摘　要

本文针对“智慧政务”中的文本挖掘问题，运用 word2vec 模型将词语转为向量，在此基础上针对群众留言分类问题建立了 SVM 分类模型，针对热点问题挖掘建立了 K-means 文本聚类模型，给出了答复意见的评价指标，最后，讨论了模型的优缺点与推广。

针对问题一要求对留言内容进行分类，我们首先去除了数据中存在的大量杂乱格式，将文本分词、去除停用词。然后使用 word2vec 模型将词转为向量，根据词向量生成一段文本对应的向量，再使用 SVM 模型对文本进行分类。分类结果在训练集上的 F1 值为 0.93，测试集上的 F1 值为 0.88。

针对问题二要求进行热点问题挖掘，我们考虑到问题的热度与关注人数有较强的相关性，建立了基于留言条数、赞成数以及反对数的热度指标。在此基础上使用 K-means 聚类算法和命名实体识别相结合对文本聚类，聚类完成后根据热度指标排序，给出了热度排名前 5 的热点问题。其中热度排名 1 和 2 的问题分别为“58 车贷案”和“汇金路五矿万境问题”。

在问题三中对答复的评价，针对相关性，我们基于文本向量使用余弦相似度来衡量相关性；针对完整性，我们使用命名实体识别的方法，判断留言和答复中命名实体的重合程度从而评价答复的完整性；针对可解释性，考虑到答复越客观，越有助于留言者了解相关政策，我们基于客观性评价建立了评价指标，给出了留言中的指标分布。

最后，我们讨论了本模型的几个优缺点，并讨论了模型的可扩展性。

关键词：Word2vec；SVM 分类；K-means 聚类；命名实体识别；客观性分析

目 录

1 问题重述 1

1.1 问题背景 1

1.2 需要解决的问题 1

2 问题分析 1

2.1 问题一 1

2.2 问题二 1

2.3 问题三 2

3 符号说明 2

4 模型假设 2

5 模型的建立与求解 3

5.1 问题一 3

5.1.1 数据预处理 3

5.1.2 文本分词 3

5.1.3 词嵌入（word embedding） 3

5.1.4 文本分类 5

5.1.5 分类结果 6

5.2 问题二 8

5.2.1 文本相似度 8

5.2.2 热度指标 8

5.2.3 K-means 聚类 9

5.2.4 聚类结果 10

5.3 问题三 11

5.3.1 相关性 11

5.3.2 完整性 11

5.3.3 可解释性 11

5.3.4 总评价指标 11

5.3.5 统计结果 12

6 模型的评价与推广 13

6.1 模型的优缺点 13

6.1.1 优点 13

6.1.2　缺点 13
6.2　模型的推广 14
7　参考文献 14
8　附录 15
8.1　词云图 15
8.2　重要代码 16

1 问题重述

1.1 问题背景

近年来，随着信息的高速化发展以及经济全球化的发展。现代政府的事务日益复杂，数量逐渐增多，给政府相关部分的工作带来了很大的挑战。其中，在民意沟通工作方面，微信、微博、市长信箱、阳光热线等网络问政平台成为政府与群众沟通的重要渠道。随着网络问政平台数据量的不断增长，传统的依靠人工来进行留言划分和热点整理的工作越来越不足以满足人民群众的需求。同时，随着信息技术的高速发展，将自然语言处理技术运用到智慧政务系统中，对提升政府的管理水平和施政效率具有极大的推动作用。

1.2 需要解决的问题

1) 对留言内容进行处理，建立关于留言内容的一级标签分类模型。
2) 定义合理的热度指标，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，给出评价结果。
3) 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

2 问题分析

2.1 问题一

由于文本搜集中不可避免的各种问题，留言中存在大量诸如空格、换行符和 html 标签等无用符号，因此，首先需要对文本进行预处理，去除这些无用符号。

去除文本的无用符号后，由于计算机无法直接处理文本字符串，还需要进一步处理文本。本文将长文本划分成不可分割的词语，并采用 CBOW 模型进行 word2vec 操作，将文本转化为向量，就可以使用传统分类模型进行分类了。

2.2 问题二

要发现热点问题，就是对各个问题进行聚类分析，并按照事件热度排序。要进行聚类，首先需要计算文本之间的相似度，计算相似度大致有 WMD (Word Mover’s Distance)、欧氏距离和余弦距离三种方法，WMD 效果最好，但计算量大；欧氏距离计算量虽小，但不适合衡量文本相似度，因此，本文选择了余弦距离。

在热度评价指标的选取上，考虑到问题的重要程度与关注人数有较强的相关性，而留言数、赞成数、反对数可以有效地反应关注人数，因此可以建立基于这三者的热度评价指标

2.3 问题三

在建立了 word2vec 模型的基础上，要对答复的相关性、完整性、可解释性做出评价，可从以下几个方面出发。1.相关性：对于相关部门答复与留言的相关性，可以借鉴文本聚类时的方法，采用余弦距离衡量二者之间的相似度，相似度越高则相关性越大。2.完整性：答复的完整性要求相关部门的留言涵盖留言者提出的问题，为了衡量涵盖程度，可采用命名实体识别的方法，判断留言中的命名实体在答复中的涵盖情况。3.可解释性：相关部门答复具有较强的官方性，回复中不可避免地存在一些的固定回复格式，且答复内容越客观，对留言者的帮助越大，可解释性越强，因此可以建立基于留言客观性的评价指标。

