智慧政务的文本挖掘应用

摘要:目前，随着互联网、物联网的鼎盛发展，政务的文本数据不断增加。运用合理的模型处理挖掘这些数据是了解民情，更好的进行社会治理的关键。本文通过建立数据挖掘模型，对收集自互联网公开来源的群众问政留言记录的文本数据进行挖掘。对留言内容的文本分类、热点问题的挖掘及排序指标的定义，以及相关政务部门对留言的答复意见的质量评估展开研究探讨。

对于问题一，利用 Word2vec，定义分类体系继而对数据进行清洗、去重、分词等常规操作后。使用多个数据包构造对应的分类模型和分类算法，训练出文本分类器，测试评价该性能，最后使用 F-sore 对分类模型进行调优。

对于问题二，首先提取对象的特征参数，再使用计算公式通过对特征参数的运算获得对象间的相似度值。采用相似度计算的方法挖掘热点问题，归并相似留言。然后定义合理的热度评价指标进行排序。

对于问题三，在一二问的基础上，需对留言回复的内容进行挖掘分析，提取需要继续分析的内容。在考虑答复意见的相关性、完整性、可解释性的基础上制定评价方案。

关键词: Word2vec 工具　文本分类模型　文本挖掘　tf 模型　中文文本挖掘模型　数据挖掘

目录

一、研究目标

1.1 研究背景

随着大数据时代的来临，导致从微博、微信、阳光热线、市长信箱等渠道产生的各类社情民意的文本数据不断攀升。这给传统的依靠人工进行热点整理和民情留言划分的政务部门的工作带来巨大挑战。而随着云计算、大数据、人工智能等技术的飞跃发展，应运而生的智慧政务系统已然是行政治理创新发展的新趋势。智慧政务系统是基于自然语言处理技术的面向政府机关内部，其他政府机构的信息处理系统。智慧政务系统利用高现代信息技术对政府部门相关的文本数据进行信息化处理挖掘，以提高政府部门依法行政的水平。且智慧政务系统也将政务工作变得更有效、更精简、更公开、更透明，为企业和居民提供更好的服务。

1.2 挖掘目标

1.2.1 在日常行政工作中，传统的对民情留言的处理工作人员先按照一定的体系对留言进行分类，再将民意留言分派到相应部门处理。为减少传统方法处理带来的工作量大、效率低、且差错率高等弊端。试根据附件二提供的数据，建立关于留言内容的一级标签分类模型。并使用 F-Sore 对分类方法进行评价：

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

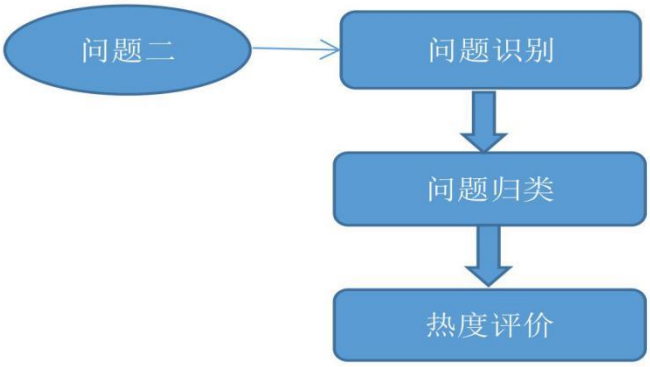
其中 P_i 为第 i 类的查准率，R_i 为第 i 类的查全率。

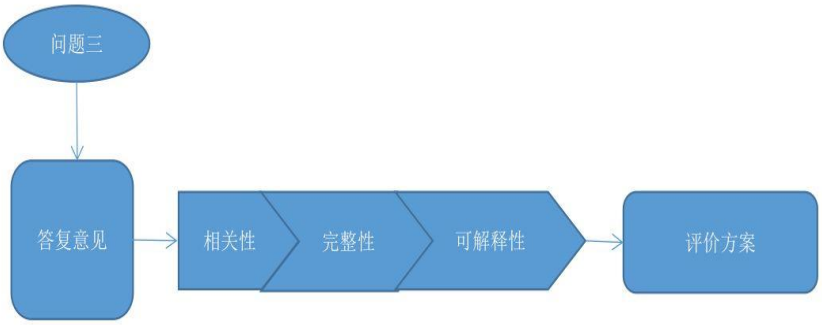
1.2.2 在行政工作中，及时发现处理热点问题是提升工作效率的关键。我们将某一时段内群众集中反映的某一问题成为热点问题。试根据附件三，对热点问题进行挖掘。识别相似留言，将相似留言归并为同一问题，建立合理的评价指标模型，给出评价结果。

