基于 CNN-DSSM 的问答社区文本匹配模型

【摘要】

近些年来越来越多的用户在社区问答平台中交流分享，提出自身的疑问、回答他人的问题，而随着平台新用户数量的不断增加，提出的问题也不断的增多。新用户提出的疑问在之前便很可能已经在平台上被提出并解答过，只是由于切入点不同或表述不同使至系统无法匹配到已经被解答的问题，这会增加平台的文本量和导致用户重复回答相同问题。为避免这种现象的发生，需要使用到 **NLP 的语义相似度分析**功能来对用户提出的问题和现有的问题进行识别。

本文针对技术类社区问答平台和附件所给问题进行分析，运用自然语言处理技术建立了 **DSSM 深度语义匹配模型**，利用**卷积神经网络** (CNN) 对其进行模型训练，在 DSSM 模型的基础之上，建立 **CNN-DSSM 模型**，用过增加 word-trigram 的处理方式其拥有了滑动窗口并可以提取在窗口中的上下文信息。之后再对 CNN-DSSM 模型进行训练，将训练数据传入神经网络，将 **tanh 函数**作为其输出之后计算得到得分。我们将得到的得分输入**最小化损失函数**中，得到误差，最后根据误差大小对识别程度的好坏进行判断。之后误差会在模型的表示层中反向传播，以反向求导的方式限制 tanh 函数以及最小化损失函数，减小误差并确定梯度向量，使得模型逐渐收敛，直到误差均值不再减小时模型最终收敛。

针对问题一，我们利用 **MPRC 人工智能经典数据集**对模型进行初步训练，再对附件所给数据进行整理，以确定样本级规模和文本长度，将其代入训练后的模型进行调试；又由于原始数据文本过长和数据量过大，故我们仅考虑问题的初始版本，利用**正则表达式**剔除不必要的信息，并限制读取问题文本的长度不超过 300。此时通过对比样本问题组发现模型的拟合效果并不理想，于是我们在 Google 数据集搜索引擎获取了四个相关性较强的数据集并以此重新对 CNN-DSSM 模型进行训练，并对再次模型进行检验。最终通过人工调整超参数确定最终模型。代入附件的样本问题后计算得到得到输出样本问题组为重复问题的概率为 **76.89 %** ，最后利用 **F1-score**公式进行评估, 得到 F_1为 **0.7796511** 。

针对问题二，我们利用在问题一中确立的模型的基础上，以 **Cosine 距离**作为文本匹配的相似度数值，对目标问题进行排序，得到与目标问题相似程度数值最大的十个问题编号，并对每个问题的预测结果利用 **Top K**列表进行评价，最终得到与目标问题重复概率最大的 10 个问题编号。

关键词：DSSM 深度语义匹配模型、卷积神经网络、F1-score

目录

1 问题的背景和重述

1.1 问题背景

一些技术问答平台时常被作为用户相互分享交流的社区平台，在这些年来也逐步变为用户寻找技术类疑难解答的首要渠道。这种情况也导致了各个分类技术性问题的文本数据量不断攀升，给问答平台的日常运营维护带来了挑战。随着用户数量的不断增加，一些用户提出的疑问可能已经在平台上被其他用户提出并解答过，但由于技术性问题的复杂性，各个用户提问的切入角度不同，用问题标题关键词匹配的搜索系统无法指引新用户至现有的问题解答方案。于是，新用户会提出重复的问题，而这些问题会进一步增加平台上的文本量，导致用户重复回答相同的问题。对于这种现象，通常的做法是及时找到新增的重复问题并打上标签，然后在搜索结果中隐藏该类重复问题，保证对应已解决问题出现的优先度。所以，建立一个基于自然语言处理技术的自动标重系统会对问答平台的日常维护起到极大帮助。

目前，问答平台上对问题重复的判断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，如果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 待解决的问题

附件 1 中给出了相关技术问答平台上问题的文本内容记录，附件 2 给出了比较两个问题之间是否重复的数据集。要求我们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，并解决以下两个问题：

(1) 使用 F1-score 对分类模型进行评价, 得到输出样本问题组为重复问题的概率；

(2) 对于每个问题的预测结果采用 Top K 列表对其进行评估，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；

