基于文本内容的网络问政平台留言分类算法

摘要

近年来，随着电子技术、信息技术和网络技术的发展，物联网、移动互联网和云计算等日益发展，并推动了大数据的发展，这为以提供公共服务为核心的智慧政务带来了发展的技术基础和推动力量，智慧政务是在信息化时代背景下，综合运用互联网和信息网络技术，以大数据为核心，通过信息化手段为公众提供高效服务的政务模式，以此推动政府廉洁高效运转、政府政策制定更加精确、服务大众更为便捷、信化透明化程度更高。因此，如何对各类社情民意文本进行快速准确的分类是一个十分关键的问题。本文根据经典的文本分类卷积神经网络模型，设计更符合本文问题的卷积神经网络模型。

针对问题一：第一步先对网络问政平台的留言内容进行数据预处理，对留言主题及其内容的特征有一定清晰了解，在进行文本分词、去除停用词后将文本转化为可视数据，便于直观清晰解决问题；第二步通过使用 word2vec 中的 Skip-gram 模型来训练以重新建构语言学之词文本，第三步根据经典的文本分类卷积神经网络模型，我们进行优化得到更符合本题的卷积网络神经模型，建立关于留言内容的一级标签分类模型；最后使用 F-Score 方法对分类模型进行评估。

针对问题二：同样是先对附件 2 留言内容和主题文本进行数据预处理、文本分词以及去除停用词；再使用 k-means 聚类算法分成多个聚类中心，得到五个热点问题；最后利用基于 gensim 的 TF-IDF 算法计算文本相似度，即通过 gensm 来训练模型，建立模型计算相似度，得到五个热点问题对应文本的留言信息。

针对问题三：先将附件四中的文本评论提出，用一个迭代判断里面是否含有回复的“禁词”，如果超过两个便进入人工审核。通过 TF-IDF 模型判断回复和留言的相关性，如果相关性低于百分之 65，也进入人工审核，且人工审核重点审核两个评价都不过关的回复。

关键词：word2vec、Skip-gram、卷积神经网络、F-Score、 k-means、TF-IDF

目录

一、简介

1.1 挖掘背景

进入 21 世纪互联网领域发生了重大变革，带来了电子政务方面深刻的变化，互联网与智慧电子政务深度融合改变了传统政务治理模式，对于提升政府治理水平、转变政府职能、优化政务效能具有重要意义。

互联网时代,社会信息化应用水平的提高，加速了人际信息的传播速度，大大增强了公民的权利意识和网络话语权。民众的知情权、参与诉求、监督诉求都比以前更为强烈；基层民众对家庭生活、个人发展紧密相关的政务服务需求迫切。对政府管理而言，社会信息化应用水平的提高对政府权力运行公开、舆论引导以及突发事件应对能力提出了更高要求，促使不断创新政府管理。信息整合及其对决策的支撑更深入，政府数据开放要求创造更大价值，面向公众的政务服务更加完善。因此，文本挖掘的研究对建造“智慧政务”具有重要的应用价值。

1.2 挖掘目标

针对问题一：我们要构建一个网络问政平台留言分类模型，该模型可以对文本进行内容去词筛选最后归类，减少人为分类带来的效率低下、差错率高等情况，同时减少人为工作量。

针对问题二：在问题一的基础上，k-means 聚类得选出五个聚类中心即五个热点问题，利用基于 gensim 的 TF-IDF 算法计算文本相似度，该模型有助于及时发现热点问题，有助于相关部门进行有针对性地处理，提升服务效率。

针对问题三：用一个迭代判断数据是否含有回复的“禁词”，利用问题二建立的 TF-IDF 模型，判断回复和留言的相关性，最后建立相对完善的留言评论体系。

二、符号说明

符号	注释
P_{i}	第 i 类的查准率

R_i	第 i 类的查全率
C	窗口大小
T	文本大小

三、问题一模型的建立与求解

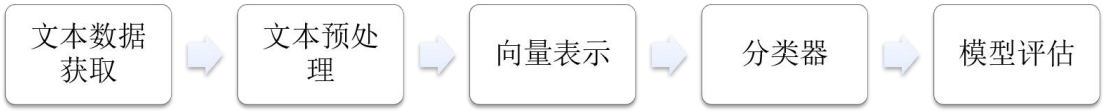


图 1 文本挖掘一般流程

3.1 数据预处理

3.1.1 分词

由于中文文本的特点是词与词之间没有明显的界限，从文本中提取词语时需要分词，本文采用 Python 开发的一个中文分词模块——jieba 分词，对问题和回答中的每一句话进行分词。

jieba 分词包因为它的简易性和高效性被广大工作者广泛的运用。Jieba 分词支持三种分词模式：（1）精确模式，试图将句子最精确地切开，适合文本分析；（2）全模式，把句子中所有的可以成词的词语都扫描出来, 速度非常快，但是不能解决歧义；（3）搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

