技术问答社区问题中的文本匹配策略

摘要

技术问答社区内文本量的激增给运维人员带来了不小的挑战。其中, 存在一些重复提出的问题, 但由于技术性平台内容的特殊性使得重复问题的信息检索较为困难。问题要求能够对于给定的问题对预测重复概率, 并输出对于某一特定问题重复概率值排名前十的文本，属于典型的文本匹配问题. 自然语言处理中对文本匹配的研究由来已久，其研究方法大致分为三类，每一类方法都可以对该问题独立求解。

模型一使用了传统的余弦相似度法, 将自然文本向量化构建词向量并求解余弦相似度作为其分类概率，简单快捷但效果未达到预期。

模型二使用了文本特征分析与 XGBoost 方法，其中 XGBoost 是时下效果非常良好的一种机器学习算法，准确度高而且鲁棒性强。文本匹配本质上是针对给定文本对判定重复或非重复的二分类问题, 应用宏观层面的文本特征并构建机器学习分类器能够很好地解决问题，相比于传统模型实现了一定程度上的提升。

模型三使用了引入深度学习之后的自然语言处理方法，采取 GRU-CNN 网络架构。该模型将循环神经网络和卷积神经网络结合起来提取文本的特征，并创新性地参考迁移学习，引入“预训练-再训练”策略训练神经网络，使网络效果达到一个极优的水平。以文本的词向量作为输入，预测文本对的准确率和 F1 分数均突破 90%，为三个模型中表现最好的模型。

三种模型分别代表文本匹配问题的三类思想策略，在效果上各有千秋，而三种模型之间的对比更能体现深度学习方法在自然语言处理问题中的重要地位。

关键字: 词向量　余弦相似度　XGBoost　GRU-CNN 网络

目录

一、 问题重述

1.1 引言

目前，问答平台上的问题标重主要依靠用户人工辨别。平台用户会对疑似重复的问题进行投票标记，然后平台内的管理员和资深用户（平台等级高的用户）对该问题是否被重复提问进行核实，若确认重复则打上重复标签。该过程较为繁琐，依赖用户主观判断，存在时间跨度大、工作量大、效率低等问题，增加了用户的工作量且延长了新用户寻求答案所需的时间。因而，如能建立一个检测问题重复度的模型，通过配对新提出问题与文本库中现存问题，找出重复的问题组合，就能提高重复问题标记效率，提高平台问题的文本质量，减少问题冗余。同时，平台用户也能及时地根据重复标签提示找到相关问题并查看已有的回复。

1.2 问题的提出

附件给出了问答平台上问题的文本内容记录，以及比较两个问题之间是否重复的数据集。请根据附件给出的问题文本数据及问题配对信息，建立一个能判断问题是否重复的分类模型，并解决：

1. 输出样本问题组为重复问题的概率；
   通常使用 F1-score 对分类模型进行评价：
   $$F_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2P_iR_i}{P_i + R_i}$$
   其中 P_i 为第 i 类的查准率，R_i 为第 i 类的查全率；
2. 从附件问题列表中，给出与目标问题重复概率最大的前 10 个问题的编号；
   对于每个问题的预测结果采用 **top K** 列表对其进行评估，评估公式如下：
   $$R = \frac{N_{detected}}{N_{total}}$$
   其中 $N_{detected}$ 为在 **top K** 列表结果中正确检测到的重复问题编号数量，N_{total} 为该样本实际拥有的重复问题数量。评估时 K 取 10，若样本中无重复问题则不会计分。

二、 模型假设

1. 余弦相似度模型中，概率大于 0.5 的预测为 1（即重复），小于 0.5 的判断为 0（不重复），从而计算 F1-score 值。
2. 构建词袋并进行向量化的过程中可以将停用词摘除后再处理来削弱无关信息的作用。

3. 在进行机器学习的过程中需要不重复的数据对，但不重复的数据对组合太多，这里为了防止造成类别不平衡故可以在不重复的数据对中采集一定量样本。

三、 符号说明

符号	意义
P_i	第 i 类的查准率
R_i	第 i 类的查全率
F_1	F_1 分数
v_i	语句 i 对应的词向量
O	XGBoost 的训练损失函数
$L()$	均方误差函数
$f_{t-1}(x)$	残差函数
$h_t(x)$	学习的基学习器
$\Omega()$	正则化项
$C\square\gamma$	常数参数
T	树的深度
w_j	节点的权重分数

四、 模型一: 词向量余弦相似度模型

4.1 词向量

词的向量化 [1] 是将语言中的词进行数学化，也即把一个词表示成一个向量。词的向量化表示主要有离散表示与分布式表示两种表示方式。其中，离散表示是传统的基于规则或基于统计的自然语义处理方法将单词看作一个原子符号的表示方式，分布式表示是将词映射到一个低维、稠密的实数向量空间中的表示方式。目前较为常用的分布式表示模型有 Distributed representation 与 word2vec 模型等。在词的分布式表示中，词之间存在距离关系，词义越相近的词在空间的距离越近，因此可以使用距离向量公式比较词向量之间的相似度。本文通过抽取问题样本组成样本对，应用“词袋模型 + 截断降

维”将问题样本向量化表示，并求出各向量之间的距离进行相似度的比较，从而判断问题间是否重复。我们根据附件 1 中的所有英文文本构建词频统计的词袋（相比中文自然语言处理英文文本有一定优势），然后对词袋按照词频进行降序排序。根据每个单词在词袋中出现的位置标号，将目标文本的每个单词替换为对应的标号，实现自然文本的数值化。考虑到模型三中应用循环神经网络，不定长的向量输入可能会导致梯度爆炸等问题，故我们利用截断降维的方法将不定长的向量映射到一个 20 维的向量便于计算求解。