2 问题分析

2.1 问题一的分析

问题一属于评价类问题，对此我们需要基于 NLP 对每一个样本问题与其它问题的语义进行比较。我们通过建立 CNN-DSSM 模型，利用 MRPC 经典人工智能训练数据集对该模型进行训练。在数据集通过神经网络对模型进行训练后我们便可以得到训练后的 CNN 模型。将附件 1、2 中的数据进行格式转化后代入训练后的 CNN 模型进行测试，并利用 F1-socre 作为文本匹配程度的衡量指标，实现对附件中样本问题组是否为重复问题概率的评价。

2.2 问题二的分析

问题二与问题一的模型类别相同，我们利用在问题一中已经训练过的模型根据 CNN-DSSM 模型的评估函数得到目标问题与其它所有问题文本之间的匹配程度数值，并按照该数值对其进行排序，利用 Top K 列表对预测结果进行评价，得到与其重复概率最大的前 10 个问题编号。

3 模型的假设

(1) 假设单词前后缀代表的语义具有通用特性

(2) 假设所比较问题文本上下文距离间隔不大

(3) 假设附件给出的正负样本准确性较好

(4) 假设将不同单词和句子映射到同一语义空间时冲突较少

4 符号说明

符号	含义
x	输入的 term 向量
y	输出向量
l_i	隐藏层
W_i	第 i 层的参数矩阵
b_i	第 i 个偏置项
Q	query1
D	query2
R	相关性分数
γ	超参数
D^+	文档列表中的正参数
Λ	模型全部训练参数
W	卷积核
W_{output}	输入层维度
X	输入矩阵
pad	填充大小
$kernel\_size$	卷积核大小
$stride$	步长
P_i	分类器判定正例中的正样本的比重
R_i	被预测为正例的占总的正例的比重
TP_i	第 i 个样本预测答案正确
TN_i	第 i 个样本预测正确答案为错误
FP_i	第 i 个样本错将其他类预测为本类
FN_i	将第 i 个样本标签预测为其他类标签
R_{Top}	Top K 评估值

5 模型的建立与训练

5.1 DSSM 模型的建立

5.1.1 模型的原理

DSSM(Deep Structured Semantic Models) 的核心思想是将不同的对象映射到统一的语义空间中，并在该空间中计算对象之间的相似度。首先用神经网络模型把 query1 和 query2 映射到同一低维语义向量空间，并且通过 Cosine 距离来计算两个语义向量的距离，最终训练出语义相似度模型。该模型既可以用来预测两个句子的语义相似度，又可以获得某句子的低纬语义向量表达。DSSM 从下往上可以分为三层结构：输入层、表示层、匹配层。

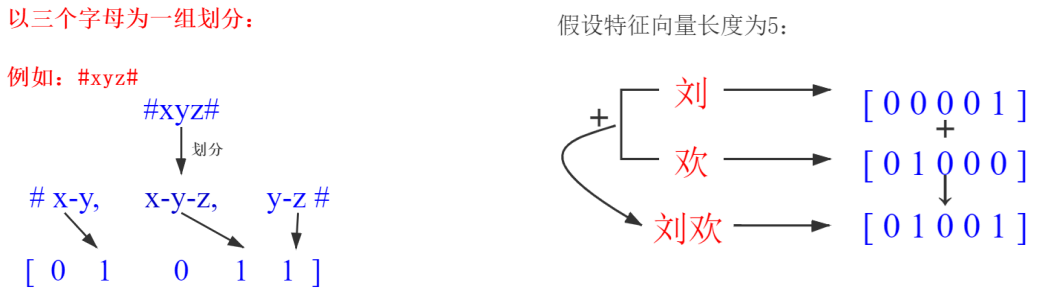
5.1.2 模型的建立

5.1.2.1 输入层

输入层的功能是将输入语句转化为特征向量并输入到神经网络中，而对于不同语言形式的输入的语句，其处理方式也大有不同。

对于英文来讲，我们的处理方式是通过 Word hashing 的处理方式，即将通过多个字母分为一组再进行映射，映射为一个特征向量，考虑英文单词和字母的特性，选取每三个字母作为一组，通过编码压缩成特征向量。(示例如图 1(a)) 这样可以压缩空间，将 50 万个词的向量空间可以通过压缩为一个 3 万维的向量空间。也可以增强范化能力，因为三个字母的表达往往能代表英文字符中的前后缀，而在英文的构词法中单词的前后缀在实际中的语义具备通用的特性。