Jieba 分词用到的算法是基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)，通过采用了动态规划查找最大概率路径, 找出基于词频的最大切分组合，对于未登录词，采用了基于汉字成词能力的 HMM 模型，使用了 Viterbi 算法。

3.1.2 去停用词

停用词是指在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词，这些字或词即被称为 Stop Words（停用词），它们通常是一些单字，单字母以及高频的单词，比如中

文中的“我、的、了、地、吗”等，英文中的“the、this、an、a、of”等。对于停用词一般在预处理阶段就将其删除，避免对文本，特别是短文本，造成负面的影响。

3.1.3word2vec

为了将文本语料输入神经网络进行训练，我们需要把自然语言符号表示成计算机能够理解的数字形式。

word2vec 的目的是用一个向量去表示一个对象，然后基于向量相似度去计算对象的相似度，找到相关的对象。word2vec 提供了一种计算关联的新思路，通过转化为 term 向量的方式，可以计算出任何 term 之间的关联度，同时这种关联能够在一个很快速的时间内被计算出来，这在实际应用中就有很大的价值。

Word2vec 依赖 skip-grams 或连续词袋（CBOW）来建立神经词嵌入。Word2vec 为托马斯·米科洛夫（Tomas Mikolov）在 Google 带领的研究团队创造。该算法渐渐被其他人所分析和解释。Skip-gram 把一个词从词窗剔除。在 skip-grams 下给定 n 词围绕着词 w，word2vec 预测一个句子中其中一个缺漏的词 c，即以机率来表示。相反地，CBOW 给定词窗中的文本，预测当前的词。

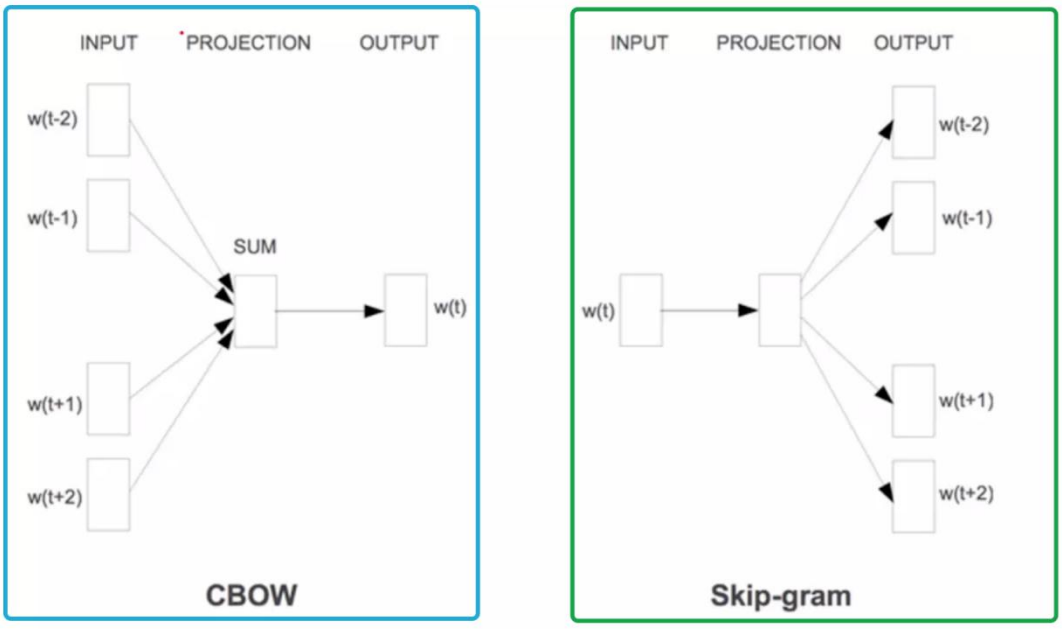


图 2 两种 word2vec 算法网络示意图

我们使用 Skip-gram 模型构建网络问政平台留言分类模型。Skip-gram 模型的训练目标就是使得下式的值最大：

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c}\log p(w_{t+j}|w_t)$$

其中，c 是窗口的大小，T 是训练文本的大小。基本的 Skip_gram 模型计算条件概率。如下式:

$$p(w_O|w_I)=\frac{exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W}exp({v'_w}^T v_{w_I})}$$

