“智慧政务”与智能文本挖掘

摘要

问题一：我们要解决的问题是从大量的、有噪声的、模糊的、随机的数据中，提取隐含在其中的潜在有用信息，用统计的方法对给定的数据集合假设一个分布或概率模型。我们利用深度学习对大量文本数据进行预处理得到长文本中的词向量及关键词向量，与一级标签的关键词进行匹配，利用 CNN 获取了文本的特征表达能力，根据 Python 程序导出的表格进行一级标签分类的匹配，并对分类模型进行了 F-Score 评价。

问题二：要搜集热点问题，就要面临特征性较弱的自然语言处理，我们使用递归神经网络 RNN 捕获变长且双向的信息。由于文本较长且连续性高，我们首先加强其特征性，因此引入 Attention 机制，增强了部分词的表达能力同时削弱其它词汇的影响。再引入 TestRNN 将长文本中的词向量与句向量分离，即可得到

问题三：针对连续型文本数据，我们在在对评价指标约简之前首先对指标中的连续属性进行离散化处理，在此基础上，基于 RS 理论对指标体系进行约简。通过属性约简，得到最终的答复意见质量指标体系，在此基础上，采用 SVM 算法对答复意见质量进行仿真。

关键词：CNN/RNN 深度学习 大数据预处理 标签分类

目录

一　问题重述……3

1.1　背景简介……3
1.2　问题简述……3

二　问题分析与假设……3

2.1　问题分析……3
2.2　假设与简化……4

三　变量定义……4

四　问题一求解……5

4.1　深度学习用于文本预处理……5
4.2　一级标签与关键词检索……5
4.3　CNN 网络获取特征表达能力……7
4.4　一级标签分类模型……8

五　问题二求解……8

5.1　热点问题的评价指标……8
5.2　Attention 机制与 TestRNN 应用……9
5.3　热点问题的评价机制……10

六　问题三求解……11

6.1　答复意见质量模型……11
6.2　答复意见评价体系……12

七　模型评估……13

7.1 模型优点……13
7.2 模型缺点……13

八　参考文献……14

附录……15

一、问题重述

1.1 背景简介

智慧政务的建设是实现电子政务升级发展的突破口，是政府从“管理型”走向“服务型、智慧型”的必然产物，也是引导智慧城市建设的主干线。传统城市和政府是按业务、管理职责分别设定的，各个部门各司其职，存在严重的部门壁垒，城市基本运行数据孤立的存在于不同的“烟囱”中。随着信息通讯技术的快速发展，政府面临电子化、信息化、网络化压力，而电子政务让传统政府向廉洁、勤政、务实和高效政府转变。

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、汇聚明智、凝聚名气的重要渠道，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的挑战。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。

1.2 问题简述

群众留言分类：传统群众留言分类可以通过人工特征工程或专家规则（Pattern）进行分类，这样做的好处是短平快的解决 top 问题，但费时费力，覆盖的范围和准确率都非常有限。题目要求根据附件 2 中大量数据，建立群众留言的一级标签分类模型。

热点问题挖掘：某一时段内群众集中反映的某一问题可称为热点问题。热点问题整理有助于相关部门进行有针对性地处理，提升服务效率。我们需要根据附件 3 将某一时段内反映特定地点或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。

答复意见的评价：针对附件 4 相关部门对留言的答复，从答复的相关性、完整性、可解释性等角度建立一套评价方案。

二、问题分析与假设

2.1 问题分析

需要解决的主要问题是如何从大量的、有噪声的、模糊的、随机的数据中，提取隐含在其中的潜在有用信息，可以用统计的方法对给定的数据集合假设一个分布或概率模型，然后根据模型采取相应的方法来进行挖掘。首先需要集中收集某一时段内群众反映的问题，依据收集的问题根据话题进行数据分类，出现频率高的即为热点话题。至于针对留言答复的评价，可制定多个指标，再根据这些指标对留言的质量进行综合全面地评价。

问题一我们需要利用文本的一些特征表达建立全自动的特征方程以根据其内容进行一级标签分类。

问题二中，首先确定热点问题的指标分配，使用递归神经网络 RNN 的双向

LSTM，捕获变长且双向的的 n-gram 信息。

问题三建立在问题一、二求解结果的基础上，通过属性约简，得到最终的答复意见质量指标体系，在此基础上，采用 SVM 算法对答复意见质量进行仿真。

2.2 假设与简化

在整个数据挖掘的过程中，被研究的对象是挖掘过程的基础，它驱动整个网络技术挖掘的全过程，同时，也是检验挖掘结果与分析人员完成挖掘的依据。对于未优化文本处理模型的应用与挖掘，我们做出如下假设：

- 各类一级标签已经保证非互斥性。
- 为语义不明，即使人工分析也需要相对长反应时间的意见预留 1%的余量，即认为文本处理模型的误差有 1%是由文本自身造成。
- 深度学习中忽略训练震荡的影响。
- 意见答复的评价体系是相关性、完整性和可解释性的线性回归模型。