对于中文来说，最复杂的部分是对中文语句的分词处理，本文中使用到 one-hot 文本特征提取的方法对其进行处理，one-hot 的处理方式是指将中文的每个字符进行编码，映射为一个特征向量，这个向量只有一列不为 0，是一个稀疏向量，对于多种字符组成的句子而言，one-hot 编码的结果为其特征向量的和 (示例如图 1(b))：



(a) Word hashing 举例　　(b) One Hot 举例

图 1: 输入层处理方式的例子

5.1.2.2 表示层

表示层的功能是将变长的句子转换为定长，并基于之前对字符的定义构造词典。由于 DSSM 模型不考虑单词或字的顺序，因此我们选择采用 Bag of Words 的处理方式, 其处理思想与在 one-hot 中表述句子的思想相同，不考虑单词词序，直接将每个字的特征向量相加。此时特征向量中每个元素的值代表词典中相关元素出现的次数，原句中单词也丧失了原本的次序特征。定长的向量可以表达少量的句子信息，采用深度神经网络学习训练其句意表达，最终表示成实现定长向量表示句子的语义。

对于神经网络结构，其将初始的文本特征映射为其在语义空间上表示的特征，在文本匹配中主要作用如下:

1. 将 query 中 term 的高维向量映射为低维语义向量

2. 根据语义向计算 query1 与 querry2 之间的相关性分数

计算公式如下：

$$
\begin{aligned}
l_1 &= W_1 x \\
l_i &= f\left(W_i l_{i-1} + b_i\right), i = 2, \ldots, N-1 \\
y &= f\left(W_N l_{N-1} + b_N\right)
\end{aligned}
$$

其中 x 用来表示输入的 term 向量，y 表示输出向量，l_i i= 1.,...,N- 1 表示隐藏层,W_i 示第 i 层的参数矩阵，b_i 表示第 i 个偏置项。

5.1.2.3 匹配层

在计算两个句子的语义匹配程度时，我们用之前表示层求得的语义向量通过计算余弦距离来表示其相似程度，余弦距离越接近 1 则表明两个句子的匹配程度越好。

在文本匹配中，我们使用 Q 来表示一个 query，D 来表示另一个 querry, 那么它们的相关性分数可以拥下面的公式衡量:

$$
R(Q, D) = \operatorname{cosine}\left(y_Q, y_D\right) = \frac{y_Q^T y_D}{\|y_Q\| \|y_D\|} \tag{1}
$$

5.1.2.4 优化函数

在匹配层中我们通过余弦距离获得了相关性分数，利用 softmax 函数将两个句子的相关性分数归一化：

$$
P\left(D_i \mid Q\right) = \frac{\exp\left(\gamma R\left(Q, D_i\right)\right)}{\sum_{k=1}^{n} \exp\left(\gamma R\left(Q, D_k\right)\right)} \tag{2}
$$

其中 γ 为超参数，即机器学习前设置的值

在模型的训练阶段，通过极大似然估计法来获得了最小化损失函数：

$$
L(\Lambda) = -\sum_{D^+} \log P\left(D^+ \mid Q\right) \tag{3}
$$

其中 D^+ 为文档列表中的正参数，Λ 为模型全部训练参数

5.2 CN-DSSM 模型的建立

为了解决 DSSM 中 Bag of Words 模型中上下文信息会丢失的问题，我们通过进一步完善输入层和表示层的模型来优化模型

5.2.1 模型的建立

5.2.1.1 输入层

对英文，增加 word-trigram(示例如图 2)：

word-trigram 是包含每个单词上下文信息的滑动窗口，对每个单词按照 DSSM 中的 letter-trigram 进行 Word hashing 后映射到一个更大的特征向量。

