题目 “智慧政务”中的文本挖掘应用

摘 要

本文是针对智慧政务中的文本数据挖掘应用的研究。通过建立基于三层网络结构的**fastText 文本分类模型，聚类量化模型，熵权评估模型**解决了群众留言分类，热点问题挖掘，答复意见评价等问题。

针对群众留言分类问题，本文利用所给数据进行词频统计和词云图分析。得到所给训练集是一种不平衡数据集，我们对已有的数据集进行采样，从而**扩充训练集，解决数据不平衡问题**。为了方便应用分类模型，我们对文本数据进行正则预处理，去停用词，jieba 智能分词来获取特征文本。分词部分，我们利用逆向最大匹配分词算法 BMM 和 jieba 分词实现了更好的分词效果。对于分类模型的建立，我们考虑**基于 TFIDF 关键词抽取和最大相似度匹配的无监督分类模型**，最终在验证集上的 F1 得分为 0.56。为了得到更加精准的分类模型，利用**表征学习进行文本词嵌入，结合 fastText 文本分类模型实现了有监督聚类**，最终的验证集 F1 评分为 0.93。该模型的分类效果较好，基本满足分类需求。

针对热点问题挖掘，本文通过建立 **k-means 聚类量化模型**实现了问题热度指数的量化。首先量化留言关注度，将一条留言所有的点赞数和反对数相加作为一个留言关注度量化评分。我们考虑从留言具体内容的角度来研究留言热度。我们利用词频共现算法来获取关键词指数，然后利用文本相关系数构建 k-means 聚类量化模型，文**本热度指数可以根据留言到中心簇的距离公式来量化**。综合考虑点赞数与反对数指标，从而加权归一化得到整体的留言热度指数。最终根据留言热度指数量化结果排序，获取了排名前五的热点问题。进一步利用聚类算法对热点问题进行归类，得到的最终热点问题结果表见正文表 3。其中前五的热点问题中有三条是关于 A 市 58 车贷案，这也说明该问题引起了广泛关注。

针对答复意见的评价问题，本文通过量化相关性，完整性，可解释性来综合量化留言质量。对于相关性，我们利用莱文斯坦相似度计算留言和答复的文本相似性来量化。对于答复意见的完整性指标量化，我们考虑利用文本分词算法，通过文本分词数来衡量。可解释性指标，我们利用字符串匹配结合高频词统计来获取。根据量化的三个指标，我们建立了**熵权综合评估模型，利用 python 编程，最终给出了每个答复的熵权评分作为答复意见质量评分**。最终给出了排名前 10 的留言答复意见结果表，具体见正文表 4。

关键词 TFIDF 算法 fastText BMM 表征学习 熵权法 余弦相似度

一、问题背景与重述

1.1 问题背景

“智慧政务”以现代信息技术（云技术）为基础，通过全面感知、信息交换、流程整合、数据智能处置方式，将社会治理优化，实现公共治理高效精准、公共服务便捷惠民、社会效益显著的一种全新政务运营模式。“智慧政务”主要是由服务、管理、应用平台、资源和智慧平台五个部分组成，通过利用现代先进的信息收集和处理技术来收集和整理城市各阶层人民的基础信息，从而对多种民生问题做出智能处理，实现整座城市对民生问题的智能、精细化处理。各类社情民意相关的数据与智慧民生密不可分息息相关，通过挖掘数据的潜在价值并为我国目前的民生问题提供可靠的决策和建议必将使大数据成为我国社会发展的强力助推器。

对于收集自互联网公开来源的群众问政留言记录，及相关部门对部分群众留言的答复意见，这类数据属于非结构化数据，直接利用非常困难，因此需要使用文本挖掘。为了得到对我们绝对有用的信息，将蕴藏在文本库中的大量信息进行提取，从而得到一些相对隐蔽的、之前从未听说的、对我们有帮助的信息，这一过程我们称之为文本挖掘。对于文本挖掘大体可以分为两大类，一种是对单个文本的挖掘，主要是做文本结构分析、文本摘要、信息表现，主要是挖掘单个文本中有价值的信息，在内容或结构上，主要应用在文本检索领域或搜索引擎；另一种是对文档集的，主要是做文本的分类、聚类，通过提取文本的某些特征对文本进行分类聚类，可以应用在文件的自动管理和垃圾邮件的过滤等方面。

1.2 问题重述

根据文本数据，对于“智慧政务”中的文本数据进行挖掘，建立数学模型，解决以下问题：

1、根据附件 2 的数据建立关于留言内容的一级标签分类模型，以便后续将群众留言分派至相应的职能部门处理；

2、根据附件 3 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，保存为热点问题表和热点问题留言明细表；

3、根据附件 4 相关部门对留言的答复意见，从答复的相关性、完整性、可解释性等角度对答复意见的质量给出一套评价方案。

二、问题分析

2.1 问题一的分析

本题要求针对文本留言内容建立一个一级标签分类模型。从而实现群众留言的自动化分类。我们首先针对附件二多给的文本数据进行定性的分析，包括文本词频统计，词云图绘制等。

为了能够较好的应用分类模型，我们首先针对文本数据进行特征预处理，利用正则替换，*jieba* 分词，去除停止词等手段来清洗数据。进一步利用清洗之后的特征数据进行分类建模。对于分词部分，虽然利用了 *jieba* 智能分词模块，但仍旧有很多比较长的专业词很难完整的划分出来，因此我们利用逆向最大匹配分词算法 *BMM* 借助自建词表实现最佳分词。利用预处理之后的分类特征，我们首先尝试利用关键词提取以及关键词和

标签词的最大相似度匹配来实现文本分类。关键词提取主要采用 *TFIDF* 算法来实现文本关键词抽取。

但是考虑到这种方法是一种无监督的分类算法，准确率可能比较低，因此我们考虑利用表征学习进行词嵌入，进一步利用开源 *Fasttext* 文本分类框架构建分类模型，从而实现有监督训练的文本分类。我们还提出了利用 *word2vec* 对文本进行表征学习，并且构建 *LGB* 最大提升树模型来实现文本分类的有监督训练。并且对比主流模型在测试集评分结果，从而得到较好的分类结果。

2.2 问题二的分析

本题要求针对热点问题进行挖掘，主要目的是从群众留言中挖掘出热点问题。也就是给每一条留言都量化一个热度指数。并且根据热度指数进行排序，从而获取热度较高的评价问题。

对于热度指数的量化，我们通过对附件 3 数据可以发现问题的点赞数与反对数可以在一定程度上反应这个问题的关注度情况。因此问题的点赞数与反对数也是衡量问题热度的一个重要指标。比如问题的点赞数越多，就越说明这个问题反应人民群众的心声。进一步我们考虑从留言具体内容的角度来研究留言热度。首先对文本数据进行预处理，同样包括正则字符处理，*jieba* 分词，然后针对预处理之后留言文本进行词频统计分析。进一步根据词频共现算法来获取关键词指数。根据关键词指数量化文本之间的相关关系，然后根据文本相关系数进行聚类。从而将距离聚类中心簇较远的留言视为热点问题。

文本热度指数可以根据留言到中心簇的距离公式来量化，再综合考虑点赞数与反对数指标，从而加权归一化得到整体的留言热度指数。进一步排序获取最终的结果。

2.3 问题三的分析

本题要求根据部门对于留言的答复意见给出一套意见的质量评价。我们尝试从各种角度来评估答复意见的质量，主要包括从相关性，完整性以及可解释性等角度。

对于答复意见的相关性质量，我们考虑利用文本相似度计算来衡量，通过利用前文的关键词抽取算法，抽取出留言的关键词文本和答复意见文本计算余弦相似度或者是莱文斯坦相似度。其中，莱文斯坦相似度描述的是两端文本之间的形体相似性。最终利用上述相似度计算结果归一化获取答复意见与问题的相关性系数。对于答复意见的完整性评价，我们利用前文的文本分词算法，通过文本词长度统计来衡量，一般来说文本含有的词语越多，回复意见越完整。此外对于答复意见的可解释性评估，主要考虑利用字符串匹配来获取，主要是统计答复意见中高频词出现在问题中高频词的次数进行统计。最终量化出三个评价指标，然后建立熵权综合评价模型，最终给出每个答复意见的质量评价得分。

三、模型的假设

结合本题的实际，为确保模型求解的准确性和合理性，本文排除一些位置因素的干扰，提出以下几点假设：

1、本文假设附件所给数据都是真实有效的数据；

2、本文假设对于利用已有的 *jieba* 中文分词模块可以很好的对中文文本进行处理；

3、本文假设 *F-score* 可以很好的评估分类结果。

四、符号说明

为了便于问题的求解，给出以下符号说明：

符号	说明
w	目标词
D	数据对象
g_j	差异系数
w_j	熵权的大小
h	隐藏层的输出值
$p(w)$	目标词为 w 的概率
C_i	簇中数据的平均值
r_{ij}	原变量和 x_j 的相关系数
$\theta_n(w,j)$	非叶子结点 $n(w,j)$的向量表示
J	数据集中所有数据点和其簇中心的均方差的综合

