，基于词的文档位置的特征量化；三，基于词的关联信息的特征量化。我们主要采取第三种方式，也就是基于词的关联信息的特征量化，

主要包括词的关联信息是指词与词、词与文档的关联程度信息，包括互信息、hits值、贡献度、依存度、TF-IDF 值等。

一个词的 TF 是指这个词在文档中出现的频率，假设一个词 w 在文本中出现了 m 次，而文本中词的总数为 n，那么。一个词的 IDF 是根据语料库得出的，表示这个词在整个语料库中出现的频率。TF-IDF 的优点是实现简单，相对容易理解。但是，TFIDF 算法提取关键词的缺点也也很明显，严重依赖语料库的，需要选取质量高且和所处理文本相符的语料库进行训练。此外，对于 IDF 来说，它本身是一种抑的制噪声的加权，本身是倾向于文本中频率较小的词，这使得 TF-IDF 算法的精度不是很高。TF-IDF 算法还有必然一个缺点就是不能反应词的位置信息，在对关键词进行了提取的时候，词的位置信息的，例如文本的标题文本的首句和尾句等含有重要的信息，应该赋予较高权重。基于统计特征关键词的重点在于特征量化指标计算，不同量化指标得到的记过也不尽相同。不同量化指标作为也有其各自优缺点，实际应用中，通常采用不同的量化指标相结合方式得到 Topk 个词为关键词。我们可以基于这个来实行文本的统计，最后得到关键词信息。

对于得到的关键词我们将提取的关键词进行列表化，才能进行下一步的操作。

接下来进行关键的一步，词的向量必须词嵌入操作，才能得到数字化的词向量，我们使用 word2vec 来进行操作，我们需要输入一个文本文档文件，来训连词语模型，输出词典，这时候每一个词语都有一个 n 维的向量可以对应，就得到了词向量，这是基于分布式假设。

相似度指的是，个体之间的距离，来体现它们的相似度，我们使用最相似的余弦相似度，其公式是

$$\frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.$$

度量两个信息实体之间相似性或距离是所有信息发现任务的核心需求。采用适当的措施不仅可以提高信息选择的质量，而且帮助减少时间和处理成本。这些措施有可能会被使用，余弦相似度也是文本挖掘和信息检索中常用的基于向量相似度度量方法。该方法比较字符串转化为向量空间，利用欧几里德余弦规则计算相似度。这种方法与其他方法相结合来限制向量空间的维数。余弦相似度是内积空间中两个非零向量之间相似度的度量，度量它们之间夹角余弦值。而一个文档的特征为一个向量，其中每个维度值对应于术语在文档中出现的次数。然后，余弦相似性给出一个有用的度量方法，用来衡量两个文档在主题方面多相似。

而答复文本的可解释性，我们又分为两个方面来看。

全局可解释性，就是试图理解“模型如何进行预测？”和“模型的子集如何影响模型决策？”。要立即理解和解释整个模型，我们需要全局可解释性。全局可解释性是指能够基于完整数据集上的依赖（响应）变量和独立（预测变量）特征之间的条件交互来解释和理解模型决策。尝试理解特征交互和重要性始终是理解全球解释的一个很好的一步。当然，在尝试分析交互时，在超过两维或三维之后可视化特征变得非常困难。因此，经常查看可能影响全局知识模型预测的模块化部分和特征子集会有所帮助。全局解释需要完整的模型结构，假设和约束知识。

局部解释性：试图理解“为什么模型为单个实例做出具体决策？”和“为什么模型为一组实例做出具体决策？”。对于本地可解释性，我们不关心模型的固有结构或假设，我们将其视为黑盒子。为了理解单个数据点的预测决策，我们专注于

该数据点并查看该点周围的特征空间中的局部子区域，并尝试基于该局部区域理解该点的模型决策。本地数据分布和特征空间可能表现完全不同，并提供更准确的解释而不是全局解释。局部可解释模型 - 不可知解释（LIME）框架是一种很好的方法，可用于模型不可知的局部解释。我们可以结合使用全局和局部解释来解释一组实例的模型决策。

三、总结

我们在此问题中用到了 python 机器学习深度学习的有关概念，用到了 jieba 分词库，tensorflow 中的 keras 库，model 模型中的 Embedding 函数，Tokenizer 分词器，很好的训练了一个成功地模型，能够解决大部分的文本分类和热点问题挖掘的情况。

我们的模型库在拥有大量训练数据的情况下能够保持着 90%以上的准确率，这对于我们的实验来说是非常成功的。

参考文献

[1] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

[2] https://wenku.baidu.com/view/15532d17aaea998fcc220ec1.html

[3] https://www.ixueshu.com/document/e4859208ffe87b814030c1c437665223318947a18e7f9386.html

[4] http://kreader.cnki.net/Kreader/CatalogViewPage.aspx?dbCode=cdmd&filename=2010113595.nh&tablename=CDFD0911&compose=&first=1&uid=WEEvREcwSlJHSldRa1FhcEFLUmViSGF0Nnhmd2VvRnpxT2I3T3RXUWIVYz0=$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4IQMovwHtwkF4VYPoHbKxJw!!