Skip-gram 顾名思义就是“跳过某些符号”，例如，句子“中国足球踢得真是太烂了”有 4 个 3 元词组，分别是“中国足球踢得”、“足球踢得真是”、“踢得真是太烂”、“真是太烂了”，可是我们发现，这个句子的本意就是“中国足球太烂”可是上述 4 个 3 元词组并不能反映出这个信息。Skip-gram 模型却允许某些词被跳过，因此可以组成“中国足球太烂”这个 3 元词组。 如果允许跳过 2 个词，即 2-Skip-gram。

3.2 模型的建立

卷积神经网络(convolutional neuralnet-works,CNN)最早由日本学者 Fukushima-12]提出。不同于传统神经网络,CNN 具有局部连接和权值共享的特点。其中局部连接表现为:卷积层中的每个神经元无需与上一层的所有神经元进行连接,仅通过卷积核分别与上一层的各个局部区域进行连接计算,其中卷积核对应一个 wXh 的权重矩阵,权值共享表现为:每个卷积核不管与上一层的哪部分区域的神经元计算都采用同一套权重矩阵。可明显降低参数个数,提升运算能力 CNN 卷积神经网络结构一共分为输人层、卷积层、池化层与输出层。设输人层中输入数据是二维矩阵 X ,大小为 n×r，第 1 层是卷积层,将第 l 层中第 j 个卷积核对应的权重矩阵用 k_{ij}^{l};表示,卷积核 x);对应的偏置用 b},卷积层计算得到的第 j 个特征图用 x_j^{1};，卷积计算公式如下:

$$x_j^l=f(\sum_{i\in M_j}x_j^l\times k_{ij}^l+b_j^l)$$

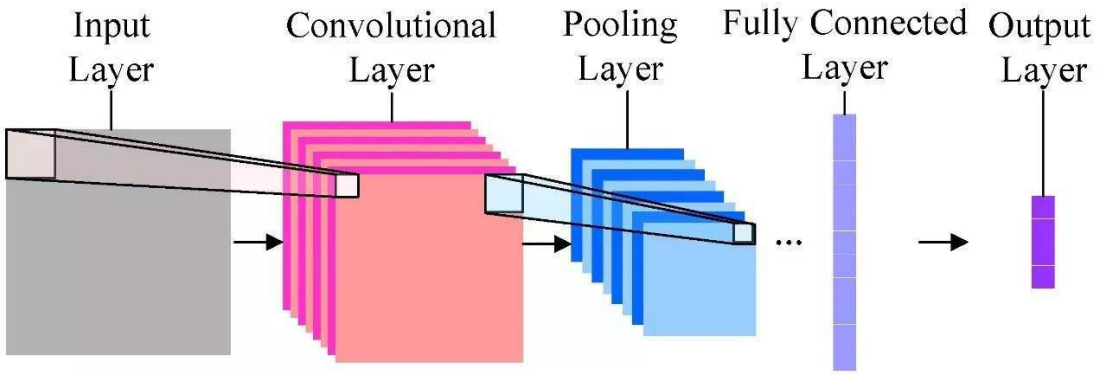


图 3 卷积神经网络一般流程

3.3 模型的求解

3.3.1 实验步骤

根据提供的数据附件 2 中提供的数据，提取出一共 27360 个句子进行预处理（分词，去除停用词），一共得到 64539 个词语，然后用其训练一个的词向量模型（gensim_w2v_sg0_model），其维度为 100 维。然后用词向量表示预处理过的词语，每篇文档表示为 100*100 的矩阵（100：文档中包含的词语，不够 100 个词语的文档用 0 填充，超过 100 个词语的文档只保留 100 个词语。100：每个词语用 100 维的向量表示）