三、变量定义

变量	定义
w_t	词权重
$\hat{P}(w_t \mid w_1^{t-1})$	权重概率函数
n	向量维数
$k^{\{n\}}$	向量表征的概念值
$C(w_k)$	文本分布式矩阵
x_t	源语言词汇
y_t	目标语言翻译词
h_t	过程表示向量
P_i	第 i 类的查准率，用于分类方法评价体系
R_i	第 i 类的查全率
α_{ij}	翻译第 i 个词时，第 j 个词的贡献（注意力）
$S=(U,A,V,F)$	粗糙集信息

R	族等价关系
$core(R)$	核约简集
$Pos_p(Q)$	约简正域
α_i^*	支持向量

四、问题一求解

4.1 深度学习用于文本预处理

传统文本预处理的方法主要依靠人工特征方程+浅层分类模型实现，称为训练文本分类器，其流程如图 4.1.1 所示。

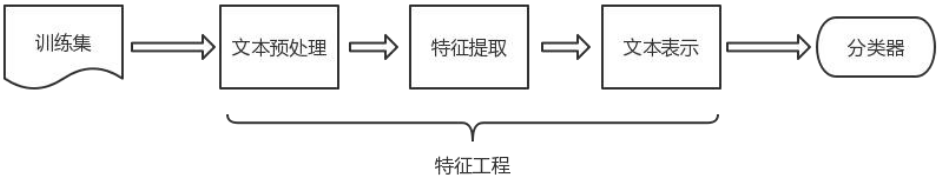

图 4.1.1 传统文本分类流程图

即大量文本问题被分为特征工程部分和分类器部分。传统文本处理存在高纬度、高稀疏度的问题，即其特征表达能力很弱，神经网络不擅长对此类问题的处理，并且需要大量人工计算，时间和空间上的成本较高。这里我们引入深度学习解决大规模文本的主要问题——解决文本表示。

4.2 一级标签与关键词检索

这里我们首先要确定文本的分布式表示，即文本的词向量。分布式表示的基本思想是将每个词表达成 n 维稠密、连续的实数向量，与之相对的 O-E 向量空间只有一个维度是 1，其余全 0。分布式表示具备高度特征表达能力，如 n 维向量每维 k 个值可以表征 $k^{\{n\}}$ 个概念。神经网络的隐层和潜在变量的概率主题模型，都要应用分布式[2]表示。

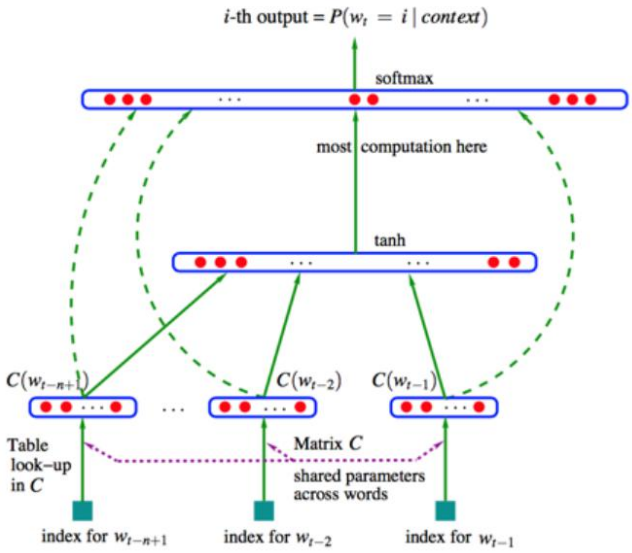

图 4.2.1 分布式 NNLM 表示（图源网络）

对于一级标签的分类，我们采用神经网络语言模型(NNLM)，其采用文本分布式表示，NNLM 模型的目标是构建以下语言模型：

$$f(w_t, w_{t-1}, \cdots, w_{t-n+1}) = \hat{P}(w_t \left| w_1^{t-1} \right.) \tag{1.1}$$

图 4.2.1 中的 $C(w_k)$ 为词的分布式表示，即词向量是训练语言模型的一个附加产物。常用的文本分析进程有两个，正向预测的 CBOW 和反向的 Skip-Gram。在该问题中我们用到 CBOW 结构，其流图如图 4.2.2 所示：