五、模型的建立与求解

5.1 问题一模型的建立与求解

本题要求针对文本留言内容建立一个一级标签分类模型。从而实现群众留言的自动化分类。我们首先针对附件二给的文本数据进行定性的分析，包括文本词频统计，词云图绘制等。

对于文本标签的分类，我们结合无监督相似性识别和有监督聚类来实现最终的多分类模型构建，从而训练一个更加准确的分类器。

根据上述分析，我们给出了问题一解题思路流程图如下：

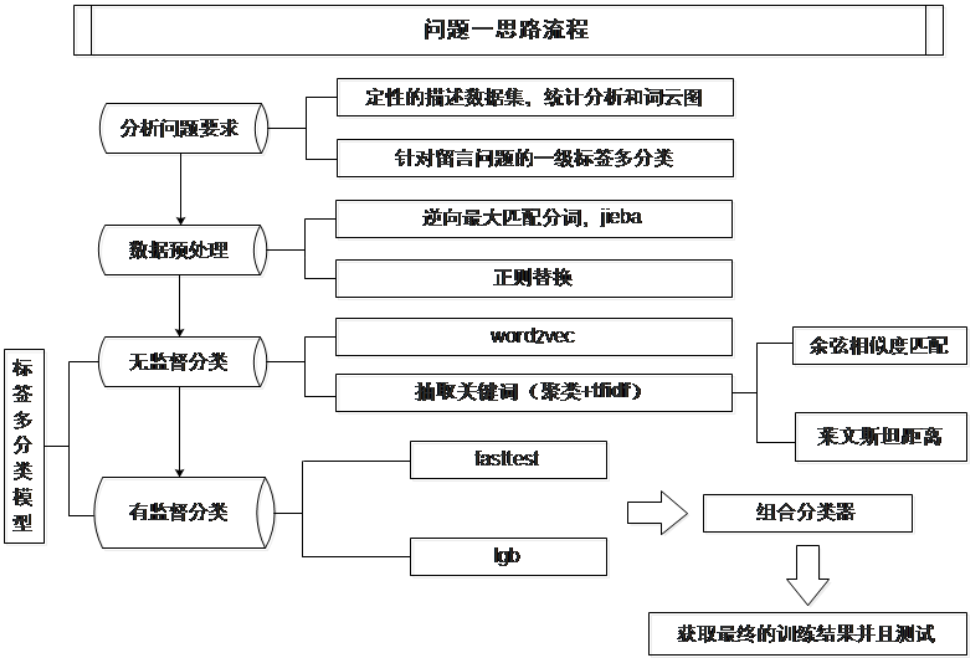

图 1 问题一解题思路流程图

5.1.1 文本数据统计分析与可视化

我们首先针对附件二给的文本数据进行定性的分析，包括文本词频统计，词云图绘制等。针对附件二所给的文本数据集，我们借助 *pandas* 等工具来统计各种标签下留言统计。

得到 7 个标签留言条数统计表如下：

表 1 留言标签统计表

标签名	留言条数
城乡建设	2009
劳动和社会保障	1969
教育文体	1589
商贸旅游	1215
环境保护	938
卫生计生	877
交通运输	613
总计	9210

根据上表内容，我们可以进一步计算出各种标签留言的占比情况。从而绘制以下留言的占比条形图。

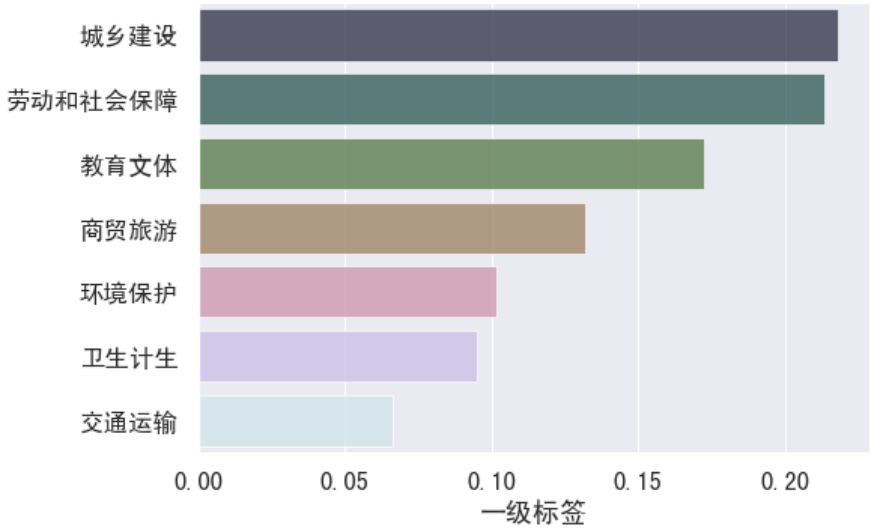

图 2 一级标签留言的占比情况

根据上图，我们可以得到各种标签下的留言的占比很不均衡，这也给分类识别带来了一定的难度。像城乡建设，劳动和社会保障等问题占据了标签数据集接近一半的留言记录。这就会造成分类器更大程度上只能学习到记录数多的标签。为了解决数据不均衡的问题。我们提出了一种数据采样方法，从而扩充一部分训练集标签记录量，这会在一定程度上改进分类器的评测。

我们根据附件二的留言数据进一步探讨，社会关注的热点问题。主要是利用分词，正则处理之后的文本进行词频统计，并且绘制了词云图如下：

图 3 附件二所有文本词云图

根据上述的词云图，我们也可以得到政府留言问题多数是关于城乡建设问题。而且学校，房屋，医院，工作等是群众问政留言记录中的高频词汇。这也从很大程度上说明老板姓真正关心的还是衣食住行等问题。

为了能够较好的应用分类模型，我们首先针对文本数据进行特征预处理，利用正则替换，*jieba* 分词，去除停止词等手段来清洗数据。进一步利用清洗之后的特征数据进行分类建模。

5.1.2 文本数据预处理

对于文本数据的预处理，本文利用 *python* 正则表达式来实现对文本数据的清洗。对于清洗干净的文本数据，我们利用分词算法实现文本分词。文本分词的好坏程度直接决定后续分类识别以及热点问题挖掘等工作。因此我们需要针对文本分词进行较好的处理。

对于分词部分，虽然利用了 *jieba* 智能分词模块，但仍旧有很多比较长的属于词很难完整的划分,因此我们利用逆向最大匹配分词算法 BMM 借助自建词表实现最佳分词。

BMM 是一个基于词典的分词算法，与之相似的还有正向最大匹配算法(FMM)和双向匹配算法(*BM*)。 算法的大致过程为：

*Setp*1：输入最大词长 *maxWordLength*，字典 *wordDict*，待分句子。

*Step*2：从待分句子的末尾开始向前截取长度为 *maxWordLength* 的子句，进行分词。

*Step*3：对一个子句的分词过程为，首先判断子句是否在字典中，若在，则保存这个子句，并从原句中删除这个子句，转到 2。 若不在，则判断子句长度是否为 1，若为 1，则将单字保存，从原句中删除单字，转到 2。若不为 1，则将子句中最右边的一个字删除，形成新的子句，转到 3。

我们在这里给出了使用 *BMM* 算法结合结巴分词和只使用 *jieba* 分词的效果对比。最终的分词对比结果如下：（我们绘制了分词结果的词云图）

测试分词的文本数据如下表：

表 2 附件二文本分词测试数据表

留言编号	留言用户	留言主题	留言时间	留言详情	一级标签
24	A00074011	A 市西湖建筑集团占道施工有安全隐患	2020/1/6 12:09:38	A3 区大道西行便下道，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。	城乡建设

利用上述测试文本，我们测试两种分词算法的效过对比，具体结果如下：

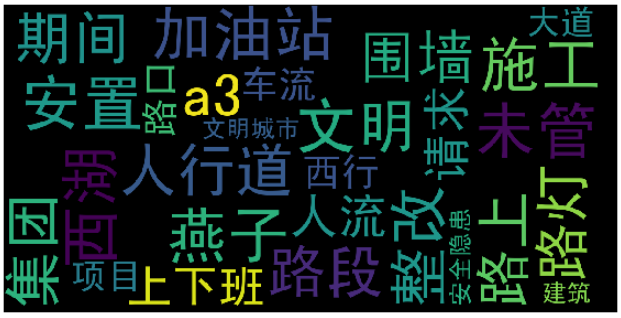

图 4 只使用 *jieba* 分词

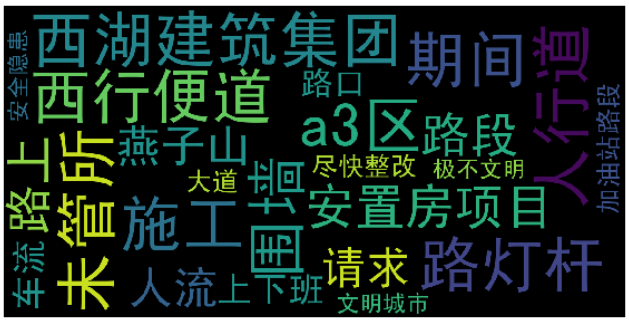

图 5 *jieba* 分词结合 *BMM* 算法

根据上图对比，我们可以得到，只利用 *jieba* 分词，文本分的比较细。有一些长词很难准确的分出来。比如‘未管所’表示未成年管教所，只采用 *jieba* 分词的结果为未管，很难理解它是什么意思。而采用基于词典的 *BMM* 结合结巴分词之后的效过相对较好。很多长词都能够较好的分割出来。