利用设计好的卷积神经网络进行训练，并测试。

3.3.2 设计的卷积神经网络结构及结果

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 50)           3227150
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 98, 64)            9664
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
dense_1 (Dense)              (None, 24)                1560
_________________________________________________________________
dropout_2 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                250
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
F-Score: 0.82421281216069489
```

图 4 卷积神经网络结构及结果

（注释：Layer：神经网络层；Output Shape：输出结构；Param：参数 Embedding_1:嵌入层；conv1d_1：卷积层；global_max_pooling1d_1:池化层；Dropout_1:正则化层；dense：全连接层。）

3.4 结果分析

3.4.1 评估指标选择

目前有多种方法来评估文本挖掘，下面列出几种比较公认的评估方法和指标。（表 1）

指标	计算方法
分类正确率	计算文本样本与待分类文本的概率得出分类正确率
查准率	正确分类的对象所占对象集的大小
查全率	集合中所含指定类别的对象数占实际目标类中对象数的比例
F－score	查准率和查全率的调和均值（查全率* 查准率）/［查全率＋查准率）/2］

表 1 检测指标

因为查准率和查全率在分类问题中，这两个指标往往是成反比的，而且在很大程度上，受预测标准的控制。那么只拿其中的某一个指标去评估预测结果是不

太合适的。从数学上来看，F-score 方法其实是查准率与查全率的调和平均数，对于本题而言该方法综合考虑了预测结果的查准率和查全率，是一个比较好的评估指标。

所以在这里我们采用 F-score 方法进行评估，F-score 公式如下：

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

其中 P_i 为第 i 类的查准率， R_i 为第 i 类的查全率。

3.4.2 评价结果

我们根据建立的模型对附件 2 里的留言文本进行统计分类，最后通过 F-score 进行评价，得到结果 F1 得分为 0.824（见图 4 最下一行数据），总体来说得分值较高，故该分类模型具有一定可用性。

四、问题二模型的建立与求解

4.1 数据预处理

4.1.1 分词、去除停用词

如同问题一的预处理，先对所给附件 3 的文本内容进行分词、去除停用词，这里就不多做赘述。

4.1.2 k-means 聚类

k-means 算法（k-均值聚类算法）是一种基本的已知聚类类别数的划分算法。它是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。它是使用欧氏距离度量的（简单理解就是两点间直线距离，欧氏距离只是将这个距离定义更加规范化，扩展到 N 维而已）。

原始的 k-means 算法首先随机选取 k 个点作为初始聚类中心，然后计算各个数据对象到各聚类中心的距离，把数据对象归到离它最近的那个聚类中心所在的类； 调整后的新类计算新的聚类中心，如果相邻两次的聚类中心没有任何变化，说明数据对象调整结束，聚类准则函数 f 已经收敛。在每次迭代中都要考察每个样本的分类是否正确，若不正确，就要调整。在全部数据调整完成后，再修改聚

类中心，进入下一次迭代。

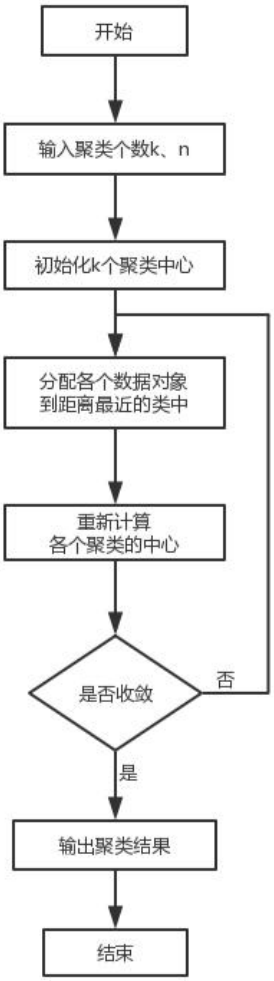


图 5 k-means 算法流程图

4.1.3.基于 gensim 的 TF-IDF

TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。 TF-IDF 的主要思想是：如果某个词在某篇文章中出现的频率高，并且在其他文章中很少出现，则认为该词可以作为该篇文章的关键词。

求某篇文章中某个词的 TF-IDF 的计算公式：

$$TF\text{-}IDF = TF \times IDF$$

TF 为词频，指的是该词在该文章中出现的频率；IDF 为逆文档频率，衡量该词在所有文章中的出现频率。

TF 的计算公式：

TF = 该词在该文章中出现次数 ÷ 该文章中的总词数

IDF 的计算公式：

IDF = log[文章总数 ÷ (出现该词的文章数+1)]

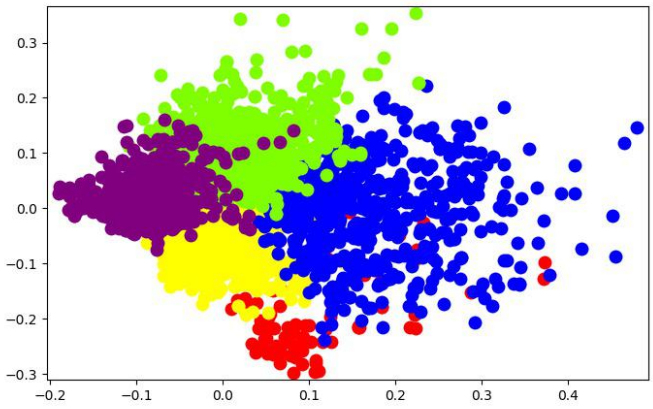
其中 log 的底数为自定义取值，一般取 e。