4.2 余弦相似度

余弦相似度是传统数学建模中一种常见的相似度估计算法，是用两个向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小。这对于已经向量化是合理的并且是较为有效的。具体的说，当余弦值越接近 1，就表明两个夹角越接近 0 度，也就意味着两个向量越相似。

而对于本题来说，由于此前已经将问题文本向量化成指定 20 维度的向量，因此，计算两个词向量 v_i, v_j 相似度的公式为

$$\begin{aligned}\cos\theta &= \frac{\sum_{k=1}^{20}(v_{ik}\times v_{jk})}{\sqrt{\sum_{k=1}^{20}(v_{ik})^2}\times\sqrt{\sum_{k=1}^{20}(v_{jk})^2}} \\ &= \frac{\boldsymbol{v_i}\cdot\boldsymbol{v_j}}{|\boldsymbol{v_i}|\times|\boldsymbol{v_j}|}\end{aligned} \tag{1}$$

在此模型中，因为词向量的每个维度都为正值，因此，我们合理地将余弦相似度作为重复概率的估计，借此计算出每个文本问题与其他所有问题的问题对的重复概率。

在第一问中，我们需要计算出样本问题组重复问题的概率，这需要我们计算出每个问题与其他七千多个问题的重复概率，因此实际上有 $C(7294,2)=26597571$ 个问题对，但是考虑到重复概率小于 0.5 的问题对数目 (我们将之近似认为不重复) 远远大于重复概率大于 0.5 的问题对数目。因此在此文中给出计算出来的高重复概率的问题对数据部分截图如1所示。

现对用模型一得到的问题一结果进行评估，这里运用 F1-score 对分类模型进行评价。

$$\begin{aligned}F_1 &= \frac{1}{2}\sum_{i=1}^{2}\frac{2P_iR_i}{P_i+R_i} \\ &= \frac{0.49+0.45}{2} \\ &= 0.47\end{aligned}$$

这里，0.49 数据是将重复问题对作为正样本，将不重复问题作为负样本得到的结果。而 0.45 是将不重复问题作为正样本，将重复问题作为正样本得到的结果。两者求其平均得

	问题1id	问题2id	cosine_distance
2	63022	62860	0.890190427
3	46866	46762	0.448353276
4	97939	80376	0.546684258
5	23129	10788	0.739649651
6	50415	25518	0.988984153
7	60905	57067	0.77908438
8	75289	14542	0.932042625
9	99850	24442	0.485709306
10	112238	25494	0.936870188
11	102106	37787	0.992076338
12	109853	30084	0.708264027
13	72393	26892	0.707652137
14	103709	9020	0.653012339
15	92748	58680	0.913620987
16	89232	78510	0.937739303
17	28406	25936	0.782726793
18	110551	4376	0.447469097
19	55129	54291	0.781467422
20	59203	57642	0.862236948

图 1　模型一第一问求解

到最终的 F_1 ，如表1所示。

表 1　模型一对问题一的回答

类别	查准率	查全率	F1 分数
不重复	0.89	0.34	0.49
重复	0.31	0.87	0.45
平均值	0.60	0.61	0.47
准确率			0.47

对于问题 2，目标样本遍历在附件 1 中的文本形成一系列文本对，计算余弦相似度作为概率。这里由于词向量的每个分量都为非负数，所以余弦相似度的计算结果在 (0，1) 之间，越接近 1 则两向量夹角越小，两文本越相似。以 id 为 15565 的文本为例，有两个文本 (id=6968，id=7374) 都与它形成了重复关系。但通过余弦相似度计算，排名前 10 的文本见表2

表 2　模型一对问题二的回答

文档 id	概率值	实际是否已知重复
73444	0.793	否
110394	0.797	否
113165	0.797	否
5279	0.801	否
97741	0.802	否
95453	0.812	否
85993	0.814	否
59573	0.821	否
37702	0.822	否
8140	0.856	否

最后求解的评价分数为

$$R_2 = \frac{0}{2} = 0 \tag{2}$$

这说明传统的余弦相似度模型效果并不理想。我们可以将文本的一些特征例如余弦相似度做二次加工，将其视作文本对的属性来训练一个机器学习模型进行分类。于是，我们提出了基于机器学习方法的模型二:

五、 模型二: 文本特征与 XGBoost 模型

基于此前的词向量数据，我们对效果欠佳的传统余弦相似度模型进行优化，引入机器学习中 XGBoost 模型的概念和原理对该问题进行建模。

5.1 文本特征

在使用机器学习之前，我们需要引入文本特征这一概念。在此问题中，我们对问题文本提取处三个方面的特征: 基本特征，词袋特征和距离特征。

基本特征不需要基于词向量，它包括问题文本的长度，问题对文本的长度差，对问题文本去掉空白符号后的字符数量，问题文本的单词数量，问题对中公用单词数目等。刻画的是问题文本字符串方面的属性特征。

词袋特征是基于语频和语义对问题对进行提取的特征，它主要包括三个步骤:

1. 利用 sklearn 库中的 CountVectorizer 将每一个问题对构建一个词袋向量，向量中的每个维度是词频数的独热表示。将该向量作为行构建问题对的稀疏矩阵。
2. 利用 tf-idf 算法构建向量
   (a) 计算词频 tf，计算出某词在该问题对中出现的词频率
   (b) 计算逆文档频率 idf。此处我们将按比例抽样出的共 4000 余个问题对当作语料库，用来模拟词汇的使用环境。
   (c) 计算 tf-idf。$tf - idf = tf \times idf$，可以看出该值与一个词在文档中出现次数成正比，与该词在整个语言中的出现次数成反比。