3 符号说明

符号	含义
$w^{(i)}$	词典中的第i个词语
$x_j^{(i)}$	词典中第i个输入词语的 one hot 向量的第j个值
y_j	CBOW 模型输出词语 one hot 向量的第j个值
h_j	CBOW 模型隐藏层的第j个值
C	CBOW 模型窗口大小
W	CBOW 模型的权值矩阵
$v^{(i)}$	最终生成的第i个词的词向量
v_t	一段文字的向量表示
s	Similarity 文本相似度
h	Hot 话题热度
M	相关部门答复评价指标

注：少部分使用较少的符号仅在第一次使用时说明。

4 模型假设

1. 假设附件中的数据真实有效，收集时没有发生错误。
2. 假设不存在刷票数情况，赞成数与反对数均为真实数据。

5　模型的建立与求解

5.1 问题一

5.1.1 数据预处理

观察附件 2 的留言内容可以发现，由于文本搜集过程中存在一些难以避免的问题，“留言主题”和“留言详情”中存在大量诸如空格、换行符和 html 标签等无用符号，这些无意义的符号会对文本分析造成影响。首先删除这些无用符号，并剔除存在空值的数据。接着，将留言主题和留言详情合并成一个字符串，称为留言，以便后续的处理。

5.1.2 文本分词

本文借助 jieba 库[1]对中文文本进行分词，其采用了动态规划查找最大概率路径，找出了基于词频的最大切分组合。

文本分词后，文本中还会存在大量语气词、副词和介词等，例如“了”、“的”、“在”。这些停用词自身并无明确意义，只有将其放入一个完整的句子中才有一定作用，且在文本中出现频率很高，将其去除可以提高文本处理的精确度和效率。本文使用停用词表[2]去除停用词。

去除停用词后，词语中还存在很多数字，在各种数字中，有些是电话号码，有些是年份，在分词的过程中，数字会被识别成不同的词语，但由于分类的任务与不同的数字关联不大，为了避免数字带来的问题，将数字全部替换为字符 NUM。

另外，文本中还大量存在一些诸如“A 县”、“B 县”的虚构名词，普通的分词工具可能不能很好地识别这些虚构名词，因此，本文先生成了针对于这些文本的虚构名词表。

5.1.3 词嵌入（word embedding）

由于计算机无法直接处理文本字符串，传统的方法是使用 one-hot 向量，其为每个词语都保留了一个独立的位，该方法简单直观，但随着词语的增加，one-hot 的向量的维度也随之增加，且不能处理文本上下文之间的关联。为了解决上述问题，本文采用 word2vec[3]的 CBOW 模型将词语转化为向量。CBOW 模型为浅层双层的神经网络，以周围的部分词语来预测句子中的一个词语，考虑了上下文的影响，降低了词向量的维度。CBOW 模型示意图如图 5-1 所示，输入层是当前词语附近词语的 one-hot 向量，输出层是当前词语的 one-hot 向量。

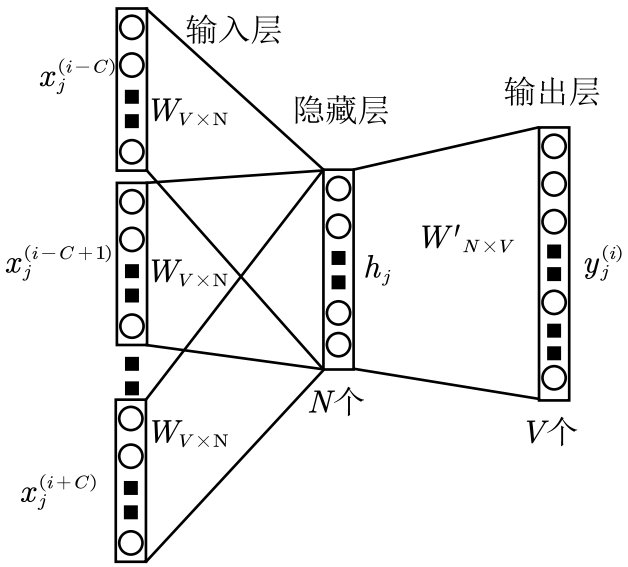

图 5-1 CBOW 模型示意图

图中V是词语 one-hot 向量的长度，N是隐藏层节点个数，隐藏层h的输出为

$$h=\frac{1}{2C}W\left(\sum_{k=-C,\ k\neq 0}^{C}x^{(i+k)}\right)$$

输出层每个结点的输入为

$$u_j=v'^{T}_{wj}h$$

其中，u_j是输出层每个结点的输入，v'^{T}_{wj}是输出矩阵W'的第j列。再经过 softmax 函数作用后即是输出y_j

$$y_j=\frac{\exp(u_j)}{\sum_{j=1}^{V}\exp(u_j)}$$

下面开始训练，本文取窗口大小(window size)为 5，迭代次数为 50，取当前单词前后最近的 5 个词作为输入，即$C=5$。且最终生成的词向量长度为 100，即$N=100$。为了提高训练速度，忽略出现次数少于 5 的词语。训练目标为输出向量y_i与当前单词的 one-hot 向量x_i差距尽可能小，即

$$\text{minimize}\ -\log\prod_{j=0,j\neq C}^{2C}P(w^{(i-C+j)}|w^{(i)})$$

训练完成后，最终生成的第i个词的词向量v_i为

$$v^{(i)}=Wx^{(i)}$$

有了单个词语的词向量后，由于要分类的是一段文字，还需要对一段文字进行向量化表示。本文采用的方法是对一段文字内所有词的词向量求和平均，即

$$v_t = \frac{1}{L}\sum_{i \in v_t} x^{(i)}$$

其中L是文本v_t中包含的有效词语个数。

5.1.4 文本分类

如图 5-2、图 5-3 所示，是交通运输类和卫生计生类的留言词云图，其余分类的词云图见附录。由图可知，不同分类的留言中出现的高频词语有明显的不同，交通运输类中出现的词语大多为“出租车”、“公司”和“快递等”，而卫生计生类中出现的词语大多为“医生”、“医院”和“患者”等，可以以此为依据，基于词向量对文本进行分类。