1.2.3 根据附件四，从相关性、完整性、可解释性等方面挖掘分析相关部门对留言的答复意见的质量，给出合理的评价方案。

二、问题分析

2.1 总体流程





2.2 具体步骤

2.2.1 问题一分析

针对问题一，我们将用到文本分类模型，在给定的分类体系下，根据文本内容自动的确定文本关联的类别。文本分类属于有监督的学习，它有以下基本步骤：

A. 定义分类体系

B. 将预先分类过的文档作为训练集，对文档做分词、去停用词等准备工作

C. 确定表达模型，对文档矩阵进行降维，提取训练集中最有用的特征

D. 应用具体的分类模型和分类算法，训练出文本分类器

E. 在测试集上测试并评价分类器性能

F. 应用性能最高的分类模型对文档进行分类

G. 用 F-sore 值对分类模型进行评价、调优。

2.2.2 问题二分析

针对问题二，若要挖掘特定的时间、特定的地点以及发生的热点事件，然后对其进行归并，将反映同一问题的留言归为一类。我们将采用相似度计算的方法。相似度的计算方法主要包括两部分，即：提取对象的特征参数；然后用一种计算公式通过对特征参数的运算获得对象间的相似度值。最后定义一个合理的热度评价指标，来做一个热度排序。

这里运用的核心思想就是文本挖掘。文本挖掘指的是从大量文本数据中获取有价值的信息和知识。其最重要最基本的应用是实现文本的分类和聚类，前者是有监督的挖掘算法，后者是无监督的挖掘算法。由此，我们将按以下步骤进行文本挖掘：

A. 获取文本

B. 文本预处理

提出噪声文档以改进挖掘精度，考虑仅选取部分样本以提高挖掘效率

C. 文本的语言学处理

经过以上步骤我们将得到比较干净的素材。然而文本中起到关键作用的是一些中心词，甚至主要的词就能决定文本取向，因此我们需要进行分词处理。分词即将连续的字序列按照一定的规范重新组合成词序列的过程。针对中文分词有很多分词算法，诸如最大匹配法、最优匹配法、机械匹配法、逆向匹配法、双向匹配法等等。同时也可以使用磁性标注。

再者我们得到的文本数据中同时存在大量无意义的词，所以需进行去停用词处理。

D. 文本的数学处理——特征提取

我们期望获取到的词汇，既能保留文本信息，同时又能反映它们的相对重要性。若对所有词语都保留，必然导致维度偏高，矩阵将变得特别稀疏，从而影响挖掘结果。此时就需要采用特征提取。

除此以外，在问题二上，进行归并处理时我们还用到了命名实体识别，提取对应的实体内容、人名地名、企业机构名。它实际上可以看成分词与词性标注任

务的集成；命名实体的边界可以通过{B,M,E,S}确定，其类别可以通过 B-nt 等附加类别的标签来确定。

2.2.3 问题三分析

在获得的留言答复意见的文本数据中，通过对答复意见内容的挖掘分析，进一步提取需要继续分析的内容深入挖掘，从答复意见的相关性、完整性、可解释性三方面来制定合理的评价方案。

三、模型的建立与求解

问题一要求根据附件二提供的数据，建立关于留言内容的一级标签分类模型。由此，利用 Word2vec，使用与二分类相关的方法，读取数据并对数据进行清洗：

```
import pandas as pd
import re

data = pd.read_csv(r'C:\Users\29072\Desktop\C 题\附件 2.csv', encoding='utf-8', dtype=str)
data = data.astype(str)

data.shape
data.head()
data['留言主题'].value_counts()
data_dup = data['留言主题'].drop_duplicates()  #去重
data_qumin = data_dup.apply(lambda x: re.sub('\t', '',x))
```

对数据进行去重处理，分词处理：

```
import jieba

data_cut = data['留言详情'].apply(lambda x: jieba.lcut(x))
```

去除停用词处理：

```
# 去停用词

with open(r'C:\Users\29072\Desktop\C 题\stopword.txt','r')as f:
    stop = f.read()
stop = stop.split()
stop = [' ','\t','\n','K','\u3000','  '] + stop
```

```
data_after_stop = [[j for j in i if j not in stop]for i in data_cut]
sentence_list.append(data_after_stop)
```