   由此构建出高维度的向量，每一个维度为某单词的 tf-idf 值。将每一个高维向量作为行构建整个问题对的稀疏矩阵。
3. 利用 SVD 对步骤 2 得到的稀疏矩阵 X 进行降维。
   (a) 对 X 进行奇异值分解 $X = U\Sigma V^T$
   (b) 取前 180 维奇异值，构建 180 维度的向量
   (c) 将该向量对原矩阵进行相乘，得到新的加权矩阵 X'，
   (d) 截取每一行向量的前 180 维度得到一个列数为 180，行数为问题对数的降维后的矩阵 Y。

距离特征则是对上述两个特征的补充，上述特征是对文本对的向量化，而距离特征则是对单个文本的向量化后的再处理，首先对整个数据集构建词袋进行词频统计，根据统计，根据统计结果将文本中每个单词变成在词袋中对应的下标，最后经过截断降维映射到一个 20 维度的向量中，也即利用模型一中的词向量进行相似度分析，这里除了模型一的 cosine 距离，还增加了诸如 cityblock_distance，jaccard_distance 等的举例指标作为距离特征成员。

5.2 XGBoost 算法

GBDT 是一种基于集成思想下的 Boosting 学习器，并采用梯度提升的方法进行每一轮的迭代最终组建出强学习器。而 XGBoost 是 GBDT 的高效实现，XGBoost 中的基学习器除了可以是 CART（GBTree）也可以是线性分类器（GBLinear）。

XGBoost 最大的特点在于它能够自动利用 CPU 的多线程进行并行计算，同时在算法上加以改进提高了精度 [2]。在目标函数中显示的加上了正则化项，基学习为 CART 时，正则化项与树的叶子节点的数量 T 和叶子节点的值有关。这使得模型在保证了准确率的同时还具有比传统 GBDT 更强的鲁棒性。

XGBoost 学习的损失函数为下式：

$$O = \sum_{i=1}^{n} L(y_i, f_{t-1}(x) + h_t(x)) + \Omega(h_t(x)) + C \tag{3}$$

在式子中，$\Omega(h_t(x))$ 为正则化项，通常是树的复杂度（与树的深度 T 和节点权重 w 有关）：

$$\Omega(h_t(x)) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{4}$$

对它进行泰勒公式的三阶展开近似：

$$O = \sum_{i=1}^{n} (L(y_i, f_{t-1}(x)) + u_i h_t(x_i) + \frac{1}{2} v_i h_t^2(x_i)) + \Omega(h_t(x)) + C \tag{5}$$

其中：

$$u_i = \frac{\partial L(y, f(x_i))}{\partial f(x_i)}, v_i = \frac{\partial^2 L(y, f(x_i))}{\partial f(x_i)} \tag{6}$$

这一处理方法使得学习器的学习过程更加迅速，避免了拟合一个过于复杂的函数带来过高的计算复杂性。

XGBoost 的核心思想与主要流程可以总结如下：

1. 不断加树特征分裂增长一棵树，每增加一棵树实际上就是学习一个新的基学习器
2. 训练完成得到 k 棵树，要预测一个样本分数实际上就是根据样本特征在每棵树中给对应的一个叶子节点。每个叶子节点对应一个分数。树的生成是节点一分为二，贪心策略最优划分。
3. 最后只需要把每棵树对应的分数加起来就是预测值。

5.3 XGBoost 训练结果

分别利用每一类特征求得结果

我们对文档的基本特征、词袋特征和距离特征分别训练 XGBoost 分类器进行学习。由于词袋模型的各属性之间存在高度关联，经过反复实验发现，词袋模型中基于 tf-idf 并使用截断奇异值分解生成的词向量用于训练效果最佳。而基本特征和距离特征将使用其中的全部距离。训练的准确率和 F1 分数如表3所示：

表 3　每一类特征训练准确率与 F1 分数

特征	准确率	F1 分数
基本特征	0.82	0.71
基于词频的词袋特征	0.77	0.63
基于 tf-idf 与 SVD 的词袋特征	0.76	0.54
距离特征	0.74	0.55

可以看到，每种模型单独使用都还不够理想。下一步，考虑将特征进行组合：

利用三种特征求解

这里为了避免维度过高使得训练数据集过于稀疏，选取文档的基本特征、基于 tf-idf-SVD 的特征与距离特征进行学习器的构建。实验证明，这一策略是所有特征组合中效果最优的，分类结果如表4所示：

表 4　利用三种特征联合训练所得结果

类别	查准率	查全率	F1 分数
不重复	0.82	0.97	0.89
重复	0.83	0.38	0.53
平均值	0.83	0.68	0.71
准确率			0.82

可以看到，最终训练得到的 F1 分数为 0.71，准确率 0.82 为所有 XGBoost 分类器中最高的。XGBoost 分类器的基分类器可以可视化，由于结构过于庞大，仅展示部分结构如图2所示：

对于问题一，给定两个问题，通过计算它们的特征作为属性数据交由 XGBoost 分类器求解，即可预测出二者概率。

而对于问题 2 输出 top10 的文本，给定一个问题，另一个配对问题则在原始附件中遍历形成文本对数据集以后再进行预测。例如，以 id 为 15565 的文本为例，它在附件 1 中是第 141 号文件。经过分类器遍历预测以后输出的 top-10 文档如表5。