图 5-2　交通运输类留言词云图

图 5-3　卫生计生类词云图

根据上述方法获得了一段文本的词向量后，使用 SVM 模型分类，分类中使用高斯核函数。

由于数据中每种类别的数量不同，存在样本失衡的问题，若直接进行分类可能会出现多数类别掩盖少数类别的问题。因此，使用基于类别增加权重的 C 值，增大误分类 少数类别 带来的影响，保证少数类别的分类正确性，避免被多数类别掩盖

$$C_j = C \times \frac{n}{kn_j}$$

其中，C 是误分类的惩罚项，n 为数据总数，k 为类别数量，n_j 是类别 j 的数据个数 c_j 是类别 j 的惩罚项。

总而言之，整个分类过程的流程图如图 5-4 所示。

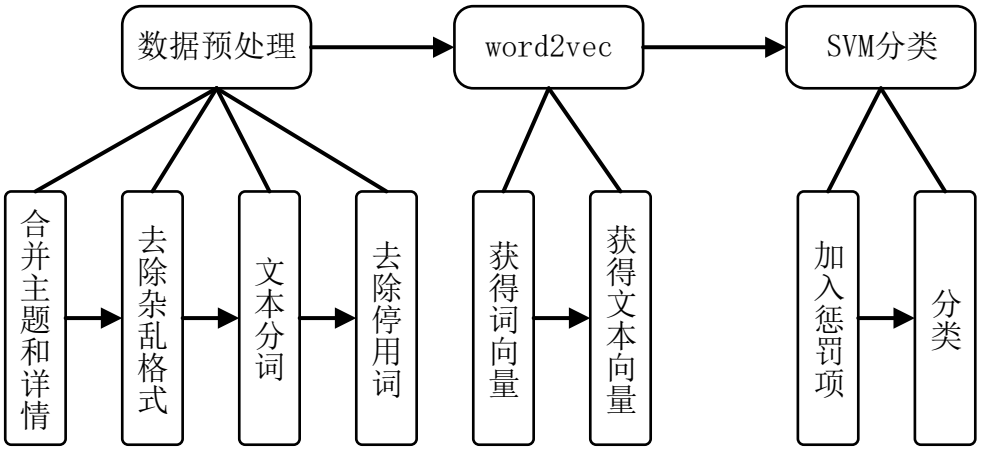

图 5-4 分类流程图

5.1.5 分类结果

从原始数据中分隔测试集和训练集，其中训练集占 80%，测试集占 20%，分类结果评价如表 5-1、表 5-2 所示。

表 5-1 训练集分类评估

类别	查准率	查全率	F1 值	个数
交通运输	0.84	0.96	0.89	504
劳动和社会保障	0.96	0.95	0.96	1558
卫生计生	0.94	0.96	0.95	714
商贸旅游	0.92	0.91	0.91	970
城乡建设	0.95	0.88	0.91	1619
教育文体	0.96	0.96	0.96	1253
环境保护	0.92	0.99	0.95	749
宏平均	0.93	0.94	0.93	7367

模型在训练集上的 F1 值接近于 1，模型在测试集上效果较好。

表 5-2 测试集分类评估

类别	查准率	查全率	F1 值	个数
交通运输	0.71	0.87	0.78	109
劳动和社会保障	0.92	0.92	0.92	411
卫生计生	0.88	0.90	0.89	163
商贸旅游	0.88	0.85	0.87	245
城乡建设	0.90	0.85	0.87	390
教育文体	0.92	0.91	0.92	335
环境保护	0.93	0.96	0.95	189
宏平均	0.88	0.89	0.88	1842

模型在测试集上的 F1 值比训练集略低，在正常范围内，未出现过拟合等情况，综上所述，说明该模型比较可靠。

如表 5-3 所示是部分内容分类结果与实际结果对比，该表按附件 2 中的编号从小到大选取了 30 个留言，表中留言编号与附件 2 中的留言编号相同，分类错误的留言已用阴影标出。

表 5-3 预测结果与实际结果对比

编号	实际分类	预测分类	编号	实际分类	预测分类
0	商贸旅游	商贸旅游	379	城乡建设	城乡建设
6	劳动和社会保障	劳动和社会保障	382	城乡建设	城乡建设
24	城乡建设	城乡建设	403	教育文体	教育文体
37	城乡建设	城乡建设	445	城乡建设	城乡建设
59	商贸旅游	商贸旅游	456	劳动和社会保障	劳动和社会保障
83	城乡建设	城乡建设	457	教育文体	教育文体
165	教育文体	教育文体	466	劳动和社会保障	劳动和社会保障
250	劳动和社会保障	劳动和社会保障	476	城乡建设	城乡建设
251	劳动和社会保障	劳动和社会保障	509	劳动和社会保障	卫生计生
260	劳动和社会保障	劳动和社会保障	530	城乡建设	城乡建设
265	劳动和社会保障	劳动和社会保障	532	城乡建设	城乡建设
277	商贸旅游	商贸旅游	553	教育文体	劳动和社会保障
284	劳动和社会保障	劳动和社会保障	555	劳动和社会保障	劳动和社会保障
303	城乡建设	环境保护	558	商贸旅游	商贸旅游
319	城乡建设	环境保护	673	城乡建设	城乡建设

5.2 问题二

5.2.1 文本相似度

要对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，就是要对留言进行聚类分析。聚类分析将距离较近的个体聚成一簇，簇的中心成为簇心，且使得簇心的距离尽量远，簇内的个体距离足够近。要计算个体之间的相似度就是要计算文本间的相似度，考虑到在问题一中已经建立了获取文本向量的模型，可以在第一问的基础上基于 word2vec 模型，将文本转为向量，对文本聚类。