我们利用预处理之后的文本再来研究文本分类，最终的目的是建立一个多标签的文本分类模型。

5.1.3 基于相似度匹配的无监督分类模型

利用预处理之后的分类特征，我们首先尝试利用关键词提取以及关键词和标签词的最大相似度匹配来实现文本分类。关键词提取主要采用 *TFIDF* 算法来实现文本关键词抽取。

我们的目标是针对一段留言文本，训练分词，然后基于 *TFIDF* 算法抽取文本关键词。利用关键词和标签词计算莱文斯坦相似度。

其中 *TF* 表示文本词频统计，也就是一个词在文本中出现次数越多，说明这个词越重要。但是同时也需要考虑逆文档频率 *IDF* 采用逆文档频率 *IDF* 进行加权，可以突出文本的关键词，从而让一条文本的语义特征向量更能表示出该文本的信息。逆文档频率 *IDF* 的计算公式如下：

$$IDF = \log\left(\frac{\text{语料库总数}}{\text{包含该词的文本数量}+1}\right)$$

在信息论和计算机科学中,莱文斯坦距离是一种两个字符串序列的距离度量。 形式化地说，两个单词的莱文斯坦距离是一个单词变成另一个单词要求的最少单个字符编辑数量(如:删除、插入和替换)。 莱文斯坦距离也被称做编辑距离,尽管它只是编辑距离的一种，与成对字符串比对紧密相关。

数学上,两个字符串 *a*、*b* 之间的莱文斯坦距离，*levab*(|*a*|,|*b*|)。如果 *min*(*i*,*j*)= 0，

$$levab(i,j) = max(i,j)$$

否则，

$$levab(i,j) = min\left(levab(i-1,j)+1, levab(i,j-1)+1, levab(i-1,j-1)+1\right)\left(a_i != b_j\right)$$

其中 $a_i != b_j$ 是指示函数，当 $a_i != b_j$ 时为 1，否则为 0。注意在最小项：第一部分对应删除操作(从 *a* 到 *b*)、第二部分对应插入操作、第三部分对应替换操作。

根据上述相似度计算方法，我们利用 *Python* 编程实现文本关键词和 7 个标签词的相似度计算。根据最大相似度原则，我们就可以获取每一条留言对应的标签。根本上来说，该算法是一种无监督的匹配算法。关键词抽取的结果直接决定最终匹配标签的准确度。

根据上述分析，我们得到本文建立的无监督分类模型的思路流程如下：

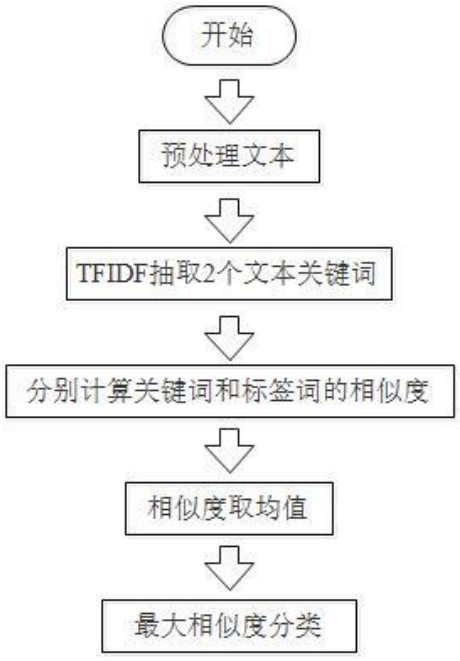

图 6 无监督分类模型的流程图

我们利用 *Python* 语言编程实现上述无监督分类模型的求解，具体代码见附录。我们利用分类标签识别结果和原结果对比，利用 *F-Score* 对分类方法进行评价：

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

使用所有数据进行无监督分类，并且将分类结果和实际结果之间计算 *F* 得分，得到 *F*1 得分为 0.59，效果相对一般的无监督方案要好，但是还没有做到精确的分类识别。因此本文建立一个有监督聚类模型来训练文本分类器，从而追求更加准确的分类效果。

5.1.4 有监督分类模型的训练与测试

考虑到上述方法是一种无监督的分类算法，准确率可能比较低，因此我们考虑利用表征学习进行词嵌入，进一步利用开源 *Fasttext* 文本分类框架构建分类模型，从而实现有监督训练的文本分类。我们还提出了利用 *word2vec* 对文本进行表征学习，并且构建 *LGB* 最大提升树模型来实现文本分类的有监督训练。并且对比主流模型在测试集评分结果，进一步组合分类器从而得到更好的分类识别结果。

fastText 是快速文本分类算法，比基于神经网络的分类算法有着更强是泛化能力。

fastText 模型架构和 *word2vec* 中的 *CBOW* 很相似， 不同之处是 *fastText* 预测标签而 *CBOW* 预测的是中间词，即模型架构类似但是模型的任务不同。下面我们先看一下 *CBOW* 的架构：

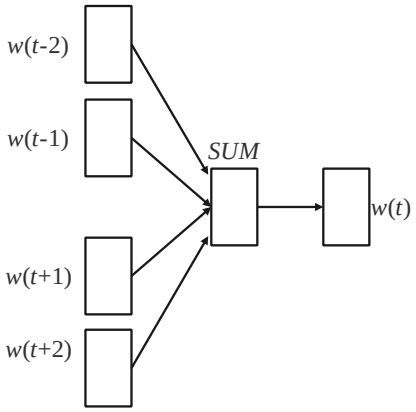

图 7 CBOW 模型结构图

word2vec 将上下文关系转化为多分类任务，进而训练逻辑回归模型，这里的类别数量|V|词库大小。通常的文本数据中，词库少则数万，多则百万，在训练中直接训练多分类逻辑回归并不现实。*word2vec* 中提供了两种针对大规模多分类问题的优化手段，*negative sampling* 和 *hierarchical softmax*。在优化中，*negative sampling* 只更新少量负面类，从而减轻了计算量。*hierarchical softmax* 将词库表示成前缀树，从树根到叶子的路径可以表示为一系列二分类器，一次多分类计算的复杂度从|*V*|降低到了树的高度

fastText 模型架构:其中 *x*1,*x*2,…,*xN*−1,*xN* 表示一个文本中的 *n-gram* 向量，每个特征是词向量的平均值。这和前文中提到的 *cbow* 相似，*cbow* 用上下文去预测中心词，而此处用全部的 *n-gram* 去预测指定类别

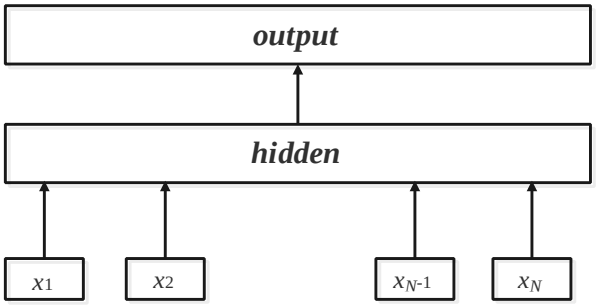

图 8 *fasttext* 模型架构图

softmax 函数常在神经网络输出层充当激活函数，目的就是将输出层的值归一化到 0-1 区间，将神经元输出构造成概率分布，主要就是起到将神经元输出值进行归一化的作用。

在标准的 *softmax* 中，计算一个类别的 *softmax* 概率时，我们需要对所有类别概率做归一化，在这类别很大情况下非常耗时，因此提出了分层 *softmax*(*Hierarchical Softmax*), 思想是根据类别的频率构造霍夫曼树来代替标准 *softmax*，通过分层 *softmax* 可以将复杂度从 *N* 降低到 *logN*，下图给出分层 *so*ftmax 示例：

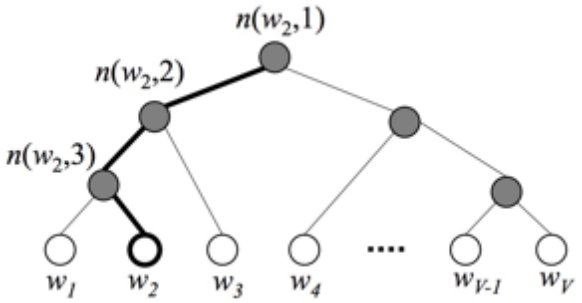

图 9 分层 *softmax* 示例

在层次 *softmax* 模型中，叶子结点的词没有直接输出的向量，而非叶子节点都有响应的输在在模型的训练过程中，通过 *Huffman* 编码，构造了一颗庞大的 *Huffman* 树，同时会给非叶子结点赋予向量。我们要计算的是目标词 *w* 的概率，这个概率的具体含义，是指从 *root* 结点开始随机走，走到目标词 *w* 的概率。因此在途中路过非叶子结点（包括 *root*）时，需要分别知道往左走和往右走的概率。例如到达非叶子节点 *n* 的时候往左边走和往右边走的概率分别是：

$$p(n,left)=\sigma\left(\theta_n^T\cdot h\right)$$

$$p(n,right)=1-\sigma\left(\theta_n^T\cdot h\right)=\sigma\left(-\theta_n^T\cdot h\right)$$

以上图中目标词为 *w*2 为例，

$$\begin{aligned}p(w_2)&=p\left(n(w_2,1),left\right)\cdot p\left(n(w_2,2),left\right)\cdot p\left(n(w_2,3),right\right)\\&=\sigma\left(\theta_n(w_2,1)^T\cdot h\right)\cdot\sigma\left(\theta_n(w_2,2)^T\cdot h\right)\cdot\sigma\left(-\theta_n(w_2,3)^T\cdot h\right)\end{aligned}$$