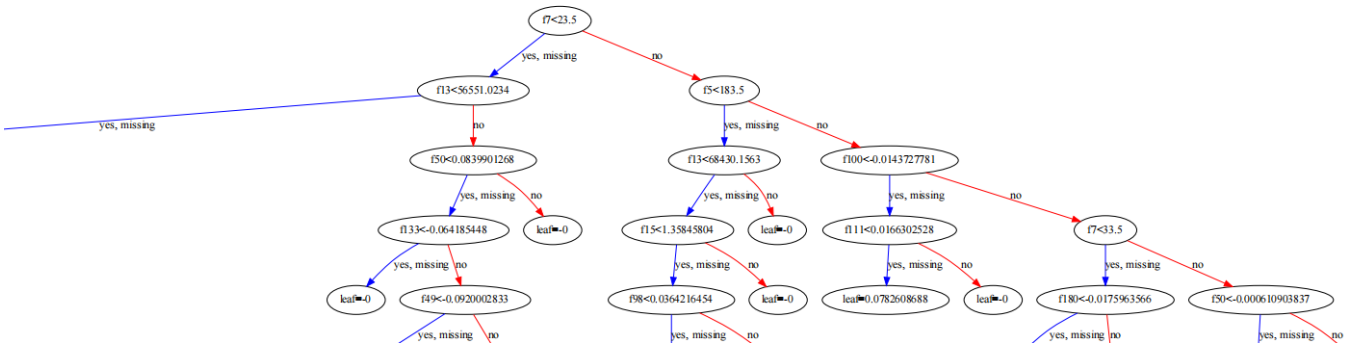

图 2　XGBoost 可视化

表 5　模型二对问题 2 的回答

文档 id	概率值	实际是否已知重复
28749	0.981	否
104607	0.981	否
67966	0.982	否
60385	0.983	否
56833	0.983	否
32024	0.984	否
63516	0.987	否
10449	0.987	否
72051	0.989	否
44718	0.992	否

最后求解的评价分数为

$$R_2 = \frac{0}{2} = 0 \tag{7}$$

这说明尽管准确率提升了，模型预测仍然存在一定问题。这也与重复样本对在所有组合总数中占比过少导致类别失衡有关。为解决这一问题，我们考虑将眼光放到深度学习方法中来。

六、 模型三: GRU-CNN 模型

6.1 模型架构

GRU（Gate Recurrent Unit）和 LSTM（Long-Short Term Memory）一样，也是为了解决长期记忆和反向传播中的梯度等问题而提出来的。Cho 等人在 2014 年进一步提出了更加简单的 GRU 模型 [3]。GRU 将 LSTM 中的遗忘门和输入门合并为了一个单一的更新门，还混合了细胞状态和隐藏状态，计算当前时刻新信息的方法和 LSTM 有所不同。经过实验，使用 GRU 的效果比使用 LSTM 更优一些。

模型的基本架构如图3。

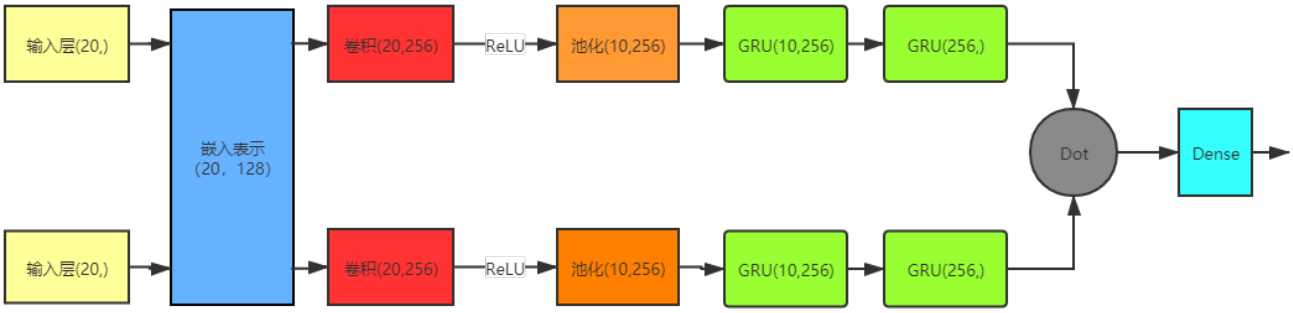

图 3　GRU-CNN 模型的架构图

输入的两个序列为词向量的形式，每个词向量为 20 维。随后经过的第一层为嵌入表示，这一层能够将离散的向量连续化从而描述词向量的特征信息。

第二层为并行的卷积层与池化层。同二维的卷积神经网络一样，一维卷积同样是对序列的特征进行抽象最终保留更高级别的特征。

第三层是两条两个连续的 GRU 并行处理，改变数据维度从而便于处理。

最后，将两条路径中的词向量点乘综合并交由最后的 Dense 层输出，以 sigmoid 函数为激活函数即可获得分类概率。

6.2 训练方法与效果

使用的训练方法为“预训练 + 再训练”，这一策略对神经网络的精度提升有很大作用。我们发现，在数据集上训练 20 轮准确率仍然达不到预期而且容易出现过拟合。为解决过拟合，我们尝试过 Drop-Out、Early-Stopping 等方法，并将数据集打乱以后再进行数据集划分与训练。但出乎意料的是，打乱以后的数据集上训练模型并没有表现出更好的泛化性能，反而加重了模型的过拟合程度。

为解决这一问题，我们参考了迁移学习的思想。迁移学习思想被广泛应用于图像分类，常常在 ImageNet 等大型网络中先预训练得到权重，然后以这个训练结果为初始值

在自己的数据集上再训练，能够迅速训练且获得高准确率的模型 [4]。这里我们先在原数据集进行预训练 10 个 epoch，然后将预训练得到的网络再次训练，得到精度表现相当好的神经网络，该网络的训练损失和训练准确度如图4和图5所示。