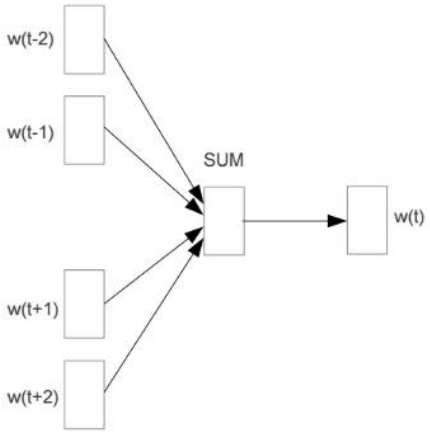

图 4.2.2 CBOW 结构

至此，我们可以利用深度学习算法的数据迁移性，通过词向量的表示方式，把文本数据从高纬度高稀疏的神经网络难处理的方式，变成了类似图像、语音的的连续稠密数据。

4.3 CNN 网络获取特征表达能力

已经用词向量解决了文本表示的问题，我们要继续探讨的文本分类问题则是

利用 CNN 深度学习网络算法及其变体解决自动特征提取（即特征表达）的问题。

由于问题一仅对留言数据进行初步分类，这里我们将 CBOW 结构演变为简单的主流文本分类模型 fastText，演变后的结构如下：

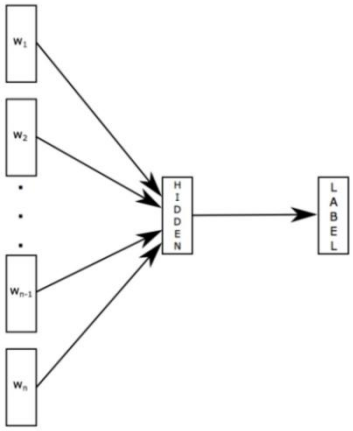

图 4.3.1 fastText 结构

FastText[8] 的原理是把句子中所有的词向量进行平均，相当于只有一个特殊 CNN，需要辅助时加入一些 n 维特征的 w_t 来捕获局部序列信息。TestCNN 的核心点在于捕获局部相关性，在文本分类任务中可以提取句子中 n 维关键信息。

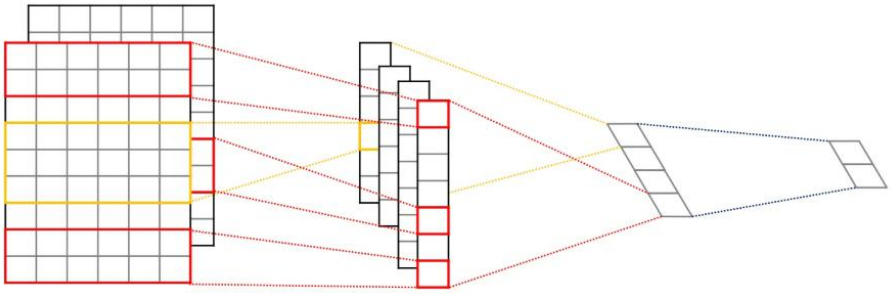

图 4.3.2 TestCNN 简略原理图（源自网络）

这里我们对 CNN 算法的适用进行选择。

特征：词向量有静态与非静态方式，其中非静态(non-static)是在训练过程中更新词向量。我们选取非静态中的 fine-tunning 方式，它是以预训练(pre-train)的 word2vec 向量初始化词向量，训练过程中调整词向量，能加速收敛，如果有充足的训练数据和资源也可以直接随机初始化词向量。

通道(channel)：文本的输入通道通常是不同方式的 embedding(比如 word2vec 或 Glove)，这里为了方便计算，选用 word2vec 方式。

卷积：文本为一维向量，因此卷积也为一维卷积，无需转化。

Pooling 层：利用 CNN 解决文本问题时，Pooling 层输入时保留全局序列信息，即 1 维最大 k 个信息。

4.4 一级标签分类模型

利用 Python 导出目标语言后，我们将原有语言进行替换，使之成为其过程词汇，利用 Excel 的查找与计数功能容易算出查准率 P_i 与查全率 R_i，即可计算得到 F 函数的结果如下：

分类	P	R	F
城乡建设	0.93950905	0.91654554	
党务政务	0.8992811	0.93099083	
国土资源	0.84937579	0.82782372	
环境保护	0.94488427	0.96071281	
纪检监察	0.88998738	0.9605362	
交通运输	0.98821306	0.96255013	
经济管理	0.97328448	0.94765685	
科技与信息产业	0.73987345	0.89760211	0.898160089
民政	0.84686458	0.91555143	
农村农业	0.88958478	0.87176479	
商贸旅游	0.91787706	0.97288634	
卫生计生	0.80414371	0.87640042	
政法	0.74513258	0.94748558	
教育文体	0.92745747	0.96003646	
劳动和社会保障	0.81837628	0.87653199	

图 4.4　F-Score 分类评价指数

五、问题二求解

5.1 热点问题的评价指标

问题一中我们运用 CNN 算法解决了文本的特征表达工作，但 CNN 算法自身受到 filter_size 的限制，一方面无法建模更长的序列信息，另一方面 filter_size 的超参调节也很繁琐。要搜集热点问题，就要面临特征性较弱的自然语言处理，此时更常用的是递归神经网络 RNN(Recurrent Neural Network)。具体在文本分类任务中， RNN 使用的是双向 LSTM，可以捕获变长且双向的的 n-gram 信息。