到这里可以看出目标词为 *w* 的概率可以表示为：

$$p(w)=\prod_{j=1}^{L(w)-1}\sigma\left(sign(w,j)\cdot\theta_n(w,j)^T\,h\right)$$

其中 $\theta n(w,j)$是非叶子结点 $n(w,j)$的向量表示（即输出向量）；*h* 是隐藏层的输出值，从输入词的向量中计算得来；*sign*(*x*,*j*)是一个特殊函数定义

$$sign(w,j)=\begin{cases}1, & 若n(w,j+1)是n(w,j)的左孩子\\-1 & 若n(w,j+1)是n(w,j)的右孩子\end{cases}$$

此外，所有词的概率和为 1，即

$$\sum_{i=1}^{n}p(w_i)=1$$

最终得到参数更新公式为：

$$\theta_j^{(new)} = \theta_j^{(old)} - n\left(\sigma\left(\theta_j^T h\right) - t_j\right)h$$

其中， $j = 1,2,\ldots,L(w)-1$。

我们利用预处理之后的文本数据进行分类，利用 *Python* 语言实现 *fasttest* 模型。最终训练了文本分类器。对于文本分类器的训练，我们将全部的数据集划分为训练集和验证集从而测试模型分类效果（8:2）。最终分类器在验证集上述的评测 F1 得分为 0.92。为了进一步验证模型的有效性。我们对比不同模型的 *AUC* 曲线得分，对比结果如下：

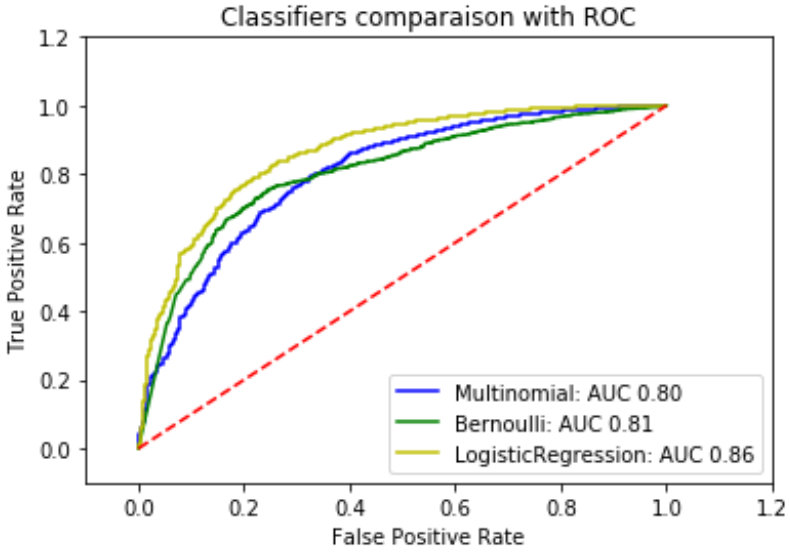

图 10 *AUC* 曲线得分

根据上图，我们也可以得到 *fasttext* 模型要好于大部分的常规机器学习模型。

5.2 问题二模型的建立与求解

本题要求针对热点问题进行挖掘，主要目的是从群众留言中挖掘出热点问题。也就是给每一条留言都量化一个热度指数。并且根据热度指数进行排序，从而获取热度较高的评价问题。对于问题热度指数的量化，我们主要从问题关注数量，以及留言具体内容两个方面来考虑。

5.2.1 留言关注度量化分析

对于热度指数的量化，我们通过对附件三数据可以发现问题点赞数与反对数可以在一定程度上反应这个问题的关注度情况。因此问题点赞数与反对数也是衡量问题热度的一个重要指标。比如问题点赞数量越多，就越说明这个问题反应人民群众的心声。

我们首先针对所有留言进行一些统计分析，主要是针对留言关注数进行统计分析。我们将一条留言所有的点赞数和反对数相加作为一个留言关注度量化评分。最终我们对附件三所有留言进行留言关注度计算，并且根据关注度指标进行量化排序。最终我们筛选出来关注度排名前 20 的留言。具体关注度排序可视化条形图如下：

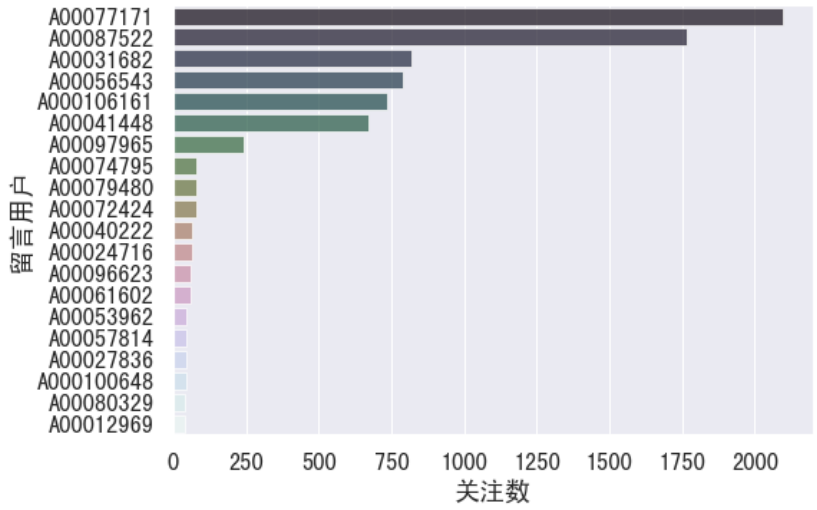

图 11 关注度排名前 20 的留言

根据上图，我们可以清楚得到哪些留言的关注度比较高。根据上图，我们可以清楚的得到排名前六的关注度留言与其他群众留言关注人数相差较大。说明这些问题更受到广大人名群众的关心。

为了进一步针对各个留言的具体内容来量化留言的热度指数，我们需要从留言的内容出发，进一步建立一个留言热度指数量化模型来获取最终的热点问题。

5.2.1 基于相关性聚类模型的留言热度指数计

本文进一步构建一个留言热度指数量化模型，考虑从留言具体内容的角度来研究留言热度。首先对文本数据进行预处理，同样包括正则字符处理，*jieba* 分词，然后针对预处理之后留言文本进行词频统计分析。进一步根据词频共现算法来获取关键词指数。根据关键词指数量化文本之间的相关关系，然后根据文本相关系数进行聚类。从而将距离聚类中心簇较远的留言视为热点问题。文本热度指数可以根据留言到中心簇的距离公式来量化，再综合考虑点赞数与反对数指标，从而加权归一化得到整体的留言热度指数。进一步排序获取最终的结果。

我们首先利用词频共现算法来获取一篇论文的关键词与论文之间的相关指数。相关指数的求解。我们通过计算词和留言之间的共线统计结果，可以按照以下公式来量化关联系数：

$$R = \log(\frac{count_{st}+1}{count_{s}+1})$$

然后利用每条留言的关键词指数量化文本之间的相关关系。根据数据得到相关系数矩阵，计算它们的相关系数矩阵：

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} \end{pmatrix}$$

其中 $r_{ij}\left(i, j = 1, 2, ..., p\right)$ 表示是原变量与 x_j 的相关系数，并且 $X_{ij} = X_{ji}$。相关系数的计算公式为：

$$r_{ij}=\frac{\sum_{k=2}^{n}\left(x_{ik}-\overline{x}_i\right)\left(x_{kj}-\overline{x}_j\right)}{\sqrt{\sum_{k=1}^{n}\left(x_{ik}-\overline{x}_i\right)^2\sum_{k=1}^{n}\left(x_{kj}-\overline{x}_j\right)^2}}$$

利用上述相关性矩阵，我们可以建立一个 *K-Means* 聚类模型来获取热点问题。由于大部分问题都是非热点问题。因此我们可以统计远离聚类中心簇的问题作为热点问题。并且结合热点问题到中心簇距离来量化热点问题的文本热度指数。对于利用相关性矩阵进行聚类，我们可以考虑一些常用的聚类算法。