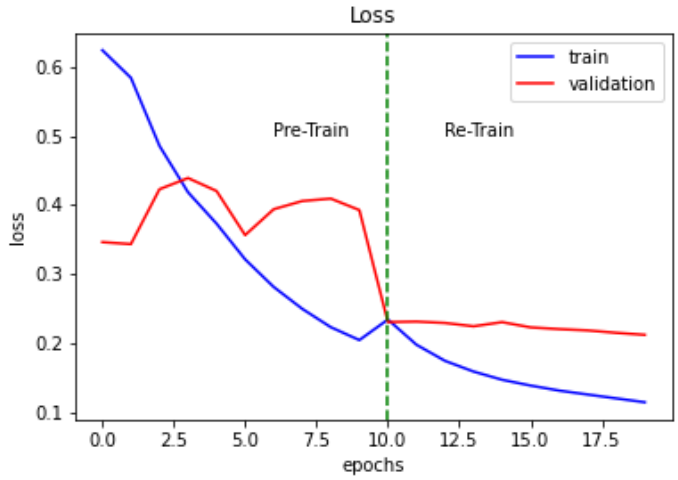

图 4　训练损失

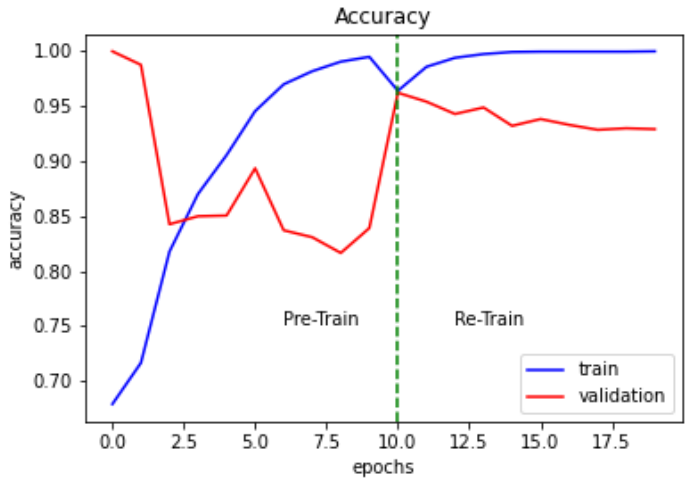

图 5　训练准确率

最终在数据集上测试的准确率达 94%，是到目前为止表现最好的模型。

6.3 模型求解与结果

利用这一神经网络在测试数据集上进行测试，相关数据如表6所示。

表 6　模型三对问题一的评估

类别	查准率	查全率	F1 分数
不重复	0.97	0.95	0.96
重复	0.84	0.91	0.88
平均值	0.91	0.93	0.92
准确率			0.94

F1 分数达到 0.92，表现已经比较良好。对于问题 1，两个文本形成的问题组被转换为两个 20 维的向量以后可以进行训练得到预测概率。例如，输入文本 “I have a directory with about 10 polygon shapefiles” 与 “I was wondering if anyone can help with batching a dissolve。”，预测为重复的概率为 0.0832。

问题 2 输出 top-10 的文档与概率，思想与上面一样，对于附件 1 中每个文本，在附件 1 中遍历式预测求解即可求解概率。仍然以 id 为 15565 号文本为例，预测结果如表7所示。

表 7　模型三对问题二的回答

文档 id	概率值	实际是否已知重复
66500	0.789	否
3924	0.794	否
6968	0.800	是
94252	0.802	否
1336	0.804	否
57437	0.804	否
7468	0.806	否
7274	0.809	是
41074	0.810	否
4452	0.814	否

最后求解的评价分数为

$$R_2 = \frac{2}{2} = 1 \tag{8}$$

很显然，相比于前两个模型，此模型的效果为最优。

七、 模型评价与改进

7.1 模型的评价

本文一共使用了三种不同的模型，分别代表文本相似度分类问题中三种经典的方法。第一种方法余弦相似度模型，是基于词向量的余弦距离提出的概念，计算简便。主要的思想难点仅在于如何将文本向量化，这里使用的是词袋模型。但不足之处在于准确度低。

第二种方法文本特征配合 XGBoost，使用文本更宏观层面的特征进行二次加工。XGBoost 是一种高效鲁棒的机器学习算法，无需我们手动进行复杂的数学计算，使用简便而且准确率高。这一方法相比于传统模型效果有一定的提升但还不够。

第三种方法是引入深度学习后的自然语言处理，是时下最火热的算法。网络结构经过对比，选择 GRU 而非 BiLSTM，经过 “预训练 + 再训练” 收获了非常良好的效果。所以，就准确度而言这一方法是最优方法。并且，基于神经网络的算法有一个优势就在于可以并行化计算，利用 GPU 算力充分利用可计算资源，提升速度。不足之处在于模型的训练比前两种而言略困难一些。

表8为三种方法的效果对比：

表 8　三种模型的效果比较

模型	准确率	F1 分数	训练时间
余弦相似度	0.47	0.47	0.28s
XGBoost	0.82	0.71	3.25s
GRU-CNN	0.94	0.92	519s

7.2 对模型的改进

我们认为，尽管三种模型经过逐步推进，其效果达到螺旋式上升，我们的方法仍有一些不足之处可以改进：

1. 在文本的向量化操作上，模型二中使用到了基于 TF-IDF 和 SVD 降维的方法，不过是针对文本对的向量化。实际上这一方法可以广泛应用在单个文本的向量化过程中，

采取截断降维的方式丢失了一些可能重要的信息从而导致模型能力较弱，所以可以将三个模型的向量化处理均替换为 TF-IDF-SVD 模式。

2. 为了防止类别失衡问题故采用了组合抽取的方式构建机器学习数据集。事实上如果有足够的时间和计算力的情况下可以将 $C(7294, 2)$ 个组合全部使用上，对那些已知重复的数据对进行数据增强，这样也可以在一定程度上削弱类别失衡的影响且数据量更大
3. 近年来，注意力（Attention）机制被广泛应用到基于深度学习的自然语言处理 (NLP) 各个任务中 [5]，我们猜想在模型三的网络中引入注意力机制能够使得模型的效果更好。

八、 参考文献与引用

参考文献

[1] 唐明, 朱磊, 邹显春. 基于 Word2Vec 的一种文档向量表示 [J]. 计算机科学,2016,43(06):214-217+269.