下图为 RNN 用于文本分类处理的网络结构示意图。

（默认最后一级直接匹配输出）

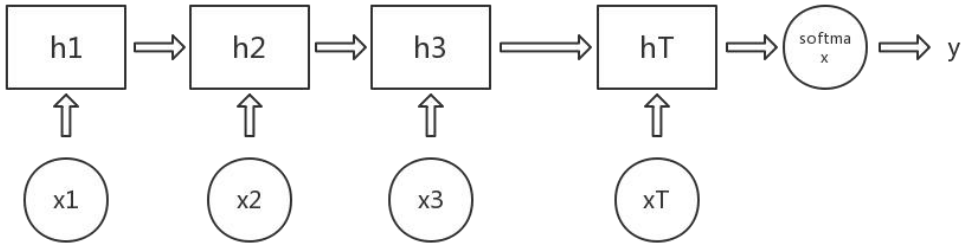

图 5.1.1 RNN 文本分类示意图

5.2 Attention 机制与 TestRNN 应用

对于问题二，文本较长且连续性高，我们首先要加强其特征性，这里需要引入 Attention 机制[9]，结构图如下。

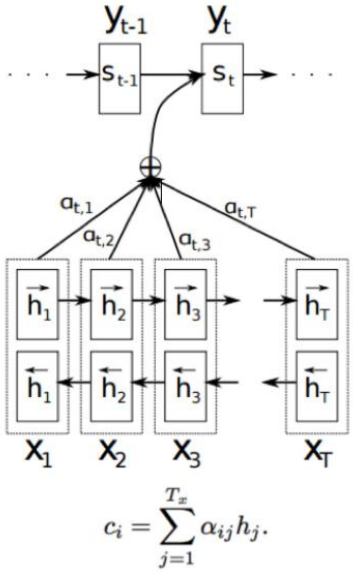

图 5.2.1 Attention 机制结构图（源自网络）

在机器翻译中，x_t 是源语言的一个词，y_t 是所求目标语言的一个词，机器翻译的任务就是给定源序列 $\{x_1, x_2, x_3, \cdots, x_t\}$ 得到目标序列 $y = \{y_1, y_2, y_3, \cdots, y_t\}$。

翻译 y_t 的过程产生取决于上一个词 $y_{t\text{-}1}$ 和源语言的词的表示 h_j，h_j 则是 bi-RNN 模型的表示。在翻译过程中，每个词所占的权重是不一样的，一些关键语言会压制其他词语的特征。α_{ij} 是翻译第 i 个词时，第 j 个词的贡献，称为注意力，这一参变量对 Attention 机制的应用起到了很大作用。

将 TestRNN 与 Attention 机制结合，得到如下处理结构[10]：

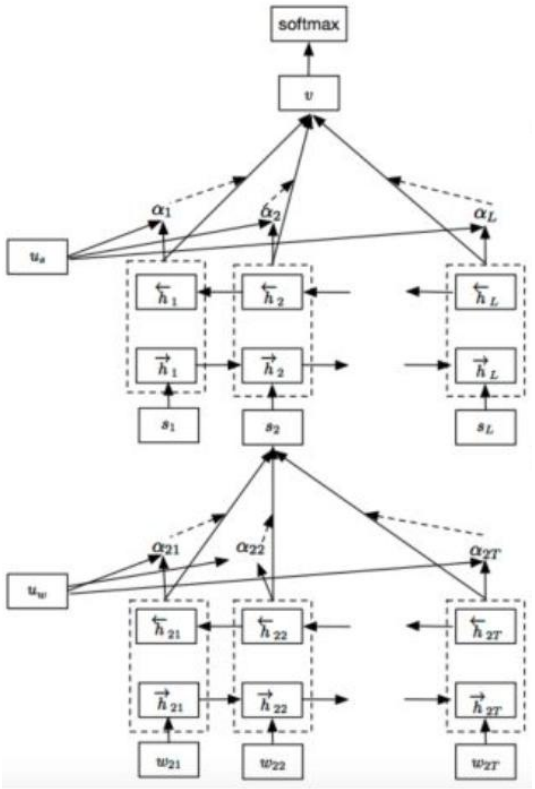

图 5.2.2 文本特征分类模型（源自网络）

对于目前的网络结构，一方面用层次化结构保留了文档的结构，另一方面分离出了 word-level 和 sentence-level。至此，我们得到了直观捕获词和句子特征的分类模型。

5.3 热点问题的评价机制

根据编程整理出的热点问题 top 5 分别为：

热度排名	热度指数	时间范围	地点/人群	问题描述
1	93.47	2019/3/26 至 2019/10/11	A 市	对红绿灯建议
2	89.76	2019/4/17 至 2019/12/30	松雅西地	地铁 3 号线出入口建议
3	86.53	2019/07/28 至 2019/09/10	魅力之城	噪音扰民
4	75.43	2019/4/15 至 2019/4/15	省商学院	拆除变压器
5	74.32	2019/6/18 至 2019/11/21	星沙大道	摩托车飙车