具体的常用聚类算法见下图：

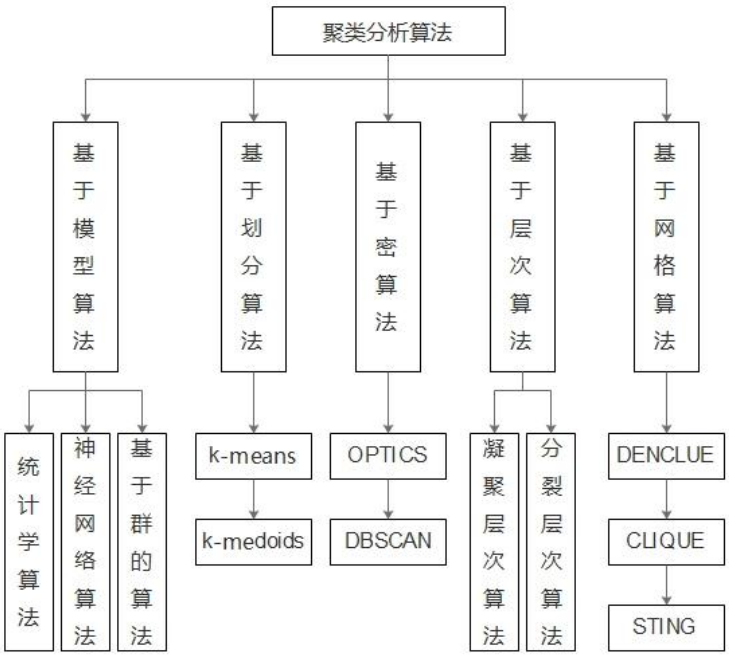

图 12　常用聚类算法

考虑到我们的主要目的是按照相关系数划分文本，从而获取热点问题，因此我们采用基于划分的聚类算法。

本文采用 *K-means* 算法作为本题事件分类的基础，该算法的基本原理如下：

K-means（k 均值）算法接受一个参数 k 用以决定结果中簇的数目。算法开始时，要在数据集中随机选择 k 个数据对象，用来当做 k 个簇的初始中心。

考虑到我们的主要目的是按照州来划分加拿大的空间温度分布等级，因此我们选择基于划分的算法 *K-means*。

本文采用 *K-means* 算法作为本题事件分类的基础，该算法的基本原理如下：

K-means（k 均值）算法接受一个参数 k 用以决定结果中簇的数目。算法开始时，要在数据集中随机选择 k 个数据对象，用来当做 k 个簇的初始中心。

通过计算剩下的各个点与每个簇类中心的距离，并将其分配至距离最近的簇中；计算每个簇中数据的平均值，并将其作为新的质心；重复上述步骤，直到目标函数收敛。

本文的目标函数选用均方差函数，其公式定义如下：

$$J=\sum_{i=1}^{k}\sum\left|D-m_i\right|^2,\ D\in C_i$$

J：数据集中所有数据点和其簇中心的均方差的综合

D：数据对象

C_i：簇中数据的平均值

聚类迭代停止的条件为均方差达到最小或者不变。

在确定好预期的聚类簇的 *k* 个中心之后，*k* 代表每个簇的种子；确定了初始的种子之后，每个簇都为空；然后计算每个点与种子之间的距离，并将其放入距离最近的簇中。在聚类方法中，常用的距离度量有欧几里得距离，余弦相似度等，它们之间的区别如下图所示：

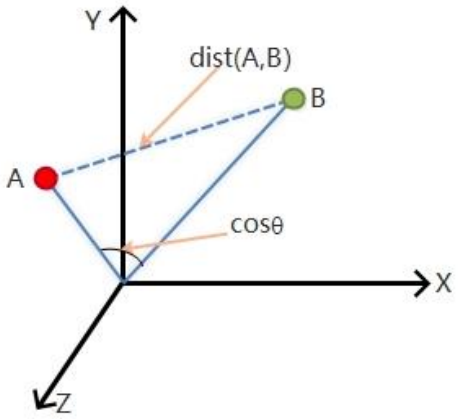

图 13 欧几里得距离和余弦相似度

衡量数据点之间的距离较常用的是欧氏距离，本文采用欧氏距离来衡量两变量之间的相似性，公式定义如下：

$$d\left(x_i, x_j\right)=\sqrt{\sum_{k=1}^{d}\left(x_{ik}-x_{jk}\right)^2}$$

经过一次聚类后，需要重新计算簇中心，然后计算距离重新划分数据点所属簇，过程中可能会出现原不属于该簇的数据被划分到了簇中，此次聚类可以看作对上一次聚类结果的校正，使得数据点划分到更合理的簇中，从而达到簇内数据点之间的距离最小，而簇之间的距离最大。

K-Means 算法的基本过程如下：

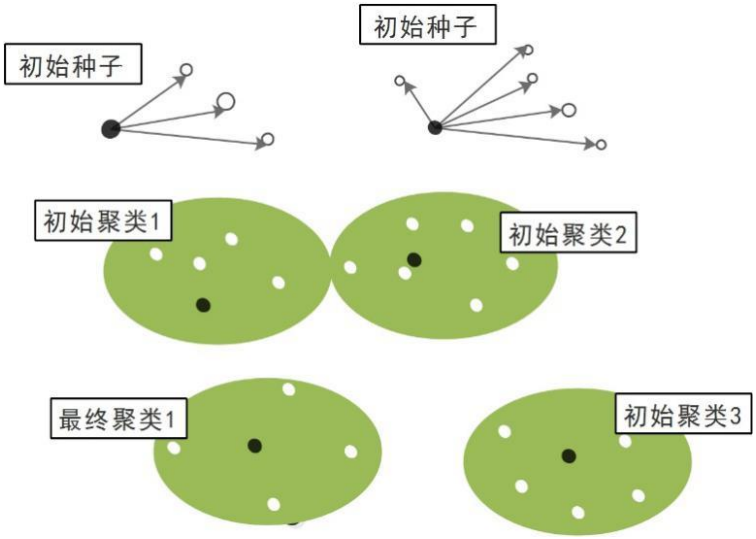

图 14　聚类示意图

K-means 是经典的聚类算法之一，简单，分类速度快，可伸缩性优。对于大量的数据集，在保证分类效果的情况下，具有较高的效率。

考虑到本文中所用到的数据集较大和 *K-means* 的明显的优势，本文采用 *K-means* 算法实现聚类分析。

首先，对数据集进行初步的变量筛选和特征提取，从而降低聚类数据集的维度，从而减小计算复杂度。本文采用主成分分析（*PCA*）降维，进而对降维后的特征变量进行聚类分析。

K-means 算法步骤如下：

输入： $n*m$ 的数据集，聚类结果 k 个簇，依据肘部法则确定最优分类数 k，随着 k 值增大，畸变程度（该类中心与其内部数据点距离的平方和）下降幅度最大对应的 k 值即肘部。

输出：满足准则函数的 k 个类。

聚类迭代过程如下：

Step 1：在数据集里随机选择 k 个簇心，将每个质心代表初始聚类的中心；

Step 2：将剩下的数据点划分到和其距离最近的簇心所在簇中；

Step 3：计算每个簇的均值得到新的簇心；

Step 4：重复 *Step* 2 和 *Step* 3，直到目标函数收敛或每个簇中数据不再发生变化。

最终的聚类实现算法流程如下：

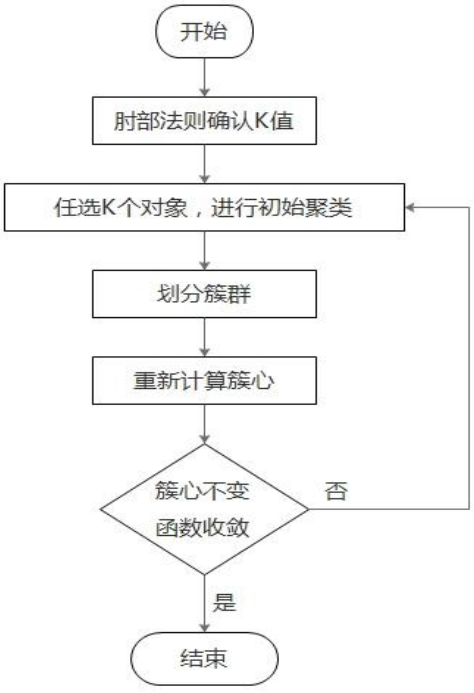

图 15 *K-Means* 聚类算法实现流程

根据上述的算法流程，我们首先利用 *PCA* 算法对原来的文本相关性矩阵进行压缩降维。得到降维碎石图如下：

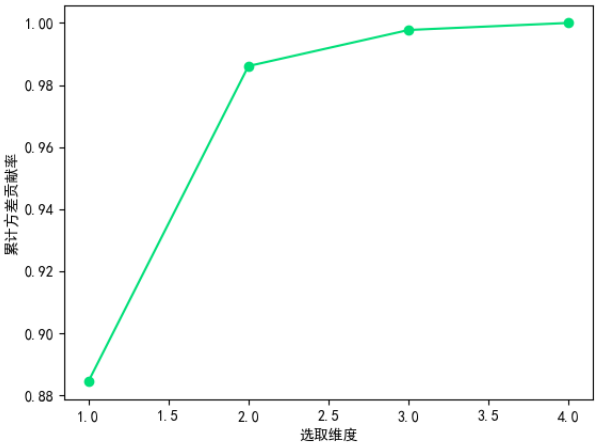

图 16 *PCA* 降维碎石图

根据上述碎石图，我们最终选取了两维特征变量进行聚类建模。然后根据两维特征变量，利用 *K-Means* 聚类算法，根据肘部法则确定出合理的 *k* 值为 4。得到最终的聚类效果图如下：

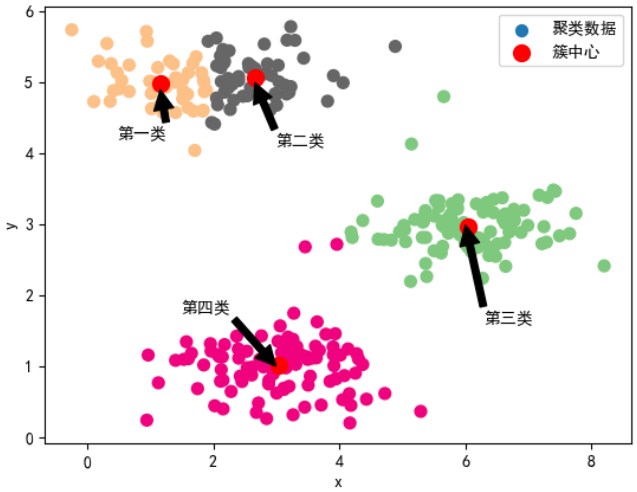

图 17　*K-Means* 聚类可视化效果图

根据上述的聚类模型，我们最终可以获取文本热度指数，然后经过文本热度指数排序，最终可以得到排名前 20 的文本热点留言。最终可以得到排名前 20 的文本热点留言如下：