[2] Chen T , Tong H , Benesty M . xgboost: Extreme Gradient Boosting[J]. 2016.

[3] Dey R , Salemt F M . Gate-variants of Gated Recurrent Unit (GRU) neural networks[C]// IEEE International Midwest Symposium on Circuits Systems. IEEE, 2017:1597-1600.

[4] S. J. Pan and Q. Yang, ”A Survey on Transfer Learning,” in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.

[5] Vaswani A , Shazeer N , Parmar N , et al. Attention Is All You Need[J]. arXiv, 2017.

附录 A　代码

编程环境：python3.8.2+jupyter notebook GPU: GEFORCE GTX 1080, CPU: Intel i7,Operating System: Windows 10 论文编写工具：Overleaf(LaTeX)

模型一 & 模型二

```
import pandas as pd
import numpy as np
import gensim
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import TruncatedSVD
from scipy.stats import skew, kurtosis
from scipy.spatial.distance import cosine, cityblock, jaccard, canberra, euclidean, minkowski,
    braycurtis
from scipy.stats import skew, kurtosis
import gensim
from gensim.models import Word2Vec
from nltk import word_tokenize
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
import scipy
from tqdm import tqdm_notebook
%matplotlib inline

df = pd.read_csv('文本匹配数据集.csv')
df.columns=['question1','question2','is_duplicate']
df

# 计算question1和question2的长度
df['len_q1'] = df.question1.apply(lambda x: len(str(x)))
df['len_q2'] = df.question2.apply(lambda x: len(str(x)))

# 计算两个问题的长度差
df['diff_len'] = df.len_q1 - df.len_q2

# 计算两个问题的字符数量
df['len_char_q1'] = df.question1.apply(lambda x: len(''.join(set(str(x).replace(' ', '')))))
df['len_char_q2'] = df.question2.apply(lambda x: len(''.join(set(str(x).replace(' ', '')))))

# 计算两个问题的单词数
df['len_word_q1'] = df.question1.apply(lambda x: len(str(x).split()))
df['len_word_q2'] = df.question2.apply(lambda x: len(str(x).split()))

# 计算两个问题的常用单词数
df['common_words'] = df.apply(lambda x:
```

```
    len(set(str(x['question1']).lower().split()).intersection(
      set(str(x['question2']).lower().split()))), axis=1)
df

fs_basic = df[['len_q1', 'len_q2', 'diff_len', 'len_char_q1',
       'len_char_q2', 'len_word_q1', 'len_word_q2',
       'common_words']]

count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
count_vect.fit(pd.concat((df['question1'],df['question2'])).unique())
trainq1_trans = count_vect.transform(df['question1'].values)
trainq2_trans = count_vect.transform(df['question2'].values)
fs_bow = scipy.sparse.hstack((trainq1_trans,trainq2_trans))
fs_bow.shape

tfidf_vect = TfidfVectorizer(analyzer='word',
min_df=3,token_pattern=r'\w{1,}',ngram_range=(1,2),max_features=5000)
tfidf_vect.fit(pd.concat((df['question1'],df['question2'])).unique())
trainq1_trans = tfidf_vect.transform(df['question1'].values)
trainq2_trans = tfidf_vect.transform(df['question2'].values)
fs_tfidf_word = scipy.sparse.hstack((trainq1_trans,trainq2_trans))

tfidf_vect_ngram_chars = TfidfVectorizer(analyzer='char',min_df=3,
    token_pattern=r'\w{1,}',ngram_range=(1,2), max_features=5000)
tfidf_vect_ngram_chars.fit(pd.concat((df['question1'],df['question2'])).unique())
trainq1_trans = tfidf_vect_ngram_chars.transform(df['question1'].values)
trainq2_trans = tfidf_vect_ngram_chars.transform(df['question2'].values)
fs_tfidf_char = scipy.sparse.hstack((trainq1_trans,trainq2_trans))

svd_word = TruncatedSVD(n_components=180)
fs_svd_word = svd_word.fit_transform(fs_tfidf_word)

svd_char = TruncatedSVD(n_components=180)
fs_svd_char = svd_word.fit_transform(fs_tfidf_char)
fs_svd_char[0]

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import xgboost as xgb

from collections import Counter
df["question1"] = df["question1"].apply(lambda x: x.split())
df["question2"] = df["question2"].apply(lambda x: x.split())
c = Counter()
sent_data = df["question1"].values + df["question2"].values
```

```
for d in sent_data:
    c.update(d)
word_counts = sorted(dict(c).items(), key=lambda x: x[1], reverse=True)

print(word_counts[:10])

vocab_words = ["<PAD>", "<UNK>"]
for w, c in word_counts:
    vocab_words.append(w)

vocab2id = {w: i for i, w in enumerate(vocab_words)}
id2vocab = {i: w for i, w in enumerate(vocab_words)}

print("vocab size: ", len(vocab2id))
print(list(vocab2id.items())[:5])
print(list(id2vocab.items())[:5])

from tensorflow.keras.preprocessing.sequence import pad_sequences
def sent2index(vocab2id, words):
    return [vocab2id[w] if w in vocab2id else vocab2id["<UNK>"] for w in words]

df["question1"] = df["question1"].apply(lambda x: sent2index(vocab2id, x))
df["question2"] = df["question2"].apply(lambda x: sent2index(vocab2id, x))

df["question1"] = df["question1"].apply(lambda x: pad_sequences([x],maxlen=20, padding='post'))
df["question2"] = df["question2"].apply(lambda x: pad_sequences([x],maxlen=20, padding='post'))

question1_vectors=df.question1
question2_vectors=df.question2
df['cosine_distance'] = [cosine(x, y) for (x, y) in zip(np.nan_to_num(question1_vectors),
    np.nan_to_num(question2_vectors))]
df['cityblock_distance'] = [cityblock(x, y) for (x, y) in
    zip(np.nan_to_num(question1_vectors), np.nan_to_num(question2_vectors))]
df['jaccard_distance'] = [jaccard(x, y) for (x, y) in zip(np.nan_to_num(question1_vectors),
    np.nan_to_num(question2_vectors))]
df['canberra_distance'] = [canberra(x, y) for (x, y) in zip(np.nan_to_num(question1_vectors),
    np.nan_to_num(question2_vectors))]
df['euclidean_distance'] = [euclidean(x, y) for (x, y) in
    zip(np.nan_to_num(question1_vectors), np.nan_to_num(question2_vectors))]
df['minkowski_distance'] = [minkowski(x, y, 3) for (x, y) in
    zip(np.nan_to_num(question1_vectors), np.nan_to_num(question2_vectors))]
df['braycurtis_distance'] = [braycurtis(x, y) for (x, y) in
    zip(np.nan_to_num(question1_vectors), np.nan_to_num(question2_vectors))]

fs_distance = df[['cosine_distance','cityblock_distance','jaccard_distance',
             'canberra_distance','euclidean_distance','minkowski_distance','braycurtis_distance',]]
fs_distance=fs_distance.replace(np.nan,0)
```

```
X = np.hstack((fs_basic, fs_distance,fs_svd_char))
y = df.loc[:, df.columns == 'is_duplicate']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)

model = xgb.XGBClassifier(max_depth=50, n_estimators=100, learning_rate=0.1,
                        colsample_bytree=.7, gamma=0, reg_alpha=4, objective='binary:logistic',
                        eta=0.3, subsample=0.8).fit(X_train, y_train.values.ravel())
prediction = model.predict(X_test)
cm = confusion_matrix(y_test, prediction)
print(cm)
print('Accuracy', accuracy_score(y_test, prediction))
print(classification_report(y_test, prediction))
```