首先，使用与问题一相同的方法分别计算用户留言标题与留言详情的文本向量，计算方法如下。

$$v_t = \frac{1}{L}\sum_{i \in v_t} x^{(i)}$$

得到文本向量后，文本间的相似度的计算有 WMD 距离[4]、欧氏距离和余弦相似度等方法。WMD (Word Mover's Distance)使用将所有单词从一个文档移动到另一个文档所需的最小累积成本作为文本之间的距离，其准确度最高，但计算量较大，聚类速度慢，对于大规模数据，效率较低。采用欧式距离计算文本相似度计算速度快，但欧式距离不能很好地反应出文本的相似度，效果较差。而采用余弦距离则兼有效率和准确度的优点，故采用余弦距离计算文本相似度，计算方法如下所示。

$$s = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

其中，A与B是任意两个长度相等的向量，A_i与B_i表示向量A与向量B中的第i个元素。

5.2.2 热度指标

考虑到群众反应的热点问题关注人数越多，热度越高，热度与关注人数有很强的相关性。而一类问题的留言数量、赞成数、反对数，在一定程度上就能反应这个问题的受关注程度，因此，定义热度评价指标如下

$$h = \sum_{i=0}^{n}(u_i - d_i + 1)$$

其中，u_i是该问题第i个留言的赞成数，d_i是该问题第i个留言的反对数，n是该问题的留言总数。

该指标假设所有的赞成票数、反对票数均由不同的人给出，不存在刷票等造假情况。该指标能反应该留言的关注人数，常数项 1 是因为该留言本身也计入一次关注人数。

5.2.3 K-means 聚类

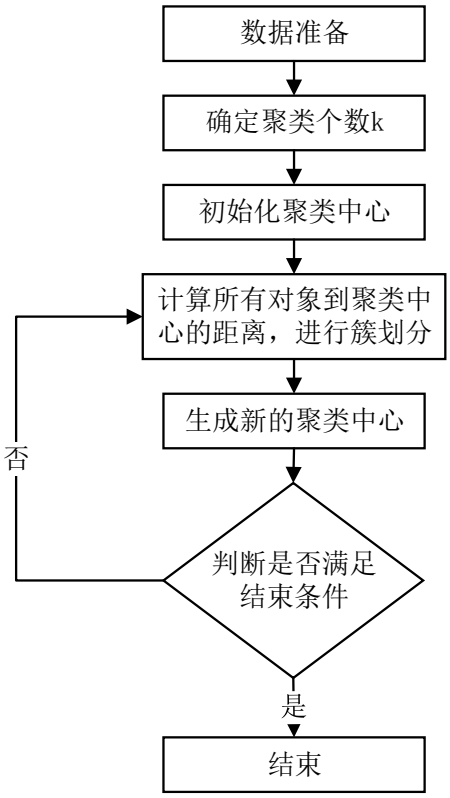

图 5-5 k-means **聚类流程图**

如图 5-5 所示，是 K-means 聚类的流程图，由图可知，K-means 聚类主要由以下几个步骤组成。

Step 1: 数据准备，确定聚类数 K，初始化聚类中心；

Step 2: 计算所有对象到聚类中心的距离；

Step 3: 根据距离进行簇划分；

Step 4: 使用划分完的簇重新生成聚类中心；

Step 5: 判断是否满足停止条件，若满足，则输出聚类结果，若不满足，则跳转到 *Step 2；*

K-means 聚类不能自动计算聚类的类别数，而需要人工指定，传统确定 K 值的手肘法由于 K 值过大，绘制出的 SSE 曲线在所有 K 的取值下都较为光滑，不能使用，如图 5-6 所示。因此，本文采用观察法确定 K-means 聚类的 K 值，并取不同的 K 值观察效果，若观察到很多不同类别的问题被归为一类，则增大 K 值，若观察到相同的话题被分为多类，则减小 K 值，本文取 K=260。