统计词频：

```
all_words = sum(data_after_stop,[])

word_count = pd.Series(all_words).value_counts()
```

使用 Gensim 库构造词向量：

```
From genism.models import word2vec
Import logging

Logging.basicConfig(format='&(asctime)s:  &(levelname)s:  &(message)s',
level=logging.INFO)
raw_sentences = sentence_list
sentences = [s.split() for s in raw_sentence]
print(sentences)

model = word2vec(sentences, min_count=1)
# 此处随机举例进行测试
model.similarity('安全','吵闹')
```

导入以下包，构造模型：

```
import tensorflow as tf
import numpy as np
import math
import pickle as pkl
import re
import jieba
import os
import os.path as path
import collections
import pandas as pd
```

此时我们需要定义 Word2vec 的一个类，并在这个类中定义函数：

①初始化函数 def__init__()

我们对读取进来的语料库进行初始化操作：

```
def __init__(self,
                vocab_list = None,
                embedding_size = 200,
                win_len = 3,
```

```
                learning_rate = 1,
                num_sampled = 100):

        self.batch_size = None
        assert type(vocab_list) == list
        self.vocab_list = vocab_list
        self.vocab_size = vocab_len(list)
        self.win_len = win_len
        self.num_sampled = num_sampled
        self.learning_rate = learning_rate

        self.word2id = {}
        for i in range(self.vocab_size):
            self.word2id[self.vocab_list[i]] = i

        self.train_words_num = 0
        self.train_sentence_num = 0
        self.train_times = 0
        self.build_graph()
```

②构造关系图 def build_graph

```
def build_graph(self):
        self.graph = tf.Graph()
        with self.graph.as_default():
            self.train_inputs = tf.placeholder(tf.int32, shape=[self.batch_size])
            self.train_labels = tf.placeholder(tf.int32, shape=[self.batch_size,1])

            self.embedding_dict                                                    =
tf.Variable(tf.truncated_normal(shape[self.vocab_size,self.embedding_size]))
            self.nec_weight                                                        =
tf.Variable(tf.truncated_normal(shape[self.vocab_size,self.embedding_size]))
            self.bias = tf.Variable(tf.zeros([self.vocab_size]))

            embed = tf.nn.embedding_lookup(self.embedding_dict,self.train_inputs)

            self.loss = tf.reduce_mean(
                tf.nn.nce_loss(weights=self.nec_weight,
                               biases=self.bias,
                               inputs=embed,
                               labels=self.train_labels,
                               num_sampled=self.num_sampled,
                               num_classes=self.vocab_size
```

```
                                        ))
            self.train_op                                                                    =
tf.train.GradientDescentoptimizer(learning_rate=self.learning_rate).minimize(self.loss)

            self.test_word_id = tf.placeholder(tf.int32, shape=[None])
            vec_12_model = tf.sqrt(tf.reduce_sum(tf.square(self.embedding_dict, 1))
                                        )
            self.normed_embedding = self.embedding_dict/vec_12_model
            test_embed = tf.nn.embedding_lookup(self.embedding_dict,self.test_word_id)
            self.similarity = tf.matmul(test_embed,self.normed_embedding)

            self.init = tf.global_variables_initializer()

            self.saver = tf.train.Saver()
```

③训练函数 def train_by_sentence

```
def train_by_sentence(self,input_sentence = []):
        sent_num = input_sentence._len_()
        batch_inputs = []
        batch_labels = []
        for sent in input_sentence:
            for i in range(sent._len_()):
                start = max(0,i-self.win_len)
                end = min(sent._len_(),i+self.win_len)
                for index in range(start,end):
                    if index == i:
                        continue
                    else:
                        input_id = self.word2id.get(sent[i])
                        label_id = self.word2id.get(sent[index])
                        batch_inputs.append(input_id)
                        batch_labels.append(label_id)
        batch_inputs = np.array(batch_inputs,int32,dtype=np)
        batch_labels = np.array(batch_labels,int32,dtype=np)
        batch_labels = np.reshape(batch_labels,[batch_labels._len_(),1])

        feed_dict = {
                        self.train_inputs:batch_inputs,
                        self.train_labels:batch_labels}
        loss = self.sess.run(self.train_op,feed_dict=feed_dict)

        self.train_words_num += batch_inputs._len_()
        self.train_sentence_num += input_sentence._len_()
```

```
        self.train_times += 1
```