同时整理出其对应热点问题留言明细表，从不同方面共计 41 条留言。

六、问题三求解

6.1 答复意见质量模型

RS理论只能处理离散型数据, 对连续型数据无法进行处理, 因此, 在对评价指标约简之前首先要对指标中的连续属性进行离散化处理。在此基础上, 基于RS 理论对指标体系进行约简, 从而降低模型复杂度。

可以用四元组 $S=(U,A,V,F)$ 来表达粗糙集中的信息, 其中：

$U=\{u_1,u_2,...,u_{|u|}\}$ ，是有限非空集合, 即论域；

$A=\{a_1,a_2,...,a_{|a_{|A|}|}\}$,为属性的非空有限集合；

$AV=\bigcup_{a\in A}V_n$, Va 为属性 a 的值域；

x→f：U×A→V 是一个信息函数, 它为每个对象的每个属性赋予了一个信息值, 即：

$$\forall_a \in A, x\in U, f(x,a)\in V_a$$

信息系统也通常用 $S=(U,A)$ 来代替 $S=(U,A,V,F)$ 。

如果 A=C∪ D, C∩ D≠φ, 则可以将 $S=(U,A)$ 看作为信息表, C 中的属性作为条件, D 中的属性起决策作用。

在所有的属性当中, 有些属性很重要, 有些属性不太重要, 甚至有些属性基本没有任何作用。属性约简就是要找到没有任何作用以及作用非常小的属性并将它们剔除, 从而降低信息系统的复杂程度。

令 R 为一族等价关系, P∈R, 如果 ind (R) =ind (R-{P, }则) 称 P 在 R 中不起任何作用;反之称 P 在 R 中具有作用。假如每一个 P∈R 都在 R 中具有作用, 就证明 R 是独立的;相反说明 R 是不独立的。设 P⊆R, 假如 P 为独立的, 并且 ind (P) =ind (R) , 就可以将 P 看做 R 的约简。

当 R 的简约不止一个时, 可以将所有必要的约简组成一个集合, 定义为 R 的核, 可以用 core (R) 表示。 $core(R)=\bigcap red(R)$ ，式中 red(R)代表 R 的所有约简。

实际运用过程中, 每一个分类对其他分类的作用不同, 有的只是有关系, 有的却十分重要, 程度有很大不同, 因此, 还必须考虑约简与核的相对性问题。

令 P 和 Q 为 U 中的等价关系, Q 的 P 正域记为 Posp (Q) ,

即: $Pos_p(Q)=\bigcup_{x\in U/Q}P(X)$

假定 P 和 Q 具有等价关系, R∈P, 当 POS_{ind} (P) (ind (Q)) = $POS_{ind(p-(R))}$ (in 时 d (, Q 则)) 称 R 为 P 中 Q 不必要的;否则 R 为 P 中 Q 必要的。如果 P 中的每个 R 都为 Q 必要的, 则称 P 为 Q 独立的。

设 S∈P, S 为 P 的 Q 约简当且仅当 S 是 P 的 Q 独立子族且 POS_{ind} (ind (Q))

$= POS_{ind(p)}$ (in, dP (Q 的) Q) 约简称为相对约简。P 中所有的 Q 必要的原始关系构成的集合称为 P 的 Q 核，记为 core Q (P) 。

相对核与相对约简的关系如下: $core_Q(R) = \bigcap red_Q(R)$，其中 $red_Q(R)$ 是所有 P 的 Q 约简构成的集合。

为此，构建 S= (U, A, V, F) 为答复意见质量指标信息集合，对于等价关系 P⊆R 有分类 U ind (P) ={ $X_1, X_2, \cdots, X_n$ }，则 P 的信息量记为：

$$I(P) = \sum_{i=1}^{n} \frac{|X_i|}{|U|}[1 - \frac{|X_i|}{|U|}] = 1 - \frac{1}{|U|^2}\sum_{i=1}^{n}|X_i|^2$$

其中, |X|表示集合 X 的基数, $|X_i|$/|U|表示等价类 Xi 在 U 中的概率。

通过考察等价关系中蕴含的信息量，就能够确定在答复意见质量指标体系中，哪个指标重要性高，哪个指标重要性低,哪个指标基本没有作用。在信息集合 S= (U, A, V, F) 中，指标属性 a∈A 在指标属性集 A 中的重要度和属性的信息量的关系可以确定为：

$$sig_{A-\{a_i\}}(a_i) = I(A) - I(A - \{a_i\})$$

可以通过衡量去掉某个指标后信息系统信息量的变化情况来确定某项指标的重要程度。假如去掉某个指标后，信息系统的信息量具有显著的变化，说明该指标在整个信息系统中具有重要的作用，必须保留。反之就可以剔除。因此通过下式

$$W_i = \frac{sig_{A-\{a_i\}}(a_i)}{\sum_{i=1}^{n} sig_{A-\{a_i\}}(a_i)}$$

就可以计算出答复意见质量指标体系中所有指标的权重。

然后通过下式，设置合理阀值，依据偏离程度的量化，对指标进行约简。

$$D_i = \frac{W_{\max} - W_i}{W_{\max}} * 100\%$$

W_i 表示答复意见质量指标体系中第 i 个指标的权重; $W_{\max}$ 表示所有答复意见质量指标中权重最高的指标，$W_{\max} - W_i$ 表示答复意见质量指标体系中第 i 个指标相对于 $W_{\max}$ 指标的偏离程度, W_i 数值越低，说明指标 W_i 与指标 $W_{\max}$ 的偏离度越高。