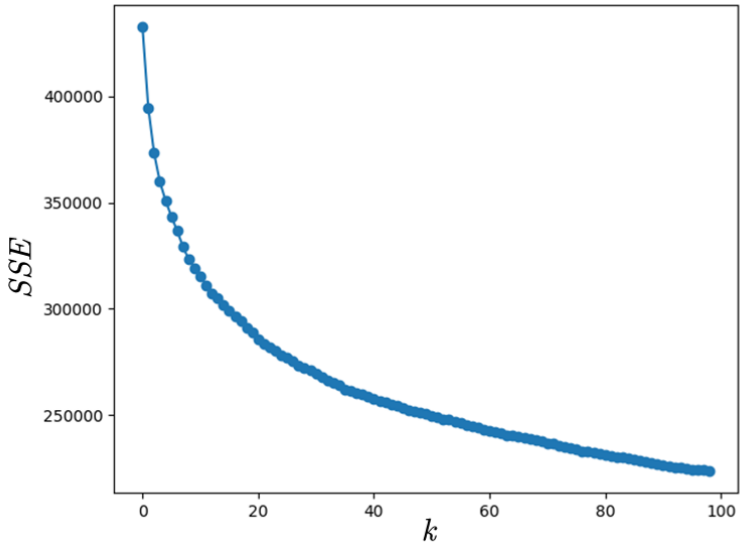

图 5-6 手肘法确定 K 值

总而言之，问题二聚类的总体流程如图 5-7 所示。

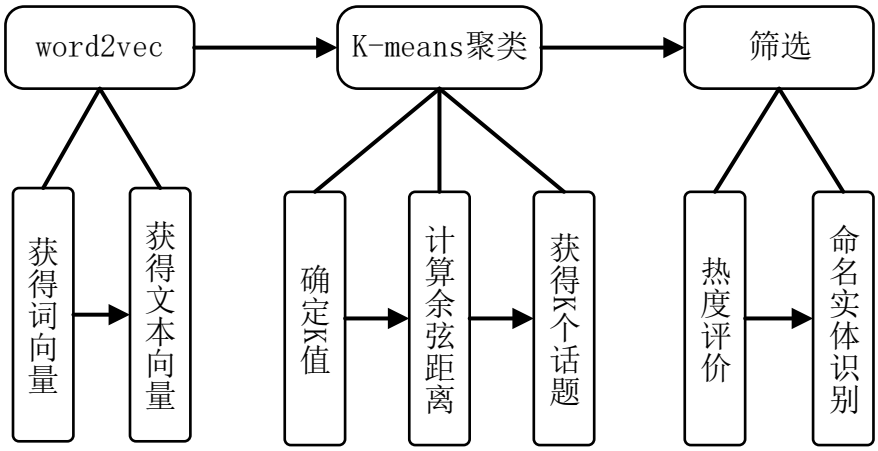

图 5-7 文本聚类流程图

5.2.4 聚类结果

表 5-4 聚类结果表

热度排名	热度指数	问题描述
1	2422	请书记关注 A 市 A4 区 58 车贷案
2	2139	A 市 A5 区汇金路五矿万境 K9 县存在一系列问题
3	1822	反映 A 市金毛湾配套入学的问题
4	709	A4 区绿地海外滩小区距长赣高铁太近
5	282	建议 A 市经开区收回东六路恒天九五工厂地块

聚类后，主要热点问题如上表所示，详细聚类结果见附件。

5.3 问题三

5.3.1 相关性

相关部门答复意见的相关性，可以继续使用余弦相似度计算，留言者的留言与相关部分的答复余弦相似度越高，相关性越大。

5.3.2 完整性

人们往往希望相关部门的答复尽量涵盖到问题的所有方面，对提出的所有问题进行回答。对此，可以结合命名实体识别进行评价，分析答复中的命名实体是否包含留言中的命名实体，若全部包含，可认为答复的完整性好。因此，完整性指标为

$$\text{integrity} = \frac{card\left(A \bigcap B\right)}{card(A)}$$

其中，$card(A)$表示有限集合A中的元素个数，集合A为留言命名实体集合，B为答复命名实体集合，本文借助 jieba 库进行命名实体识别。

5.3.3 可解释性

由于相关部门答复具有较强的官方性，回复中不可避免地存在一些的固定回复格式，该部分回复仅是正式性的要求，而对群众了解信息的帮助较小。因此，这部分对答复的可解释性没有帮助。同时，相关部门的答复应该具有较强的客观性，而不应主观性过强，因此，可以提出固定格式后，使用主观性作为评价指标。