gensim 是一款强大的自然语言处理工具(python 的第三方库)，用于从原始的非结构化的文本中，无监督地学习到文本隐层的主题向量表达。它支持包括 TF-IDF，LSA，LDA，和 word2vec 在内的多种主题模型算法。

4.2 模型的建立

4.2.1 实验步骤

首先从附件三中提出对热点话题有相关性的文本，将其预处理（分词，去停留词），由于输入 K-Means 模型的数据不能是字符串类型，所以我们需要对文本进行转换。将文本的词语转换成词频矩阵接着计算其 tf-idf 值，然后输入模型中，设置模型聚成 5 类。效果图如下



接着分别从 5 类词语中计算出现频率较高的词语，通过人工干预分别挑选出能够代表一个事件的词语。接着将挑选出来的 5 类词语分别输入 Tf-idf 模型中，计算出原文本中与这 5 类词语相似的事件。

4.3 模型的求解

在 gensim 默认参数下，测试文本中某词的 TF-IDF 计算方法如下：

TF = 该词在测试文本中出现的次数

IDF = log(语料库中的文本总数 ÷ 语料库中出现该词的文本数）÷ log2

其中，log 的底数为 2

TF-IDF(初步值) = TF×IDF

TF-IDF(标准化) = 该词的 TF-IDF 初步值 ÷ (测试文本中所有词的 TF-IDF 初步值的平方和，然后对平方和开方)

标准化使得测试文本中的所有词的 TF-IDF 值的平方和为 1，方便后面计算余弦相似度(只需要计算余弦相似度公式的分子部分即可)。

热度排名	问题 ID	热度指数	时间范围	地点/人群	问题描述
1	1	0.00525	2019/07/28 -2020/1/9	丽发新城小区	搅拌站发出噪音影响居民生活
2	2	0.0017	2019/3/16-2019/9/25	A3 区	有多个店面深夜发出噪音扰民
3	3	0.0025	2019/3/4 -2020/1/8	A3 区	存在多处违章建筑
4	4	0.004	2019/04/28-2019/11/22	A 市经济学院	A 市经济学院强制学生外出实习
5	5	0.003	2019/1/11-2019/7/8	A 市 58 车贷	A 市 58 车贷老板跑路美国，经侦拖延办案

表 2 热点问题表

热点问题留言明细表内容过多，放于附件中。

4.4 结果分析

通过以上模型得到的热点问题前五具体内容如表 2，热点问题明细见附件，热度指数分别为 0.00525、0.0017、0.0025、0.004、0.003。时间的跨度不一，有的持续将近一年，可见部分地区的管理仍不到位。

五、问题三模型的建立与求解

5.1 数据预处理

5.1.1 分词、去除停用词

如同问题一的预处理，先对所给附件 4 的文本内容进行分词、去除停用词，这里就不多做赘述。

5.1.2k-means 聚类

k-means 聚类在前面已经讲述过，这里就不做展开。

在每次迭代中都要考察每个样本的分类是否正确，若不正确，就要调整。在全部数据调整完成后，再修改聚类中心，进入下一次迭代。

5.2 模型的建立

将附件 4 中的评论文本数据提出，用一个迭代判断里面是否含有回复的“禁词”（从网上下载），如果超过两个便进入人工审核。通过 tf-idf 模型判断回复和留言的相关性，如果相关性低于百分之 65，也进入人工审核，且人工审核重点审核两个评价都不过关的回复。

六、模型的评价与展望

6.1 模型的优点

在问题一使用了 word2vec，Word2vec 会考虑上下文，相比 Embedding 方法，效果要更好且维度更少，速度更快、通用性很强，可以用在各种 NLP 任务中。对问题一建立了卷积神经网络模型，该模型的有优点在于共享卷积核，对高维数据处理无压力，且无需手动选取特征，训练好权重，即得特征分类效果好。在建立模型之后使用了 F-score 方法进行评估，建立在查准率和查全率两个指标的基础上，综合地考虑了分类结果的精确性和全面性。

问题二 k-means 聚类算法易于实现，算法的可解释度比较强，主要需要调参的参数仅仅是簇数 k，得出的聚类效果较优。

6.2 模型的缺点

1）卷积神经网络模型需要调参，需要大样本量，训练最好要 GPU。

2）k 均值算法非常简单且使用广泛，但是其有主要的两个缺陷，第一个是 K 值需要预先给定，属于预先知识；第二个是 K-Means 算法对初始选取的聚类中心点是敏感的，不同的随机种子点得到的聚类结果完全不同。

6.3 模型的展望

在问题一如果利用 word2vec 和 TF-IDF 结合进行特征提取会更好提高该分类模型的准确率，使召回率减少，大大减少网络问政平台留言的出错率。

七、参考文献

[1] https://github.com/fxsjy/jieba

[2] 夏海峰，陈军华.基于文本挖掘的投诉热点智能分类[D].上海：上海师范大学，2009.

[3] 弭晓月，毕冠群，黄成梓.基于双重注意力机制与 Bi-LSTM 的智能阅读系统[D].山东：山东大学，2017.

[4] 顾俊.基于关键句的 K-means 算法在热点发现领域的研究与应用[J].2016.

[5] 曾凡锋，李玉珂，肖珂.基于卷积神经网络的语句级新闻分类算法[J]. 2020 年 4 月，第 41 卷(第 4 期).

[6] https://www.jianshu.com/p/e2ea39d83bdd

附录

问题一：