模型三

```
import pandas as pd

data_df = pd.read_csv("文本匹配数据集.csv")
data_df.head(10)

data_df["sent1"] = data_df["sent1"].apply(lambda x: x.split())
data_df["sent2"] = data_df["sent2"].apply(lambda x: x.split())
data_df.head(10)

from collections import Counter

c = Counter()
sent_data = data_df["sent1"].values + data_df["sent2"].values
for d in sent_data:
    c.update(d)
word_counts = sorted(dict(c).items(), key=lambda x: x[1], reverse=True)

print(word_counts[:10])

vocab_words = ["<PAD>", "<UNK>"]
for w, c in word_counts:
    vocab_words.append(w)

vocab2id = {w: i for i, w in enumerate(vocab_words)}
id2vocab = {i: w for i, w in enumerate(vocab_words)}

print("vocab size: ", len(vocab2id))
print(list(vocab2id.items())[:5])
print(list(id2vocab.items())[:5])

def sent2index(vocab2id, words):
```

```
    return [vocab2id[w] if w in vocab2id else vocab2id["<UNK>"] for w in words]

data_df["sent1"] = data_df["sent1"].apply(lambda x: sent2index(vocab2id, x))
data_df["sent2"] = data_df["sent2"].apply(lambda x: sent2index(vocab2id, x))

data_df.head(10)

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K

print(tf.__version__)

class BaseTextCNN(keras.Model):
    def __init__(self, filters, kernel_sizes, output_dim, name):
        super(BaseTextCNN, self).__init__(name=name)
        self.kernel_sizes = kernel_sizes
        self.conv_layers = []
        self.max_poolings = []
        for kernel_size in kernel_sizes:
            self.conv_layers.append(
                keras.layers.Conv1D(filters=filters, kernel_size=kernel_size,
                            activation='relu', padding="same")
            )
            self.max_poolings.append(keras.layers.GlobalMaxPool1D())
        self.concatenate = keras.layers.Concatenate()
        self.dense = keras.layers.Dense(output_dim, activation='tanh')

    def call(self, inputs):
        convs = []
        for i in range(len(self.kernel_sizes)):
            x = self.conv_layers[i](inputs)
            x = self.max_poolings[i](x)
            convs.append(x)
        x = self.concatenate(convs)
        output = self.dense(x)
        return output

max_len = 20
vocab_size = len(vocab2id)
embedding_size = 128
filters = 200
kernel_sizes = [3,4,5]
output_dim = 100

from tensorflow.keras.layers import Input,Embedding,Dot,Dense
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.layers import Dense,Conv1D,GlobalMaxPool1D,concatenate
from tensorflow.keras.layers import Convolution1D,Activation,MaxPool1D,GRU
from tensorflow.keras.layers import Bidirectional,LSTM,Dense
from tensorflow.keras.layers import Attention

# EMBEDDING_SIZE = 100
hidden_size= 64
attention_size = 50
# hidden_size=64
class_nums=2

input1=Input(name='sent1',shape=(max_len,))
input2=Input(name='sent2',shape=(max_len,))
embedding=Embedding(vocab_size,embedding_size)
sent1_embed=embedding(input1)
sent2_embed=embedding(input2)
x=Convolution1D(256, 3, padding='same', strides = 1)(sent1_embed)
x=Activation('relu')(x)
x=MaxPool1D(pool_size=2)(x)
x=GRU(256, dropout=0.2, recurrent_dropout=0.1, return_sequences = True)(x)
output_sent1=GRU(256, dropout=0.2, recurrent_dropout=0.1)(x)
x1=Convolution1D(256, 3, padding='same', strides = 1)(sent2_embed)
x1=Activation('relu')(x1)
x1=MaxPool1D(pool_size=2)(x1)
x1=GRU(256, dropout=0.2, recurrent_dropout=0.1, return_sequences = True)(x1)
output_sent2=GRU(256, dropout=0.2, recurrent_dropout=0.1)(x1)
cosine_output=Dot(axes=[1,1],normalize=True)([output_sent1,output_sent2])
outputs=Dense(1,activation='sigmoid',name="output")(cosine_output)
model=Model(inputs=[input1,input2],outputs=outputs)

import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

def batch_generator(all_data, batch_size, maxlen, shuffle=True):
    """
    :param all_data : all_data整个数据集,包含输入和输出标签
    :param batch_size: batch_size表示每个batch的大小
    :param shuffle: 是否打乱顺序
    :return:
    """
    # 输入all_datas的每一项必须是numpy数组,保证后面能按p所示取值
    all_data = [np.array(d) for d in all_data]
    # 获取样本大小