④保存函数模型 def save_model

```
def save_model(self, save_path):

        if os.path.isfile(save_path):
            raise RuntimeError('the save path should be a dir')
        if not os.path.exists(save_path):
            os.mkdir(save_path)


        #记录模型参数
        model = {}
        var_names = ['vocab_size',
                     'vocab_list',
                     'learning_rate',
                     'word2id',
                     'embedding_size',
                     'logdir',
                     'win_len',
                     'num_sampled',
                     'train_words_num',
                     'train_sents_num',
                     'train_times_num',
                     'train_loss_records',
                     'train_loss_k10',
                    ]
        for var in var_names:
            model[var] = eval('self.'+var)

        param_path = os.path.join(save_path,'params.pkl')
        if os.path.exists(param_path):
            os.remove(param_path)
        with open(param_path,'wb') as f:
            pkl.dump(model,f)

        #  记录 tf 模型
        tf_path = os.path.join(save_path,'tf_vars')
        if os.path.exists(tf_path):
            os.remove(tf_path)
        self.saver.save(self.sess,tf_path)
```

⑤读取函数模型 def load_model

```
def load_model(self,model_path):
        if not os.path.exists(model_path):
            raise RuntimeError('file not exists')
        param_path = os.path.join(model_path,'params.pkl')
        with open(param_path,'rb') as f:
            model = pkl.load(f)
            self.vocab_list = model['vocab_list']
            self.vocab_size = model['vocab_size']
            self.logdir = model['logdir']
            self.word2id = model['word2id']
            self.embedding_size = model['embedding_size']
            self.learning_rate = ['learning_rate']
            self.win_len = model['win_len']
            self.num_sampled = model['num_sampled']
            self.train_words_num = model['train_words_num']
            self.train_sents_num = model['train_sents_num']
            self.train_times_num = model['train_times_num']
            self.train_loss_records = model['train_loss_records']
            self.train_loss_k10 = model['train_loss_k10']
```

⑥构造相似度计算模型

```
def cal_similarity(self,test_word_id):
        sim_matrix = self.sess.run(self.similarity,feed_dict = [self.test_word_id])
        test_words = []
        near_words = []
        for i in range(test_word_id._len_()):
            test_words.append(self.vocab_list[test_word_id][i])
            nearest_id = [sim_matrix[i,:].argsort()[1:10]]
            nearest_word = [self.vocab_list[x] for x in nearest_id]
        return test_words,near_words
```

四、模型的检验与推广评价

通过对三个问题的研究探讨，基于模型的实验结果的检验。在给定文本分类体系的条件下，本文的分类模型具有较好的实用价值。利用常规的数据挖掘手段，使用一般文本挖掘的基本步骤可以完成中文文本挖掘的基本任务。对给题的相关

要求及问题分析，通过数据的检验可知模型具有可信度。

五、结论

本文从常规的文本挖掘模型出发，利用中文文本挖掘的相关方法，建立了文本分类模型，并对分类方法进行了评价。实现了热点问题的挖掘、归并，并提供了合理的热度评价排序指标。但不可否认的是我们的模型仍存在一些弊端，在考虑数据不平衡带来的影响下、相似度计算复杂以及如何将问题三中相关性、完整性、可解释性较好的量化，构建合理指标来计算等方面，本文的模型仍存在很大的改进发展空间。最后，感谢指导老师的对我们的指导，感谢相关学者的文献提供的理论帮助。

六、参考文献

[1]王福,刘宇霞,康丽琴.基于最大频繁模式挖掘的移动政务系统场景化服务研究[J].现代情报,2020,40(01):41-48.

[2]郭明军,童楠楠,王建冬.政务数据与社会数据共享利用中存在的问题及应对举措[J].中国经贸导刊,2019(08):37-38.

[3]汪祖柱,阮振秋.基于关联规则的政务微博公众评论观点挖掘[J].情报科学,2017,35(08):19-22.

[4]林崇贵,杜伟杰.政务大数据应用的“五大趋势”和“三大形态” 基于浙江省经济信息中心大数据的实践探索[J].信息化建设,2017(06):26-29.

[5]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.

[6]罗东玲,刘瑛.加快推进江苏数字政府建设的对策研究[J].江苏科技信息,2019,36(26):25-27.

[7]黄乙中.浅谈数据治理建设方案[J].轻工科技,2020,36(01):65-67.

[8]李建中,王宏志,高宏.大数据可用性的研究进展[J].软件学报,2016,27(07):1605-1625.