```
Word2vec
import pandas as pd
import jieba
from gensim.models.word2vec import Word2Vec


# 创建停用词 list
stopwords = [line.strip() for line in open("stop_word.txt",
                                           errors="ignore",
                                           encoding='utf-8').read().splitlines()]
```

```
import pandas as pd
import jieba
from gensim.models.word2vec import Word2Vec

# 创建停用词 list
stopwords = [line.strip() for line in open("stop_word.txt",
                                                    errors="ignore",
                                                    encoding='utf-8').read().splitlines()]

# 读取文件
data = pd.read_excel("data.xlsx")
data.dropna()

# 取出对分类有用的特征
data_one = data["留言主题"].values.tolist()
data_two = data["留言详情"].values.tolist()

# 分词,去除停留词
data_two_cut = []
data_one_cut = []

for line in data_one:
    segs = jieba.lcut(line)
    segs = list(filter(lambda x: len(x) > 1, segs))
    segs = list(filter(lambda x: x not in stopwords, segs))
    for seg in segs:
        data_one_cut.append(seg)

for line in data_two:
```

```
        segs = jieba.lcut(line)
        segs = list(filter(lambda x: len(x) > 1, segs))
        segs = list(filter(lambda x: x not in stopwords, segs))
        for seg in segs:
            data_two_cut.append(seg)



a = []
a.append(data_two_cut)
a.append(data_one_cut)



# 构建 Word2Vec 模型
# 保存数据
def text_save(filename, data):            # filename 为写入 CSV 文件的路径，data 为要写入数据列表.
        file = open(filename, 'a')
        for i in range(len(data)):
            s = str(data[i]).replace('[','').replace(']','')   # 去除[],这两行按数据不同，可以选择
            s = s.replace("'","').replace(',','') +'\n'        # 去除单引号，逗号，每行末尾追加换行符
            file.write(s)
        file.close()

text_save(r"E:\about_pycharm\taidi\模型数据.csv", a)

# 读取数据，用 gensim 中的 word2vec 训练词向量
file = open('模型数据.csv')
sss = []
```

```
while True:
    ss = file.readline().replace('\n', '').rstrip()
    if ss == '':
        break
    s1 = ss.split(" ")
    sss.append(s1)
file.close()
model = Word2Vec(size=100, workers=5, sg=1)  # 生成词向量为 100 维，考虑上下 5 个单词共 10 个单词，采用 sg=1 的方法也就是 skip-gram
model.build_vocab(sss)
model.train(sss, total_examples=model.corpus_count, epochs=model.iter)
model.wv.save_word2vec_format('embedding1.txt', binary=False)  # 保存模型
```

卷积神经网络模型

```
import numpy as np
import pandas as pd
import jieba
from keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras import layers
from keras.preprocessing.sequence import pad_sequences
from keras.optimizers import SGD
from sklearn.metrics import accuracy_score, recall_score
# 创建停用词 list
stopwords = [line.strip() for line in open("stop_word.txt",
                                           errors="ignore",
                                           encoding='utf-8').read().splitlines()]
```

```
def create_embedding_matrix(f_path, word_index, embedding_dim):
    vocab_size = len(word_index) + 1
    embedding_matrix = np.zeros((vocab_size, embedding_dim))

    with open(f_path,
              encoding="utf-8",
              errors="ignore") as f:
        for line in f:
            word, *vector = line.split()
            if word in word_index:
                idx = word_index[word]
                embedding_matrix[idx] = np.array(vector,

dtype=np.float)[:embedding_dim]
    return embedding_matrix



# 读取文件
data = pd.read_excel("data.xlsx")
data.dropna()

# 取出对分类有用的特征
data_two = data["留言详情"].values.tolist()
label = data["一级标签"].values.tolist()
label = np.array(label)
# 分词,去除停留词
data_two_cut = []
```