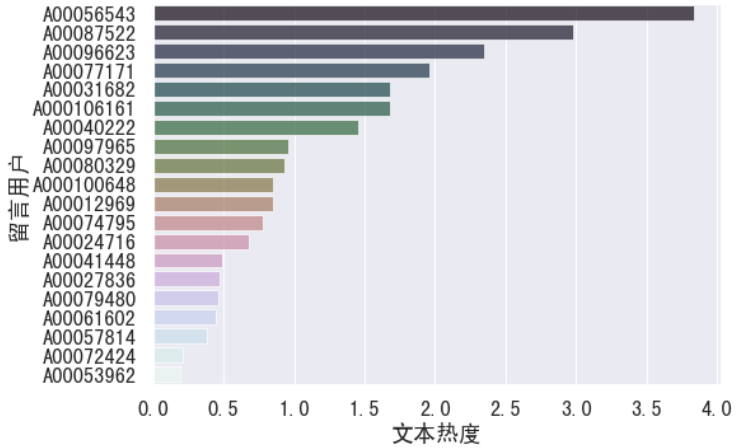

图 18　文本热度指数

而对于留言的最终热度指数量化，我们主要是利用加权评估量化法来计算：

$$\omega = w_1 \times \frac{\sum_{i=1}^{n}(y_i - \dot{y})^2}{n} + w_2 \times \sum_{i=1}^{n}(y_i - y_0)^2$$

根据上述公式，我们可以最终得到所有留言的热度指数量化结果，最终筛选了排名前五的热点问题表结果表如下：

表 3　前五的热点问题结果表

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	0.93	2019-08-19 11:34:04	A 市 A5 区汇金路	A 市 A5 区汇金路五矿万境 K9 县存在一系列问题
2	2	0.87	2019-02-25 09:58:37	几十位 58 车贷受害者	严惩 A 市 58 车贷特大集资诈骗案保护伞
3	3	0.81	2019-04-11 21:02:44	金毛湾 2800 户业主	反映 A 市金毛湾配套入学的问题
4	4	0.73	2019-02-21 18:45:14	A 市 A4 区	请书记关注 A 市 A4 区 58 车贷案
5	5	0.69	2019-03-01 22:12:30	58 车贷案	承办 A 市 58 车贷案警官应跟进关注留言

为了更加清晰的分析结果，我们给出了排名前五的留言热度指数条形图：

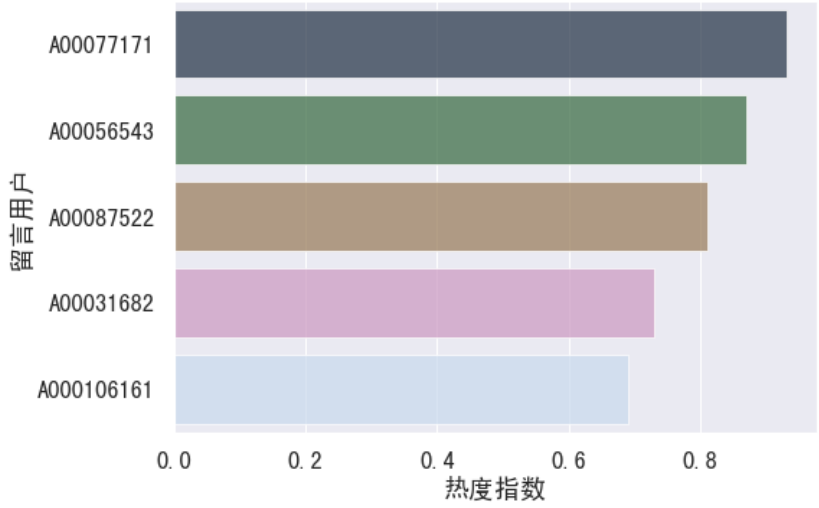

图 19 排名前五的留言热度指数

根据上图，我们可以得到最终的留言热度指数和文本热度以及留言关注数都有着强相关性。这主要是因为排名前五的留言热度指数也都出现在文本热度以及留言关注数排名前 10 的结果中。

为了验证上述结论，我们给出了部分量化指标的相关热力图。相关热力图如下：

图 20 问题二建模量化指标的相关热力图

根据上图，我们显然可以得到影响留言热度指数的指标主要是点赞数，关注数，以及文本热度。

5.3 问题三模型的建立与求解

本题要求根据部门对于留言的答复意见给出一套意见的质量评价。我们尝试从各种角度来评估答复意见的质量，主要包括从相关性，完整性以及可解释性等角度。

5.3.1 各种指标量化数据的计算

对于答复意见的相关性质量，我们考虑利用文本相似度计算来衡量，通过利用前文的关键词抽取算法，抽取出留言的关键词文本和答复意见文本计算余弦相似度或者是莱文斯坦相似度。

其中，莱文斯坦相似度描述的是两端文本之间的形体相似性。最终利用上述相似度计算结果归一化获取答复意见与问题的相关性系数。利用前文的 TFIDF 算法抽取文本关键词，然后利用留言内容文本与答复意见文本抽取词计算文本相似度。为了更加准确的计算出文本相似度，我们针对文本分词，同样采用逆向最大匹配 BMM 结合 jieba 智能分词来完成。预处理分词之后，利用词嵌入加权获取文本嵌入，然后根据余弦相似度计算公式来获取留言文本和答复意见文本的相似度。

$$\cos(\theta)=\frac{\sum_{i=1}^{n}(x_i\times y_i)}{\sqrt{\sum_{i=1}^{n}(x_i)^2}\times\sqrt{\sum_{i=1}^{n}(y_i)^2}}=\frac{a\bullet b}{\|a\|\times\|b\|}$$

量化的文本相似度就可以看作是答复意见的相关性质量指标。对于答复意见的完整性指标量化衡量，我们考虑利用前文的文本分词算法，通过文本词长度统计来衡量，一般来说文本含有的词语越多，回复意见越完整。也就是利用答复意见的文本分词数量作为完整性量化指标。

而对于答复意见的可解释性评估，主要考虑利用字符串匹配来获取，主要是统计答复意见中高频词出现在问题中高频词的次数进行统计。根据上述量化公式，我们最终可以得到针对答复意见的三个量化结果。分别是相关性质量，完整性质量以及可解释评估性质量。

最终量化出三个评价指标，然后建立熵权综合评价模型，最终给出每个答复意见的

质量评价得分。

5.3.2 熵权评估模型的建立

对于一个多指标的综合评价问题，首先就要确定各个指标在综合评价过程中的权重系数，而主观赋权法存在主观性较强的缺陷，往往使得计算结果的有效性、科学性降低。因此我们需要在专家经验基础上选择模型所需的指标后，用更客观的方法进行权重的分配，避免人为因素造成的误差，充分做到让数据说话。考虑到客观权重求解的简便性以及准确性，本文采用熵权法来进行权重的确定。

按照信息熵的思想，根据熵的特性，我们可以通过计算熵值来判断一个事件的随机性及无序程度，也可以用熵值来判断某个指标的离散程度，指标的离散程度越大，该指标对综合评价的影响越大。

熵权法的一般步骤如下：

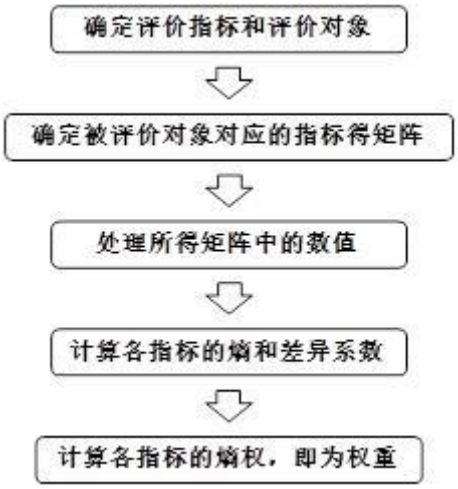

图 21 熵权算法的一般步骤

本文主要采用的数据标准化方法是 $Z-score$ 标准化方法，一般来说没这种方法较为简便，而且标准化效果也较好。这种方法基于原始数据的均值（*mean*）和标准差（*standard deviation*）进行数据的标准化。将 A 的原始值 x 使用 $Z-score$ 标准化到 x'。$Z-score$ 标准化方法适用于属性 A 的最大值和最小值未知的情况，或有超出取值范围的离群数据的情况：

$$\text{新数据}=(\text{原数据}-\text{均值})/\text{标准差}$$

其中， *SPSS* 软件默认的标准化方法就是 $Z-score$ 标准化。

$$p_{ij}=X_{ij}\Big/\sum_{i=1}^{n}X_{ij}\ \ (i=1,2...,n,\ j=1,2...,m)$$

计算第 j 项指标的熵值。

$$e_j=-k\sum_{i=1}^{n}p_{ij}\ln(p_{ij})$$

其中， $k>0$， $k=1/\ln(n)$， $e_j\geq 0$ 。

计算第 j 项指标的差异系数。对第 j 项指标，指标值的差异越大，对方案评价的左右就越大，熵值就越小，定义差异系数： $g_j=1-e_j/m-E_e$ 。式中：

$$E_e = \sum_{j=1}^{m} e_j \ , \quad 0 \le g_i \le 1 , \quad \sum_{j=1}^{m} w_j = 1$$