6.2 答复意见评价体系

通过属性约简，得到最终的答复意见质量指标体系，在此基础上，采用 SVM 算法对答复意见质量进行仿真。

假设样本集 (x_i, y_i), 1≤i≤N, $x_i \in R^d$, $y_i \in$|-1,1|是类别符号, g (x) $= w^T x$+b 可以作为线性判别函数, $w^T x$+b=0 可以作为分类面方程, 当分类面满足以下要求时就能够确保对每一个样本进行分类。$y_i(w^T x_i + b) - 1 \geq 0, i = 1,2,...,n$。

定义如下函数：

$$L(w,b,\alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i [y_i (w^T x_i + b) - 1]$$

把上式分别对 w、b、α_i 求偏微分并令它们等于 0, 然后加上原约束条件可以把原问题转化为凸二次规划的对偶问题。凸二次规划的对偶问题可以计算出最佳解α_i^*, 将α_i^*不为零的样本称为支持向量。通过

$$\alpha_i [y_i (w^T x_i + b) - 1] = 0$$

可以计算出b^*，点积运算可以转化为

$$\Phi(x_i) \cdot \Phi(x_j) = K(x_i, x_j), K(x_i, x_j)$$

这就是核函数, 从而得到最终的 SVM 如下：

$$f(x) = \operatorname{sgn}(\sum_{i=1}^{n} a_i^* y_i x_i^* + b^*) = \operatorname{sgn}(\sum_{i=1}^{n} a_i^* y_i K(x_i, x) + b^*)$$

七、模型评估

7.1 模型优点

时代背景下，数据集合的增长速度远远超过了传统的手工分析技术所能处理的程度，通过该模型的建立可以及时地利用数据提供的信息，解决了人类的分析和抽象能力难以处理高维和海量数据的难题，有利于建立服务型、智慧型的政府，借助该模型的分析，政府可以有效率地处理群众的建议，并及时给予有效的反馈。而且，随着数据库和互联网的广泛应用，数据的规模呈指数的飞速增长，该模型还可以用于公司、学校收集员工、学生的意见,商业投资等场合，应用十分广泛。

7.2 模型缺点

在对大数据进行机器分类时，默认每个训练点在分类中所起的作用是一致的，但是由于初始训练点的很少或分类面发生迁移，将过去的点和现在的点相等的权值便不太合适。此外，人是在决策过程中的决策者，这就涉及到一个对热点话题的主观感受程度，而该模型中分类距离都是客观的，可以考虑引入效益函数来构造带主观因数的分类模型。

八、参考文献

[1] 朱晓华，浅析数据挖掘技术在图书馆自动化中的应用，图书官学研究，2002，（5）：41-45.

[2] https://blog.csdn.net/u010417185/article/details/80652233

[3] https://blog.csdn.net/wenxueliu/article/details/80275686

[4] 田艳．数据挖掘技术的应用及发展[J]．统计与信息论坛，2004，(7)：18—21.

[5] 朱世武．数据挖掘运用的理论与技术[J]．统计研究，2003．8：45—51．

[6] 黄解军，数据挖掘技术的应用研究。计算机工程与应用，2003，（2）：45-48.

[7] 沈清，汤霖，模式识别导论，长沙：国防科技大学出版社，1993.

[8] 阎平凡，黄端旭，人工神经网络一模型分析与应用，合肥：安徽教育出版社，1993.

[9] Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. In *EMNLP*

[10] Diyi Yang. Hierarchical Attention Networks for Document Classification. 2018