```
for line in data_two:
        segs = jieba.cut(line, cut_all=True)
        segs = list(filter(lambda x: len(x) > 1, segs))
        segs = list(filter(lambda x: x not in stopwords, segs))
        data_two_cut.append(" ".join(segs))

# 打乱顺序
a = pd.DataFrame({"留言详情": data_two_cut,
                              "一级标签": label})
cities = a.reindex(np.random.permutation(a.index))
data_two_cut = cities["留言详情"].values.tolist()
label = cities["一级标签"].values.tolist()



x_train, x_test, y_train, y_test = train_test_split(data_two_cut,
                                                                                label,
                                                                                test_size=0.2,
                                                                                random_state=1234)



tokenizer = Tokenizer()
tokenizer.fit_on_texts(data_two_cut)
x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)

vocab_size = len(tokenizer.word_index) + 1

embedding = 100
```

```
x = create_embedding_matrix("embedding1.txt",
                                        tokenizer.word_index,
                                        embedding_dim=embedding)


# 对标签进行向量化
label_v = LabelEncoder()
y_test = label_v.fit_transform(y_test)
y_train = label_v.fit_transform(y_train)


# 补齐
maxlen = 100
x_test = pad_sequences(x_test, padding="post", maxlen=maxlen)
x_train = pad_sequences(x_train, padding="post", maxlen=maxlen)



# 卷积
modle = Sequential()
embedding_dim = 50
layer = modle.add(layers.Embedding(vocab_size,
                                           embedding_dim,
                                           input_length=maxlen,
                                           trainable=False))
# 第一个卷积层
layer = modle.add(layers.Convolution1D(64,
                                               3,
                                               activation="tanh"))
# 第一个池化
layer = modle.add(layers.GlobalMaxPooling1D())
layer = modle.add(layers.Dropout(0.1))
```

```
layer = modle.add(layers.BatchNormalization())


# 第一个连接层
layer = modle.add(layers.Dense(units=24,
                               activation="tanh",
                               bias_initializer="one",
                               kernel_initializer='random_uniform'))
# Dropout
layer = modle.add(layers.Dropout(0.1))
# 第二个连接层
layer = modle.add(layers.Dense(10,
                               activation="tanh",
                               bias_initializer="zeros"))


layer = modle.add(layers.Dense(units=1,
                               activation="softmax",
                               bias_initializer="zeros"))


epochs = 10
lrate = 0.15
decay = lrate/epochs
sgd = SGD(learning_rate=lrate,
          momentum=1.2,
          decay=decay)


modle.compile(optimizer=sgd,
              loss='binary_crossentropy',
              metrics=["accuracy"])
modle.summary()
```

```
# 训练模型
modle.fit(x_train, y_train, batch_size=60, epochs=epochs)

a = modle.predict(x_test)

accuracy_score=accuracy_score(y_test, a, normalize=True, sample_weight=None)

print("F-Score:", accuracy_score)
```

问题二：

K-means 模型

```
import pandas as pd
import jieba
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
def labels_to_original(labels, forclusterlist):
    assert len(labels) == len(forclusterlist)
    maxlabel = max(labels)
    numberlabel = [i for i in range(0, maxlabel + 1, 1)]
    numberlabel.append(-1)
    result = [[] for i in range(len(numberlabel))]
    for i in range(len(labels)):
        index = numberlabel.index(labels[i])
        result[index].append(forclusterlist[i])
    return result
if __name__ == '__main__':
```

```
    # 分类数
    num = 5
# 读取停留词数据
stopwords = [line.strip() for line in open("stop_word.txt",
                                           errors="ignore",
                                           encoding='utf-8').read().splitlines()]
# 读取数据
data = pd.read_excel("all_data/附件 3.xlsx")
# 提取有用特征
uesful_data = data["留言详情"].values.tolist()
# 分词和去停留词
data_cut = []
for line in uesful_data:
    segs = jieba.cut(line, cut_all=True)
    segs = list(filter(lambda x: len(x) > 1, segs))
    segs = list(filter(lambda x: x not in stopwords, segs))
    data_cut.append(" ".join(segs))
# 该类会将文本中的词语转换成词频矩阵
vectorezer = CountVectorizer()
# 该类会统计每个词的 tf—idf 权值
tf_idf_transfromer = TfidfTransformer()
# 将文本转为词频矩阵并计算 tf—idf
tfidf = tf_idf_transfromer.fit_transform(vectorezer.fit_transform(data_cut))
# 获取词袋模型中的所有词语
tfidf_matrix = tfidf.toarray()
word = vectorezer.get_feature_names()
# 聚成 5 类
clf = KMeans(n_clusters=num)
s = clf.fit(tfidf_matrix)
```