    data_size = all_data[0].shape[0]

    if shuffle:
        # 随机生成打乱的索引
```

```
        p = np.random.permutation(data_size)
        # 重新组织数据
        all_data = [d[p] for d in all_data]

    batch_count = 0
    while True:
        # 数据一轮循环(epoch)完成,打乱一次顺序
        if batch_count * batch_size + batch_size > data_size:
            batch_count = 0
            if shuffle:
                p = np.random.permutation(data_size)
                all_data = [d[p] for d in all_data]
        start = batch_count * batch_size
        end = start + batch_size
        batch_count += 1
        batch_data = [d[start: end] for d in all_data]
        batch_sent1, batch_sent2, batch_label = batch_data

        batch_sent1_pad = pad_sequences(batch_sent1, maxlen=max_len, padding='post')
        batch_sent2_pad = pad_sequences(batch_sent2, maxlen=max_len, padding='post')

        yield [batch_sent1_pad, batch_sent2_pad], batch_label

sent1_datas = data_df["sent1"].values.tolist()
sent2_datas = data_df["sent2"].values.tolist()
labels = data_df["label"].values.tolist()

# 划分训练 测试数据集
count = len(labels)
idx1, idx2 = int(count*0.8), int(count*0.9)
sent1_train, sent2_train = sent1_datas[:idx1], sent2_datas[:idx1]
sent1_val, sent2_val = sent1_datas[idx1:idx2], sent2_datas[idx1:idx2]
sent1_test, sent2_test = sent1_datas[idx2:], sent2_datas[idx2:]

train_labels, val_labels, test_labels = labels[:idx1], labels[idx1:idx2], labels[idx2:]

print("train data: ", len(sent1_train), len(sent2_train), len(train_labels))
print("val data: ", len(sent1_val), len(sent2_val), len(val_labels))
print("test data: ", len(sent1_test), len(sent2_test), len(test_labels))

# batch数据的生成器
batch_size = 64
maxlen = 15
batch_count = int(len(train_labels) / batch_size)
batch_gen_train = batch_generator([sent1_train, sent2_train, train_labels], batch_size,
    max_len)
batch_gen_val = batch_generator([sent1_val, sent2_val, val_labels], batch_size, max_len)
```

```
batch_gen_test = batch_generator([sent1_test, sent2_test, test_labels], batch_size, max_len)

epochs = 10

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 训练模型
history=model.fit(batch_gen_train,
                  verbose=1,
                  validation_data=batch_gen_val,
                  validation_steps=100,
                  steps_per_epoch=batch_count,
                  epochs=10)

# 预测模型
score, acc = model.evaluate(batch_gen_test, steps=50,
                                max_queue_size=10,
                                use_multiprocessing=False)
print('score:', score, 'accuracy:', acc)

# 保存训练好的模型
# model.save("output/cnndssm_semantic_match.h5")
# model.save_weights("output/match_model_weight.h5")

import matplotlib.pyplot as plt
def plot_history(history,title):
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('loss_value')
    plt.plot(hist['epoch'], hist['loss'],
          label='Train Loss')
    plt.plot(hist['epoch'], hist['val_loss'],
          label = 'Val Loss')
    plt.ylim([0,5])
    plt.legend()

    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('acc_value')
    plt.plot(hist['epoch'], hist['accuracy'],
          label='Train acc')
    plt.plot(hist['epoch'], hist['val_accuracy'],
          label = 'Val acc')
#   plt.ylim([0,20])
```

```
    plt.legend()
    plt.title(title)
    plt.show()

plot_history(history,'CNN+RNN')

def sent2index(vocab2id, words):
    return [vocab2id[w] if w in vocab2id else vocab2id["<UNK>"] for w in words]

sent1 = "I have a directory with about 10 polygon shapefiles."
sent2 = "I was wondering if anyone can help with batching a dissolve."

sent1_ids = sent2index(vocab2id, sent1.split())
sent2_ids = sent2index(vocab2id, sent2.split())

sent1_pad = pad_sequences([sent1_ids], maxlen=max_len, padding='post')
sent2_pad = pad_sequences([sent2_ids], maxlen=max_len, padding='post')

# model.load_weights("output/match_model_weight.h5")

preds = model.predict([sent1_pad, sent2_pad])

print("sent1: %s" % sent1)
print("sent2: %s" % sent2)
print("score: %s" % preds[0])
```