[11] https://blog.csdn.net/tcx1992/article/details/78194384

附录

问题一

编程软件：Python

程序源代码：

```
import numpy as np
import sys
sys.path.append("..")
import random

# file path
# dataset_path =
'/data/PycharmProjects/question_matching_framework/work_space/example/dataset/aaa'

def load_cn_data_from_files(classify_files):
    count = len(classify_files)
    x_text = []
    y = []
    for index in range(count):
        classify_file = classify_files[index]
        lines = list(open(classify_file, "r").readlines())
        label = [0] * count
        label[index] = 1
        labels = [label for _ in lines]
        if index == 0:
            x_text = lines
            y = labels
        else:
            x_text = x_text + lines
            y = np.concatenate([y, labels])
    x_text = [clean_str_cn(sent) for sent in x_text]
    return [x_text, y]

def clean_str_cn(string):
    """
    Tokenization/string cleaning for all datasets except for SST.
    Original taken from
https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
    """
    return string.strip().lower()

def load_data(classify_files, config, sort_by_len=True, enhance = True, reverse=True):
    x_text, y = load_cn_data_from_files(classify_files)
```

```
new_text = []
if reverse == True:
    for text in x_text:
        text_list = text.strip().split(' ')
        text_list.reverse()
        reversed_text = ' '.join(text_list)
        new_text.append(reversed_text)
    x_text = new_text
else:
    pass

y = list(y)
original_dataset = list(zip(x_text, y))
if enhance == True:
    num_sample = len(original_dataset)
    # shuffle
    for i in range(num_sample):
        text_list = original_dataset[i][0].split(' ')
        random.shuffle(text_list)
        text_shuffled = ' '.join(text_list)
        label_shuffled = original_dataset[i][1]
        x_text.append(text_shuffled)
        y.append(label_shuffled)
else:
    pass
# Randomly shuffle data
shuffle_indices = list(range(len(y)))
random.shuffle(shuffle_indices)
# print(shuffle_indices)
x_shuffled = []
y_shuffled_tmp = []
for shuffle_indice in shuffle_indices:
    x_shuffled.append(x_text[shuffle_indice])
    y_shuffled_tmp.append(y[shuffle_indice])
y_shuffled = np.array(y_shuffled_tmp)
# train_set length
n_samples = len(x_shuffled)
# shuffle and generate train and valid data set
sidx = np.random.permutation(n_samples)
n_train = int(np.round(n_samples * (1. - config.valid_portion)))
print("Train/Test split: {:d}/{:d}".format(n_train, (n_samples - n_train)))
valid_set_x = [x_shuffled[s] for s in sidx[n_train:]]
valid_set_y = [y_shuffled[s] for s in sidx[n_train:]]
```

```
    train_set_x = [x_shuffled[s] for s in sidx[:n_train]]
    train_set_y = [y_shuffled[s] for s in sidx[:n_train]]
      train_set = (train_set_x, train_set_y)
    valid_set = (valid_set_x, valid_set_y)
    # test_set = (x_test, y_test)
    # test_set_x, test_set_y = test_set
    valid_set_x, valid_set_y = valid_set
    train_set_x, train_set_y = train_set
    def len_argsort(seq):
        return sorted(range(len(seq)), key=lambda x: len(seq[x]))
    if sort_by_len:
         sorted_index = len_argsort(valid_set_x)
        valid_set_x = [valid_set_x[i] for i in sorted_index]
        valid_set_y = [valid_set_y[i] for i in sorted_index]
        sorted_index = len_argsort(train_set_x)
        train_set_x = [train_set_x[i] for i in sorted_index]
        train_set_y = [train_set_y[i] for i in sorted_index]
    train_set=(train_set_x,train_set_y)
    valid_set=(valid_set_x,valid_set_y)

    max_len = config.num_step
     def generate_mask(data_set):
        set_x = data_set[0]
        mask_x = np.zeros([max_len, len(set_x)])
        for i,x in enumerate(set_x):
            x_list = x.split(' ')
            if len(x_list) < max_len:
                mask_x[0:len(x_list), i] = 1
            else:
                mask_x[:, i] = 1
        new_set = (set_x, data_set[1], mask_x)
        return new_set
     train_set = generate_mask(train_set)
    valid_set = generate_mask(valid_set)
    train_data = (train_set[0], train_set[1], train_set[2])
    valid_data = (valid_set[0], valid_set[1], valid_set[2])
    return train_data, valid_data
def batch_iter(data,batch_size, shuffle = True):
    x, y, mask_x = data
    mask_x = np.asarray(mask_x).T.tolist()
    data_size = len(x)
    if shuffle:
        shuffle_indices = list(range(data_size))
        random.shuffle(shuffle_indices)
```

```
            shuffled_x = []
            shuffled_y = []
            shuffled_mask_x = []
            for shuffle_indice in shuffle_indices:
                shuffled_x.append(x[shuffle_indice])
                shuffled_y.append(y[shuffle_indice])
                shuffled_mask_x.append(mask_x[shuffle_indice])
        else:
            shuffled_x = x
            shuffled_y = y
            shuffled_mask_x = mask_x
        shuffled_mask_x = np.asarray(shuffled_mask_x).T    # .tolist()
        shuffled_x = np.array(shuffled_x)
        shuffled_y = np.array(shuffled_y)
        shuffled_mask_x = np.array(shuffled_mask_x)
        num_batches_per_epoch = data_size // batch_size
        for batch_index in range(num_batches_per_epoch):
            start_index=batch_index*batch_size
            end_index=min((batch_index+1)*batch_size,data_size)
            return_x = shuffled_x[start_index:end_index]
            return_y = shuffled_y[start_index:end_index]
            return_mask_x = shuffled_mask_x[:,start_index:end_index]
            yield (return_x,return_y,return_mask_x)
```