最后一步就是求解选取的几个特征指标的熵权大小如下：

$$w_j = g_j \Big/ \sum_{j=1}^{m} g_j , \ 1 \le j \le m$$

根据上述熵权算法流程，我们利用三个数据集中销量最高的产品，研究产品熵权重随着时间的变化趋势。

根据上述模型，我们最终可以计算得到针对每一条回复意见的熵权评分。

我们给出了排名前 10 的留言答复意见，如下表：

表 4 问题三最终的留言答复意见表

留言编号	留言用户	相关性质量	完整性质量(分词数)	可解释评估性	意见质量
3704	UU0081480	0.84	15	6	1.63
4192	UU008524	0.88	17	4	1.61
4394	UU0082398	0.76	26	8	1.54
96283	UU0082127	0.91	13	3	1.53
107063	UU0086	0.81	19	2	1.51
94611	UU0081251	0.73	29	4	1.43
90081	UU008411	0.59	38	5	1.41
179601	UU008609	0.87	17	4	1.38
159027	A00052336	0.64	27	7	1.27
139942	UU008419	0.82	11	5	1.24

根据上表，我们进一步针对上表数据进行可视化，得到分析结果图如下：

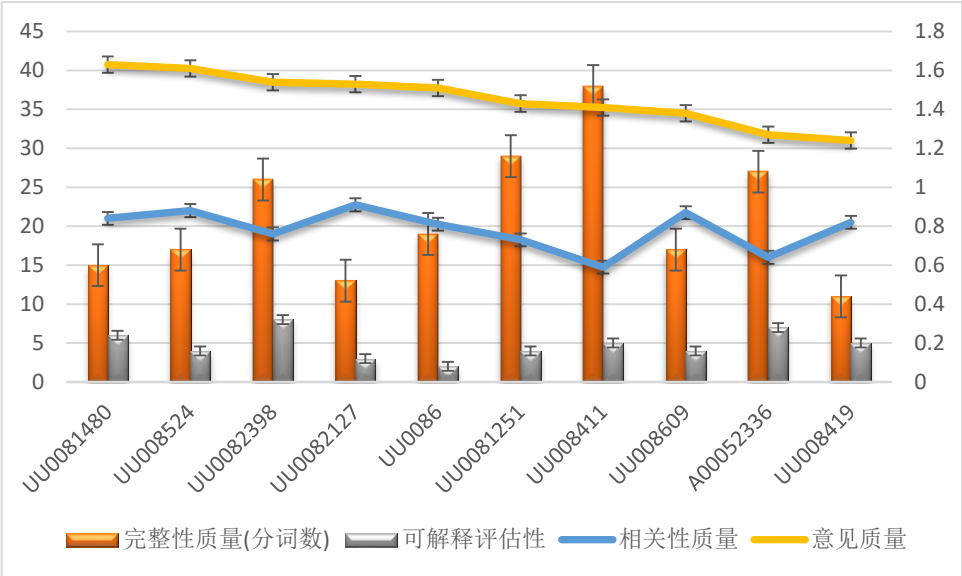

图 22 回复质量评估可视化

根据以上图和表我们可以得到利用熵权法对指标进行综合评估可以很好的确定出各个留言的评分状况。答复意见的评估质量水平量化得分和相关性，完整性以及可解释性等都有较强的关联关系。

六、模型的优缺点与改进

7.1 模型的优缺点

7.1.1 模型的优点

1、*fastText* 在保持高精度的情况下加快了训练速度和测试速度；
2、*fastText* 不需要预训练好的词向量，fas*tText* 会自己训练词向量；
3、*fastText* 两个重要的优化：*Hierarchical Softmax*、*N-gram*。

7.1.2 模型的缺点

1、对问题二的问题归类，没有做到较好的自动化归类，需要一定的人工评估；
2、对问题三的可解释性评估量化，我们利用词频统计，很难做到较好的量化。

7.2 模型的改进

对于聚类分析算法我们可以提出以下改进方案，具体的方案实现，这里不再进行讨论。

本文针对 *K-means* 聚类算法对初始值的选取依赖性极大以及算法常陷入局部极小解得缺陷提出该种聚类算法的改进方案。

K-means 聚类算法首先随机地选取 *k* 个点作为初始聚类中心，再利用迭代的重定位技术寻找最优聚类中心，直到算法收敛。因此初始值的不同可能导致算法聚类效果的不稳定。 *K-means* 聚类算法对初始聚类中心有严重的依赖性。

目前对于聚类中心的选取由以下几种方法：

1、任意的选择 *k* 个样本作为初始聚类中心，*Forgy* 最早提出了这种方法所以也有文献把随机选择初始聚类中心的方法称为 *FA(ForgyApproach)*；
2、凭经验选取有代表性的点作为初始聚类种子。根据个体性质，考察数据结构，筛选出比较合适的点；
3、把全部混合样本直观地分成 *k* 类，计算各均值作为初始聚类中心；

除了以上的选取方法意外，另外还有一种扩展的聚类中心选取方法。这中选取方与上述方法有一个很大的区别，即由原点的点延伸到一条线段，这种选取方法在类之间有干扰点时效果较好。

八、模型的推广

本文应用的 *K-Means* 的目标就是在相似的基础上收集数据分类。聚类分析的结果可以揭示出数据间的差别，发现内在联系，此外，还为更深层次的数据分析与知识发现提供了可靠的依据，如数据间的关联规则、分类模式以及数据的变化趋势等。从实际应用的角度看，聚类分析是数据挖掘的主要任务之一，还可以作为其他算法的预处理步骤。聚类分析具有普遍意义，应用于很多领域，如图像处理、生物学、信息检索、数据挖掘、音频分析检索和统计学等。

TF-IDF 是一种用于信息检索与数据挖掘的常用加权技术。*TF* 是词频，*IDF* 是逆文本频率指数。权重计算方法经常会和余弦相似度一同使用于向量空间模型中，用以判断两份文件之间的相似性。*TF-IDF* 采用文本逆频率 *IDF* 对 *TF* 值加权取权值大的作为关键词，但 *IDF* 的简单结构并不能有效地反映单词的重要程度和特征词的分布情况，使其无法很好地完成对权值调整的功能，所以 *TF-IDF* 算法的精度并不是很高，尤其是当文本集已经分类的情况下。*TF-IDF* 应用于很多领域包括搜索引擎、关键词提取、文本相似性、文本摘要等。

LightGBM 是微软开源的集成算法，在精度不低于 *XGBoost* 的情况下，内存占用，训练速度方面特别友好。*LightGBM* 提出的主要原因就是为了解决 *GBDT* 在海量数据遇到的问题，让 *GBDT* 可以更好更快地用于工业实践。*LightGBM* 模型的参数选择，也很重要，需要了解算法原理才能更好的进行调参。主要有更快的训练效率、低内存使用、更好的准确率、支持并行学习、可处理大规模数据等优势。*LightGBM* 优化部分包含以下基于 *Histogram* 的决策树算法、带深度限制的 *Leaf-wise* 的叶子生长策略、直方图做差加速、直接支持类别特征、*Cache* 命中率优化、基于直方图的稀疏特征优化、多线程优化等。

九、参考文献

[1]郭梅. 智慧政务信息资源共享系统建设研究[D].燕山大学,2015；
[2]郭建永. 聚类分析在文本挖掘中的应用与研究[D].江南大学,2008；
[3]谌志群,张国煊.文本挖掘与中文文本挖掘模型研究[J].情报科学,2007(07):1046-1051；
[4]李荣陆. 文本分类及其相关技术研究[D].复旦大学,2005；
[5]何清. 机器学习与文本挖掘若干算法研究[D].中国科学院研究生院（计算技术研究所）,2002；
[6]胡小娟. 基于特征选择的文本分类方法研究[D].吉林大学,2018；
[7]张勇. 基于词性与 LDA 主题模型的文本分类技术研究[D].安徽大学,2016；
[8]樊小超. 基于机器学习的中文文本主题分类及情感分类研究[D].南京理工大学,2014；
[9]杨杰明. 文本分类中文本表示模型和特征选择算法研究[D].吉林大学,2013；
[10]高扬. 基于 LDA 主题模型的 TFIDF 算法改进及应用[D].广西大学,2015；
[11]王芳杰,王福建,王雨晨,边驰.基于 LightGBM 算法的公交行程时间预测[J].交通运输系统工程与信息,2019,19(02):116-121；
[12]马晓君,沙靖岚,牛雪琪.基于 LightGBM 算法的 P2P 项目信用评级模型的设计及应用[J].数量经济技术经济研究,2018,35(05):144-160；
[13]孙鸿飞,侯伟.改进 TFIDF 算法在潜在合作关系挖掘中的应用研究[J].现代图书情报技术,2014(10):84-92；
[14]肖根胜. 改进 TFIDF 和谱分割的关键词自动抽取方法研究[D].华中师范大学,2012；
[15]施聪莺,徐朝军,杨晓江.TFIDF 算法研究综述[J].计算机应用,2009,29(S1):167-170+180。