```
# 每个样本所属的簇
label = []
i = 1
while i <= len(clf.labels_):
    label.append(clf.labels_[i - 1])
    i = i + 1
# 获取标签聚类
y_pred = clf.labels_
# pca 降维，将数据转换成二维
pca = PCA(n_components=2)  # 输出两维
newData = pca.fit_transform(tfidf_matrix)  # 载入 N 维
xs, ys = newData[:, 0], newData[:, 1]
# 设置颜色
cluster_colors = {0: 'r', 1: 'yellow', 2: 'b', 3: 'chartreuse',
                  4: 'purple', 5: '#FFC0CB', 6: '#6A5ACD',
                  7: '#98FB98'}
# 设置类名
cluster_names = {0: u'类 0', 1: u'类 1', 2: u'类 2', 3: u'类 3',
                 4: u'类 4', 5: u'类 5', 6: u'类 6', 7: u'类 7'}
df = pd.DataFrame(dict(x=xs, y=ys, label=y_pred, title=data_cut))
groups = df.groupby('label')
fig, ax = plt.subplots(figsize=(8, 5))  # set size
ax.margins(0.02)
for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='',
            ms=10,
            label=cluster_names[name],
            color=cluster_colors[name],
            mec='none')
```

```
plt.savefig('test2.jpg')
plt.show()
res = labels_to_original(y_pred, data_cut)
for i in range(len(res)):
    for j in range(5):
        print(res[i][j])
    print("======================")
```

TF-IDF

```
import jieba
from gensim import corpora, models, similarities
import codecs
import pandas as pd



# 读取停留词数据
stopwords = [line.strip() for line in open("stop_word.txt",
                                           errors="ignore",
                                           encoding='utf-8').read().splitlines()]



a = pd.read_excel("all_data/附件 3.xlsx")
b = a["留言详情"].values
c = pd.DataFrame({"留言详情": b})
c.to_csv("xiangsi_data.txt")

# 读取训练词库
Train_test = 'xiangsi_data.txt'
Traintest = codecs.open(Train_test, 'rb').readlines()
```

```
Traintest = [w.strip() for w in Traintest]
# 分词完毕得到结果
Traintest_word = []
for word in Traintest:
    words_list = [words for words in jieba.lcut(word)]
    Traintest_word.append(words_list)
# 去除停用词
result =[]
result1 = []
for word in Traintest_word:
    if word not in stopwords:
        result = result1.append(word)

# 测试用词
a = [line.strip() for line in open("test_1.txt",
                                   errors="ignore",
                                   encoding='utf-8').read().splitlines()]

doc_test = a
doc_test_list = doc_test
TestResult = doc_test_list

# 用 dictionary 方法获取词袋
dictionary = corpora.Dictionary(Traintest_word)
# 词袋中用数字对所有词进行了编号
dictionary.keys()
# 使用 doc2bow 制作语料库，利用词袋模型中的字典将其映射到向量空间
corpus = [dictionary.doc2bow(doc) for doc in Traintest_word]
# 对测试文档也进行制作语料库，利用词袋模型中的字典将其映射到向量空间
```

```
doc_test_vec = dictionary.doc2bow(doc_test_list)


# 使用 TF-IDF 模型对语料库建模
tfidf = models.TfidfModel(corpus)
# 获文档中，每个词的 TF-IDF 值 tfidf[corpus]
# 对每个目标文档，分析测试文档的相似度
index = similarities.SparseMatrixSimilarity(tfidf[corpus], num_features=len(dictionary.keys()))
sim = index[tfidf[doc_test_vec]]
# 根据相似度排序是一个列表 ?表中每一项是一个元组 ？元组中前面是原句索引 ?后面是相似度
SimilaritiesList = sorted(enumerate(sim), key=lambda item: -item[1])
num = 0
while(num<=100):
    Result_tutple = SimilaritiesList[num]   # 获取元组 ？索引 ?相似度
    # 获取索引
    Result_index = Result_tutple[0]
    if len(Traintest_word[Result_index]) > 5:
        print(Result_tutple)
        print(Traintest_word[Result_index])     # 输出分词后数值
    num = num + 1
```