5.3.4 总评价指标

综合上述三个指标，总评价指标M为

$$M(\text{text}) = \text{similarity} + \text{integrity} + \text{objectivity}$$

指标计算流程如图 5-8 所示（略去了数据预处理部分）。

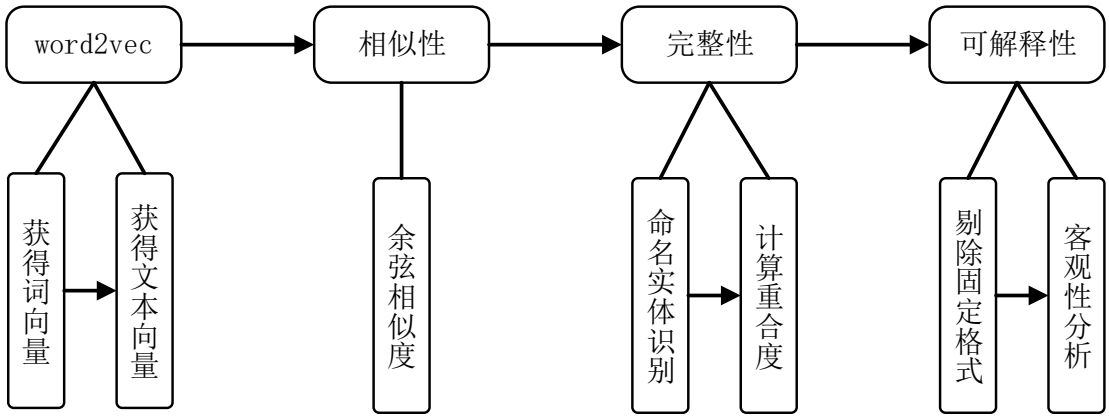

图 5-8 答复评价计算流程

5.3.5 统计结果

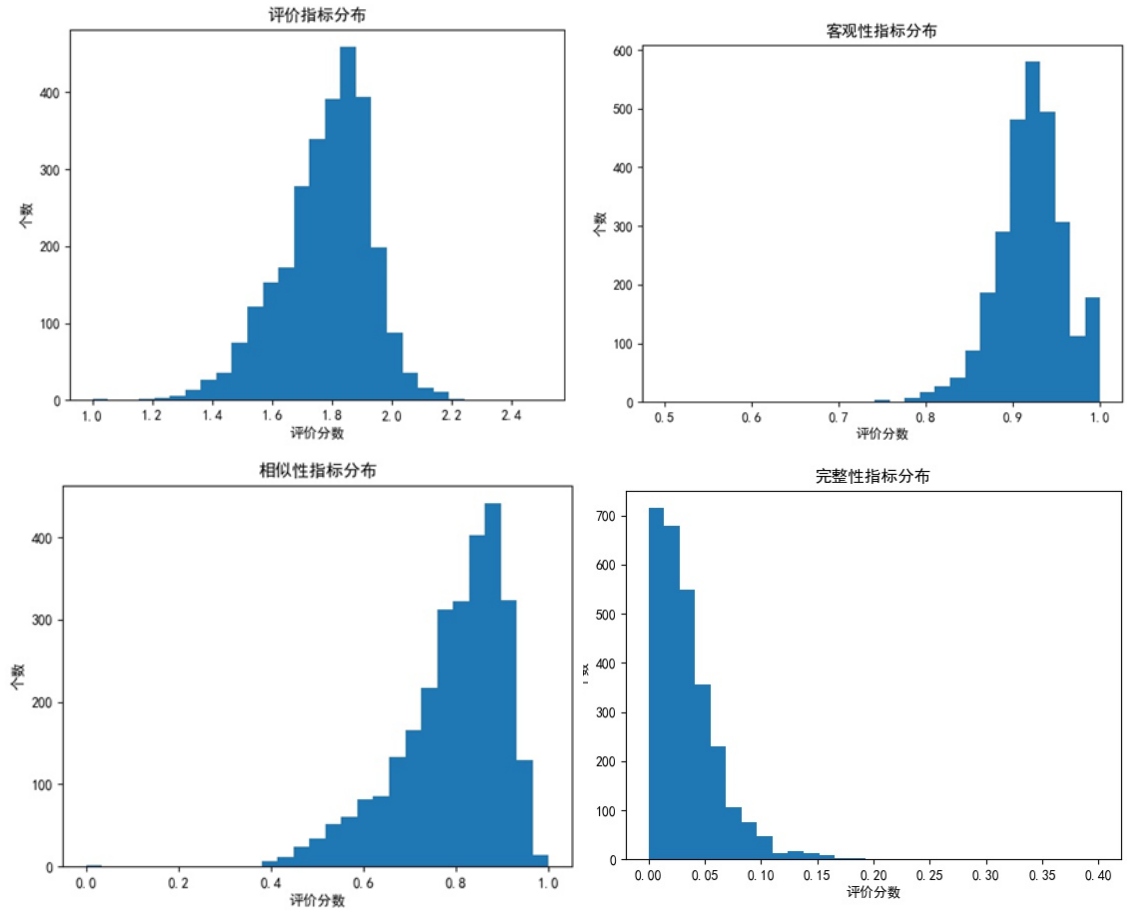

图 5-9　评价指标图

在对附件 2 中 2816 条数据进行统计后，绘出了分布图如图 5-9 所示。从图中可以观察出，评价指标总体的中位数大约在 1.8-1.9 左右，答复的客观性总体较高，说明答复的可解释性总体是较强的；大部分答复的相似性是较高的，但仍有小部分的答复与留言相似性不高，这可能是由于大量书面格式的使用，导致实际回答问题的内容在文中占比减少，或是内容与留言有偏差。对于完整性指标，总体的指标值是偏低的，这可能与留言中不可避免地会提到一些无关紧要的命名实体有关。

6 模型的评价与推广

6.1 模型的优缺点

6.1.1 优点

1. 基于 Word2vec 的模型

Word2vec 模型以周围的部分词语来预测句子中的一个词语，相比普通的 one-hot 向量和 TF-IDF 方法，考虑了上下文对一个单独词语的影响，符合人们在实际中的阅读过程，提高了模型的准确性，同时，大大降低了词向量的维度，通用性好，提高了基于 word2vec 的分类、聚类等模型的训练速度。

2. 基于 CBOW 的文本分类模型效果较好

本文基于 CBOW 模型，使用 SVM 模型对文本分类，在训练集中的平均 F1 值为 0.93，测试集中的平均 F1 值为 0.88，训练速度快，分类效果好。

3. 基于余弦距离的相似度评价

WMD (Word Mover's Distance)使用将所有单词从一个文档移动到另一个文档所需的最小累积成本作为文本之间的距离，其准确度最高，但计算量较大，对于大规模数据，效率较低。采用欧式距离计算文本相似度计算速度快，但欧式距离不能很好地反应出文本的相似度，结果往往不能令人满意。而本文中采用余弦距离则兼有效率和准确度的优点。

6.1.2 缺点

1. 不能很好地处理多义词

在 Word2vec 模型中，由于词和向量是一对一的关系，所以相同的词语对应的向量一定是相同，但相同的词语在不同的语言环境中会具有不同的意义，该模型不能很好地处理多义词问题。

2. 文本向量的计算

本文使用 CBOW 模型将词语转化为向量，在获取整个留言对应的文本向量时，采用了直接对每个词的词向量求平均的方法。该方法虽然简单、直观，但可能掩盖了文本中的部分信息。

3. 样本不均衡对分类结果造成影响

在附件二所给数据中，“交通运输”数据的数量明显少于其他类别数据的数量，尽管在 SVM 分类中加入了惩罚项，该类的 F 值仍是所有分类中最低的。尽管总体分类结果较好，分类中可能仍存在一些多数样本掩盖少数样本的问题，该问题仍有待进一步改进。

4. K-means 聚类需要人工指定类别数

K-means 聚类不能自动计算聚类的类别数，而是需要人工指定，传统确定 K 值的手肘法由于 K 值过大，绘制出的 SSE 曲线在所有 K 的取值下都较为光滑，而不能使用。因此，本文使用了观察法确定 K-means 聚类的 K 值，在实际应用中可能不能很好的估计 K 值，给聚类带来问题。