问题二

编程软件：Python

程序源代码：

```
x_embedded = wv.embedding_lookup(len(list(x)), config.num_step, config.embed_dim,
list(x), 0)
import inspect
import tensorflow as tf
class RNN_Model(object):
    def __init__(self, config, num_classes, is_training=True):
        keep_prob = config.keep_prob
        batch_size = config.batch_size
        num_step = config.num_step
        embed_dim = config.embed_dim
        self.embedded_x = tf.placeholder(tf.float32, [None, num_step, embed_dim],
name="embedded_chars")
        self.target = tf.placeholder(tf.int64, [None, num_classes], name='target')
        self.mask_x = tf.placeholder(tf.float32, [num_step, None], name="mask_x")

        hidden_neural_size=config.hidden_neural_size
        hidden_layer_num=config.hidden_layer_num
```

```
            def lstm_cell():
                if 'reuse' in
inspect.signature(tf.contrib.rnn.BasicLSTMCell.__init__).parameters:
                    return tf.contrib.rnn.BasicLSTMCell(hidden_neural_size,
reuse=tf.get_variable_scope().reuse)
                else:
                    return tf.contrib.rnn.BasicLSTMCell(
                        hidden_neural_size, forget_bias=0.0, state_is_tuple=True)
            attn_cell = lstm_cell
            if is_training and keep_prob < 1:
                def attn_cell():
                    return tf.contrib.rnn.DropoutWrapper(
                        lstm_cell(), output_keep_prob=config.keep_prob)
            cell = tf.contrib.rnn.MultiRNNCell(
                [attn_cell() for _ in range(hidden_layer_num)], state_is_tuple=True)
              self._initial_state = cell.zero_state(batch_size, dtype=tf.float32)
            inputs = self.embedded_x
            if keep_prob < 1:
                inputs = tf.nn.dropout(inputs, keep_prob)
            out_put = []
            state = self._initial_state
            with tf.variable_scope("LSTM_layer"):
                for time_step in range(num_step):
                    if time_step > 0: tf.get_variable_scope().reuse_variables()
                    (cell_output, state) = cell(inputs[:, time_step,:],state)
                    out_put.append(cell_output)
            out_put=out_put*self.mask_x[:,:,None]
            with tf.name_scope("mean_pooling_layer"):
                out_put = tf.reduce_sum(out_put,0)/(tf.reduce_sum(self.mask_x,0)[:,None])
            with tf.name_scope("Softmax_layer_and_output"):
                softmax_w =
tf.get_variable("softmax_w",[hidden_neural_size,num_classes],dtype=tf.float32)
                softmax_b = tf.get_variable("softmax_b",[num_classes],dtype=tf.float32)
                # self.logits = tf.matmul(out_put,softmax_w)
                # self.scores = tf.add(self.logits, softmax_b, name='scores')
                self.scores = tf.nn.xw_plus_b(out_put, softmax_w, softmax_b, name="scores")
            with tf.name_scope("loss"):
                self.loss = tf.nn.softmax_cross_entropy_with_logits(labels=self.target,
logits=self.scores + 1e-10)
                self.cost = tf.reduce_mean(self.loss)
            with tf.name_scope("accuracy"):
                self.prediction = tf.argmax(self.scores, 1, name="prediction")
                correct_prediction = tf.equal(self.prediction, tf.argmax(self.target, 1))
                self.correct_num = tf.reduce_sum(tf.cast(correct_prediction, tf.float32))
```

```
                self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),
name="accuracy")
                self.probability = tf.nn.softmax(self.scores, name="probability")
            loss_summary = tf.summary.scalar("loss", self.cost)
            accuracy_summary = tf.summary.scalar("accuracy_summary", self.accuracy)
            if not is_training:
                return
            self.global_step = tf.Variable(0, name="global_step", trainable=False)
            self.lr = tf.Variable(0.0, trainable=False)
            tvars = tf.trainable_variables()
            grads, _ = tf.clip_by_global_norm(tf.gradients(self.cost, tvars),
config.max_grad_norm)
            grad_summaries = []
            for g, v in zip(grads, tvars):
                if g is not None:
                    grad_hist_summary =
tf.summary.histogram("{}/grad/hist".format(v.name), g)
                    sparsity_summary = tf.summary.scalar("{}/grad/sparsity".format(v.name),
tf.nn.zero_fraction(g))
                    grad_summaries.append(grad_hist_summary)
                    grad_summaries.append(sparsity_summary)
            self.grad_summaries_merged = tf.summary.merge(grad_summaries)
            self.summary =
tf.summary.merge([loss_summary,accuracy_summary,self.grad_summaries_merged])
            optimizer = tf.train.GradientDescentOptimizer(self.lr)
            optimizer.apply_gradients(zip(grads, tvars))
            self.train_op=optimizer.apply_gradients(zip(grads, tvars))
            self.new_lr = tf.placeholder(tf.float32,shape=[],name="new_learning_rate")
            self._lr_update = tf.assign(self.lr,self.new_lr)
        def assign_new_lr(self,session,lr_value):
            session.run(self._lr_update,feed_dict={self.new_lr:lr_value})
```