附录

附录 *I*

1.1 问题 1 无监督分类实现

```
'''提取关键词脚本，最大相似度匹配'''
from jieba import analyse
import math
import jieba
import pandas as pd
import sys,codecs
import re
from nltk.tokenize import word_tokenize
from utils import data_utils
from collections import defaultdict as dd

class zh_process_func:
    def split_func(x):
        return jieba.lcut(x) # list(x)
    def merge_func(x):
        return ''.join(x)
    def pre_process(x):
        return x.strip().lower()

'''利用词表分词的中文分词模块'''
class tokenizer:
    def __init__(self, dic, process_func):
        self.dic = dic
        self.split_func = process_func.split_func
        self.merge_func = process_func.merge_func
        self.pre_process = process_func.pre_process
        self.max_len = max([len(x) for x in self.dic])

    def tokenize_greedy(self, text, max_len=None, oov=False):
        '''Locally longest word first, also good when doing from backward to frontward'''
        if max_len is None:
            max_len = self.max_len
        text = self.pre_process(text)
        text = self.split_func(text)
        inds = []
        now = ''
        for w in text:
            inds.append(len(now))
            now = self.merge_func([now, w])
        text = text[::-1]
        inds = inds[::-1]
        words = []
        L = len(text)
        i = 0
```

```
            while i < L:
                flag = -1
                for j in range(min(L, i + max_len), i, -1):
                    if self.merge_func(text[i:j][::-1]) in self.dic:
                        flag = j
                        break
                if flag != -1:
                    words.append((self.merge_func(text[i:flag][::-1]), inds[flag-1]))
                    i = flag
                else:
                    if oov: words.append((text[i], inds[i]))
                    i += 1
            return words[::-1]



    ## 数据预处理操作，正则去除一些符号
    def precess_feature(name):
        name = str(name).strip().lower()
    #         name = re.sub('[!“”"#$%&\'()*+,-./:：《》（）;<=>?@[\\]^_`{|}~—～’]+',' ', name)
        name = re.sub('[^\u4E00-\u9FFF]',' ', name)
        name = re.sub( r'\s{2,}', ' ', name).strip()
        return name

    ## 利用词表分词，去停用词，保留长度大于 2 的词
    def dataPrepos(text, stopkey):
        l = []
        zh_tokenizer = tokenizer({'油烟污染','环境保护','环境污染','江东路','污染物','无人施工',
                                  '蓝波旺酒店','人体细胞','脂肪氧化物','劳动和社会保障',
                                  '城乡建设','教育文体','卫生计生','交通运输','商贸旅游','环境保护'}, zh_process_func)
        word_split = zh_tokenizer.tokenize_greedy(text,oov=True)
        word_ = list([word[0] for word in word_split if len(word[0])>=2])
        for i in word_:
            if i not in stopkey:  # 去停用词
                l.append(i)
        return l

    ## 文本预处理函数
    def prepro_data(data):
        stopkey = [w.strip() for w in codecs.open('data/stop_word.txt', 'r').readlines()]
        data['特征'] = data.loc[:,'留言主题']+' '+data.loc[:,'留言详情'].map(lambda x:x.strip().lower())
        data=data.dropna()      #去空行处理
        idList, titleList, abstractList,label = data['留言编号'], data['留言主题'], data['特
```

```
征'],data['一级分类']

        corpus = {} # 将所有文档输出到一个 list 中，一行就是一个文档
        for index in range(len(idList)):
            if idList[index] == 100611:
                print(titleList[index],label[index])
            text = '%s。%s。%s' % (titleList[index], abstractList[index],label[index]) #
拼接标题和摘要
            text = precess_feature(text)
            text = dataPrepos(text,stopkey) # 文本预处理
            text = " ".join(text) # 连接成字符串，空格分隔
            corpus[idList[index]] = text
        return corpus



    '''计算逆文档频率 idf = log（总文档数量/包含这个词的文档数量+1）'''
    def idf_count(data_content):
        idf_dic = {}
        doc_count    = len(data_content) # 总共有多少条文本记录
        for i in data_content:
            new_content = data_content[i].split(' ')
            for word in set(new_content):
                if len(word) > 1:
                    idf_dic[word] = idf_dic.get(word, 0.0) + 1.0

        # 此时 idf_dic 的 v 值：有多少篇文档有这个词
        for k,v in idf_dic.items():
            w = k
            p = '%.10f' % (math.log(doc_count / (1.0 + v))) # 结合上面的 tf-idf 算法
公式
            if w > u'\u4e00' and w<=u'\u9fa5':
                idf_dic[w] = p
        data_utils.dump_data(idf_dic, 'data', "feature_idf.pkl")

    '''计算 tf-idf 提取出关键词：tf-idf = (词频/文档中词的个数)*idf'''
    def tf_idf_count(data,top=10):
        idf_dic =    data_utils.load_data( 'data', "feature_idf.pkl")
        result = {}
        for i,doc in enumerate(data):
            word_split = data[doc].split(' ')
            counter = dd(int)
            for word in word_split:
                if len(word)>1:
                    counter[word]+=1
            tf_idf ={}
            for word in counter:
    #                 if tf_idf[word] is None:
```

```
                    tf_idf[word] = float(((counter[word])/len(word_split)))*float(idf_dic[word])
                tfidf_result = sorted(tf_idf.items(),key=lambda e:e[1],reverse = True)[0:top]
        #             print(doc,tfidf_result)
                result[doc] = tfidf_result
            print(result[100611])
            data_utils.dump_data(result, 'keyword_result', "keyword_tfidf.pkl")


        if __name__ == '__main__':
            # 读取数据集
            data = pd.read_excel('data/附件 2.xlsx',encoding = 'utf_8')
            corpus = prepro_data(data)
            print(corpus[100611]) ##打印其中一段文本分词结果

            idf_count(corpus)
            print(len(corpus))
        tf_idf_count(corpus,10)
```

1.2 问题 1 fasttext 有监督分类器训练

```
        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import StratifiedShuffleSplit
        from sklearn.preprocessing import OneHotEncoder
        import random
        import jieba
        import re
        import fasttext


        def strip_str(df):
            df = df.strip().lower()
            return df

        def writeData(sentences,fileName):
            out=open(fileName,'w',encoding='utf-8')
            for sentence in sentences:
                out.write(sentence+"\n")

        def stopwordslist(filepath):
            stopwords = [line.strip() for line in open(filepath, 'r', encoding='utf-8').readlines()]
            return stopwords

        def precess_feature(name):
            name = str(name).strip().lower()
        #         name = re.sub('[!“”"#$%&\'()*+,-./:：《》（）;<=>?@[\\]^_`{|}~—～’ ]+',' ', name)
```

```
        name = re.sub('[^\u4E00-\u9FFF]',' ', name)
        name = re.sub( r'\s{2,}', ' ', name).strip()
        return name

    def pre_data_process(df,filename):
        df=df.dropna()      #去空行处理
        langue=df['特征'].values.tolist()
        intent=df['一级分类'].values.tolist()
        #分割语句   这个里面你可以用 stopwords 进行停用词的处理
        stopwords = stopwordslist('data/stop_word.txt')  # 这里加载停用词的路径
        sentences=[]
        for i,j in zip(langue,intent):
            segs=jieba.lcut(precess_feature(i)) #分词
    #             segs = [word for word in segs if word not in stopwords]
            segs = [word for word in segs if word not in stopwords and len(word)>1 ]
    #             segs = list(set(segs))
            sentences.append(" ".join(segs) + "\t" + "__label__" +j)
        random.shuffle(sentences) #乱序处理
        writeData(sentences,'data/{}.txt'.format(filename))

    '''读取数据'''
    data = pd.read_excel('data/附件 2.xlsx',encoding = 'utf_8')
    # print(data.columns)
    # print(data.loc[0,'留言详情'].strip())
    # print(len(data))
    # print(data['一级分类'].value_counts()/len(data))
    data['特征'] = data.loc[:,'留言主题']+' '+data.loc[:,'留言详情'].map(lambda x:x.strip().lower())
    # data.head()

    '''划分训练集和测试集'''
    s_data = StratifiedShuffleSplit(n_splits=1, test_size=0.2, train_size=0.8, random_state=5)
    for train_index, test_index in s_data.split(data, data['一级分类']):
        data_train = data.loc[train_index]
        data_test = data.loc[test_index]
    print(len(data_train))
    print(data_train['一级分类'].value_counts()/len(data_train))

    pre_data_process(data_train,'train_data')
    pre_data_process(data_test,'test_data')


    if __name__ == '__main__':
        model = fasttext.train_supervised('data/train_data.txt', lr=0.1, dim=100, epoch=2000, word_ngrams=2)
        model.save_model("data/classifier.bin")
```

```
    # 加载已经训练好的文本分类器
    classifier = fasttext.load_model("data/classifier.bin")
    # pre_data_process(data_test)
    # 使用测试数据集评估模型,数据格式与训练数据集一样
    result = classifier.test("data/test_data.txt",k = 1)
print('F1 得分为：%0.2f'%(2*result[1]*result[2]/(result[1]+result[2])))
```