5. 聚类不能区分部分相似度较高的不同话题

聚类结果中存在这部分相似度较高的不同话题被归为一类，例如“承办 A 市 58 车贷案警官应跟进关注留言”与“A2 区余易贷受害人期盼返回血汗钱”，虽是不同事件，但相似度很高，被归为了一类。

6.2 模型的推广

本文的模型建立虽然针对“智慧政务”中的文本挖掘应用，但其具有一定的普适性，文中的 word2vec 方法、SVM 分类方法以及聚类方法等，均具有一定的通用性，并不仅能用于智慧政务，可以稍加修改应用到其他不同领域中。例如：微博热点话题挖掘、热点新闻挖掘等。

7 参考文献

[1] Sun Junyi. jieba 中文分词[EB/OL]. (2012). https://github.com/fxsjy/jieba

[2] 中文常用停用词表[EB/OL]. (2019). https://github.com/goto456/stopwords

[3] Rong X. word2vec parameter learning explained[J]. arXiv preprint arXiv: 1411. 2738, 2014.

[4] Pele O, Werman M. A linear time histogram metric for improved sift matching[C]//European conference on computer vision. Springer, Berlin, Heidelberg, 2008: 495-508.

8 附录

8.1 词云图

图 8-1 城乡建设类词云图

图 8-2 环境保护类词云图

图 8-3 教育文体类词云图

图 8-4 劳动和社会保障类词云图

图 8-5 商贸旅游类词云图

8.2 重要代码

```
1.  # dataprocess.py
2.  # 用于数据处理
3.
4.  import re
5.  import pandas as pd
6.  import jieba
7.  import jieba.posseg as pseg
8.
9.
10. # 数据预处理，数据全部以 csv 格式存放在 data 文件夹下
11. class DataProcess:
12.     def __init__(self, question):
```

```
13.         self.question = question          # 问题几的数据
14.
15.     def process(self):      # 根据不同的问题进行不同的预处理
16.         if self.question == 1:
17.             return self.question1_process()
18.         elif self.question == 2:
19.             return self.question2_process()
20.         elif self.question == 3:
21.             return self.question3_process()
22.         else:
23.             raise ValueError('不存在的 question')
24.
25.     def question1_process(self):
26.         # 类别
27.         print('正在加载问题一数据')
28.         categories = pd.read_csv('../data/附件 1.csv')
29.         categories = categories.iloc[:, 0]
30.         categories = list(categories.drop_duplicates())
31.
32.         # 留言
33.         data = pd.read_csv('../data/附件 2.csv')
34.         data = data.dropna(axis=0)
35.         data = data.drop(['留言时间'], axis=1)
36.         # 将主题和详情合并，合并后新的列改名为留言
37.         data.loc[:, '留言详情'] = data.loc[:, '留言主题'] + data.loc[:, '留言
    详情']
38.         data = data.drop(['留言主题'], axis=1)
39.         data = data.rename(columns={'留言详情': '留言'})
40.
41.         # 数据清洗
42.         print('加载完成，开始分词')
43.         data.loc[:, '留言'] = self.process_message(data.loc[:, '留言'])
44.         data.loc[:, '留言'] = self.cut(data.loc[:, '留言'])
45.
46.         pd.set_option('display.max_rows', 500)
47.         pd.set_option('display.max_colwidth', 100)
48.         print('分词结果：')
49.         print(data.loc[:, '留言'])
50.         return data, categories
51.
52.     def question2_process(self):
53.         # 留言
54.         print()
55.         print('正在加载问题 2 数据')
```

```
        data = pd.read_csv('../data/附件 3.csv')
        data = data.dropna(axis=0)

        # 分词
        print('加载完成，开始分词')
        data.loc[:, '留言主题'] = self.process_message(data.loc[:, '留言主题'])
        data.loc[:, '留言详情'] = self.process_message(data.loc[:, '留言详情'])
        data['原始留言主题'] = data.loc[:, '留言主题']
        data['原始留言详情'] = data.loc[:, '留言详情']
        data.loc[:, '留言主题'] = self.cut(data.loc[:, '留言主题'])
        data.loc[:, '留言详情'] = self.cut(data.loc[:, '留言详情'])
        return data

    def question3_process(self):
        print('正在加载问题三数据')
        # 留言
        data = pd.read_csv('../data/附件 4.csv')
        data = data.dropna(axis=0)
        # 将主题和详情合并，合并后新的列改名为留言
        data.loc[:, '留言详情'] = data.loc[:, '留言主题'] + data.loc[:, '留言详情']
        data = data.drop(['留言主题'], axis=1)
        data = data.rename(columns={'留言详情': '留言'})

        # 数据清洗
        print('加载完成，开始分词')
        data.loc[:, '留言'] = self.process_message(data.loc[:, '留言'])
        data.loc[:, '留言'] = self.cut(data.loc[:, '留言'])
        data.loc[:, '答复意见'] = self.process_message(data.loc[:, '答复意见'])
        data.loc[:, '答复意见'] = self.cut(data.loc[:, '答复意见'])

        pd.set_option('display.max_rows', 500)
        pd.set_option('display.max_colwidth', 100)
        return data

    @staticmethod
    def process_message(message):              # 清洗原始文本并分词
        # 删除超链接
        deleter = re.compile(r'http://[a-zA-Z0-9.?/&=:]*', re.S)
        message = message.apply(lambda w: deleter.sub('', w))
        deleter = re.compile(r'https://[a-zA-Z0-9.?/&=:]*', re.S)
```

```
        message = message.apply(lambda w: deleter.sub('', w))
        deleter = re.compile(r'<[^>]+>', re.S)  # 删除 html 标签
        message = message.apply(lambda w: deleter.sub('', w))
        message = message.apply(lambda w: w.upper())
        # 去除换行和空格，这里存在几种不同的空格
        useless_symbols = ['%', '\n', '\t', '\r', ' ', ' ', '  ']
        for s in useless_symbols:
            message = message.apply(lambda w: re.sub(s, '', w))
        return message

    def cut(self, message):
        message = message.apply(lambda w: jieba.lcut(w, cut_all=False))
        stop_words = pd.read_csv('../misc/stop_words.txt', sep='-
' * 10, encoding='utf8', header=None, engine='python')
        stop_words.iloc[:, 0] = stop_words.iloc[:, 0].apply(lambda w: re.su
b('  ', '', w))
        stop_words = list(stop_words.iloc[:, 0])
        message = message.apply(lambda w: [i for i in w if i not in stop_wo
rds])

        # 将整数和浮点数都替换为 NUM
        message = message.apply(lambda w: [i if not i.isdigit() else 'NUM'
for i in w])
        message = message.apply(lambda w: [i if not self.isfloat_str(i) els
e 'NUM' for i in w])
        # 删除连续出现的'NUM'
        message = message.apply(lambda w: [j for i, j in enumerate(w) if (i
 == 0 or j != w[i - 1] or j != 'NUM')])
        return message

    def isfloat_str(self, str_number):  # 判断字符串是否是浮点数
        try:
            float(str_number)
            return True
        except ValueError:
            return False
```

```
# wordprocess.py
# 用于 word2vec

from ..config import config
from functools import reduce
```

```
from wordcloud import WordCloud
from collections import defaultdict
from gensim.models.word2vec import Word2Vec
import matplotlib.pyplot as plt


class WordProcess:
    def __init__(self, words: list, labels=None):
        self.words = words
        self.labels = labels
        self.w2v_model = None
        # 格式：
        # [[句子 1], [句子 2], [句子 3], ...]

    # 保存的文件名，是否展示
    def word_cloud(self, show=False):
        print('开始绘制词云图')
        wc = WordCloud(scale=16, font_path='misc/simhei.ttf', background_color='white')
        word_frequency_by_class = self.get_word_frequency()
        for key, word_frequency in word_frequency_by_class.items():
            word_frequency['NUM'] = 0
            wc.fit_words(word_frequency)
            wc.to_file('../img/' + key + '_词云' + '.png')
            if show is True:
                plt.imshow(wc, interpolation="bilinear")
                plt.axis("off")
                plt.show()
        print('绘制完成，词云图已保存在"img"文件夹中')

    # 从 self.words 统计词频
    def get_word_frequency(self):
        word_frequency = defaultdict(lambda: defaultdict(lambda: 0))
        word_it = iter(self.words)
        label_it = iter(self.labels)
        while 1:
            try:
                words = next(word_it)
                label = next(label_it)
                label_dict = word_frequency[label]
                for w in words:
                    label_dict[w] += 1
            except StopIteration:
                break
```

```
        return word_frequency

    def word2vec(self):
        if config.load is True:
            print('config.load == True, 正在加载 word2vec 模型')
            self.w2v_model = Word2Vec.load('../misc/word2vec.model')
        else:
            print('分词结束，开始 word2vec')
            self.w2v_model = Word2Vec(sentences=self.words, size=config.word_vec_size, min_count=config.min_count,
                                      window=config.word2vec_window, iter=config.word2vec_iter, sg=0)
            if config.save is True:
                self.w2v_model.save('../misc/word2vec.model')
                print('word2vec 模型已保存在"misc/classify.model"中')
        return self.w2v_model
```

```
# classification.py
# 用于分类

import numpy as np
from ..config import config
from sklearn import svm, metrics
from sklearn.externals import joblib
from sklearn.model_selection import train_test_split


class Classification:
    def __init__(self, w2v_model, messages, labels):     # 用词向量模型分类
        self.w2v_model = w2v_model
        self.messages = messages
        self.message_vec = np.zeros((len(messages), config.word_vec_size))     # 文本向量
        self.labels = labels
        self.classify_model = None

    def calc_message_vec(self):      # 计算一段文本对应的向量，直接求平均
        for index, m in enumerate(self.messages):
            counter = 0
            for word in m:
                try:
```

```
                    self.message_vec[index, :] += self.w2v_model.wv[word]
                    counter += 1
                except KeyError:
                    continue
            self.message_vec[index, :] /= counter

    def classify(self):
        print()
        print('开始 SVM 分类')
        self.calc_message_vec()
        x_train, x_test, y_train, y_test = train_test_split(self.message_vec, self.labels, test_size=0.2)
        classify_model = svm.SVC(kernel='rbf', class_weight='balanced', probability=True)
        classify_model.fit(x_train, y_train)
        print('分类完成，进行分类评估')
        # 训练集评估
        y_train_pred = classify_model.predict(x_train)
        print(metrics.classification_report(y_train, y_train_pred))
        # 测试集评估
        y_test_pred = classify_model.predict(x_test)
        print(metrics.classification_report(y_test, y_test_pred))
        self.classify_model = classify_model
        # 使用全部数据训练分类器并保存
        if config.save is True:
            print('config.save == True，正在使用全部数据训练并保存')
            classify_model = svm.SVC(kernel='rbf', class_weight='balanced', probability=True)
            classify_model.fit(self.message_vec, self.labels)
            joblib.dump(classify_model, '../misc/classify.model')
            print('SVM 模型已保存在"misc/classify.model"中')
        return classify_model

    # 使用训练好模型对附件 2 进行预测
    def predict(self, data):
        if config.load is True:
            self.classify_model = joblib.load('../misc/classify.model')
        pred = self.classify_model.predict(self.message_vec)
        data['预测'] = pred
        data.to_csv('../misc/附件 1 预测.csv')
```