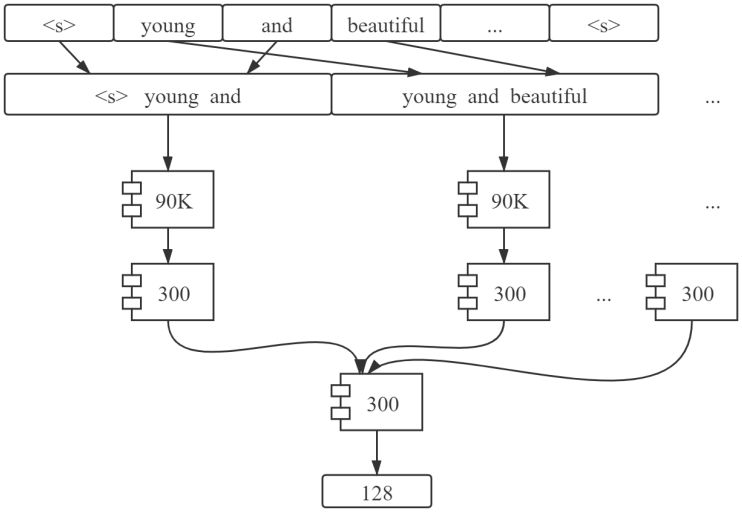


图 2: Word trigram

对于中文如果用 word-trigram 则向量空间会过大，因此我们仍然采用字向量的方式对中文进行处理。

5.2.1.2 表示层

CNN-DSSM 的表示层由一个卷积神经网络组成，总共分为三层: 卷积层、池化层、全连接层。

(1) 卷积层 - - Convolutional layer

在 CNN 模型中使用的卷积公式和严格数学意义中的定义略有不同，对于本文中对两个问题语义相似度进行比较的二维模型，采用以下计算方式：

$$s(i,j) = (X * W)(i,j) = \sum_m \sum_n x(i+m, j+n) w(m,n)$$

其中，W 为卷积核，X 为输入。当 X 为二维输入的矩阵时，W 也为二维的矩阵，X 为多维张量时，W 也是多维张量。

卷积层用于提取滑动窗口下的上下文特征。假设输入是一个 $m \times n$ 的二维矩阵，代表 m 个字量，query1 的和 querry2 的长度小于 300 时用 0 字符补齐，大于 300 时舍去多余部分。每个字向量的维数为九万维。而卷积核为权值矩阵，并以步长为 1 的长度向下移动，得到的 feature map 是个 300 ×1 的矩阵，feature map 的计算公式为[1]：

$$H_{output} = \frac{H_{input} + 2 \times pad - kernel\_size}{\text{stride}} + 1$$

$$W_{output} = \frac{W_{input} + 2 \times pad - kernel\_size}{\text{stride}} + 1$$

其中输入层维数为 W_{output}，填充大小为 pad(Padding)，卷积核大小为 $kernel\_size$, 步长大小为 stride. 而这样的卷积核有 300 个故形成了 300 个 300×stride 的 feature map 矩阵。

(2) 池化层 - - Max pooling layer

池化就是对输入张量的各个子矩阵进行压缩。假如是 2×2 的池化，那么就将子矩阵的每 2×2 个元素变成一个元素，以减小输入矩阵的维度。若要想将输入子矩阵的每 n×n 个元素变成一个元素，则需要一个池化标准。常见的池化标准有 2 个，MAX 或者是 Average。即取对应区域的最大值或者平均值作为池化后的元素值。

池化层的作用则是为句子找到全局的上下文特征，本文我们采用 Max-over-time pooling 的操作，使得每个 feature map 都取到最大值，得到一个 m 维的向量。Max-over-pooling 可以解决可变长度的句子输入问题 (不论 Feature Map 中有多少个值, 仅需要提取其中的最大值)。在上一步中我们已经做过句子的定长处理 (固定句子长度为 m)，故无可变长度句子的问题。最终池化层的输出为各个 feature map 的最大值，即一个 300×stride 的向量。这里多提一句, 之所以 Max pooling 层要保持固定的输出维度，是因为下一层全链接层要求有固定的输入层数, 才能进行训练。

(3) 全连接层 - - Semantic layer

最后通过全连接层把一个 300 维的向量转化为一个 128 维的低维语义向量。全

[1]注：当 stride 为 1 时, 若 $pad = \frac{kernel\_size-1}{2}$, 那么经过计算后的 feature map 大小不变

连接层采用 tanh 函数作为激活函数:

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{4}$$

5.2.1.3 匹配层

计算方式与 DSSM 中一致，但对于优化后的模型来说，CNN-DSSM 可以在卷积层获取了滑动窗口中的上下文信息，在池化层获取了全局的上下文信息，克服了 DSSM 中无法获得字序等上下文信息的缺点。

5.3 CNN-DSSM 模型的训练

模型的训练是通过数据寻找特定的模型参数值，即带参函数的参数值。

5.3.1 深度模型训练的基本要素

模型训练的基本影响要素通常包含模型的复杂性，模型结构，对数据的处理和最终目标函数这四大要素，各个基本要素又通过不同的要素因子来对训练的结果造成影响 (详情请见图 3)：

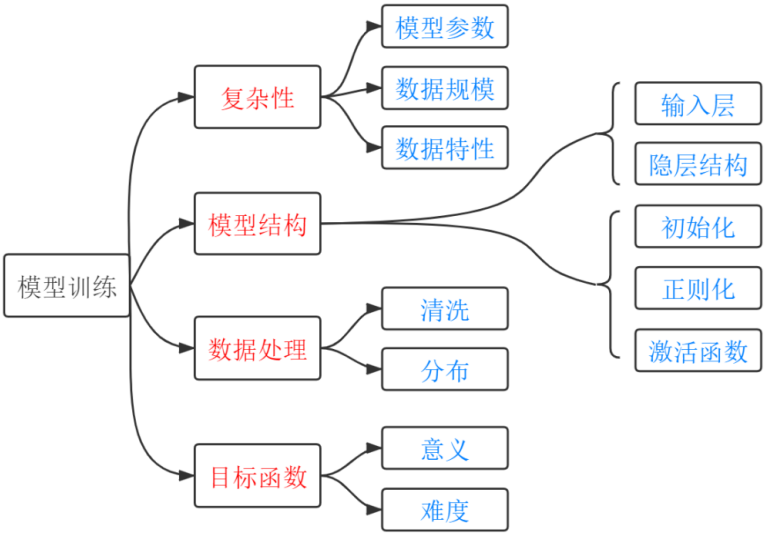


图 3: 深度模型训练的基本要素

本文对 CNN-DSSM 模型的训练便依据上述四个方面进行考虑。

5.3.2 CNN-DSSM 模型的训练过程

首先我们将训练数据传入到神经网络模型中，将每个神经元输入值加权累加后，将 tanh 函数 (公式 4) 作为其输出，正向传播之后计算得到得分。

计算得到得分后，我们将得分输入到误差函数即最小化损失函数中 (公式 3) 防止其过度拟合并同期待值进行比较，得到误差，根据误差的大小对识别程度的好坏进行判断。

上一步中我们得到的误差会在表示层中反向传播，通过反向求导来限制 tanh 函数、最小化损失函数，减小误差并确定梯度向量，使模型逐渐收敛。

重复上述过程，直到误差均值不再减小时，模型最终收敛。

6 问题一的求解

对于问题一，我们的求解步骤如下:

Step1:我们利用 MRPC 人工智能经典数据集对 CNN-DSSM 模型进行初步的训练。再将附件 1、2 汇总的数据整理成规范的格式 (是否匹配，query1，query2), 在确定了样本级的规模和文本的长度的参数之后，带入已经训练后的 CNN-CSSM 模型中进行测试。

训练结果如下：

测试集：MRPC

测试集准确率：0.6649

测试集序列	loss
1	0.6978
2	0.6043
3	0.6183
4	0.6471
5	0.6790
6	0.5969
7	0.6623
8	0.6150
…	…
793	0.6248
794	0.7180
795	0.6956
796	0.5650
797	0.6139
798	0.6458
799	0.6253
800	0.7280

表 1

Step2:测试时，我们发现数据文本过长和数据量过大导致处理时间太长难以得到结果，因此对原始的数据中的问题文本进行处理，处理的方法如下：

1：只考虑问题的英文版本，因为英文词语在模型中压缩映射后的语义空间较小。同时 word hashing 的冲突率较低，因此可以显著改善测试的效率。

2：限制问题文本的长度，附件中给出的部分样本的问题长度过长，导致求解的效率显著下降，由于中短长度的文本基本可以包含该问题的大部分信息，因此限制问题样本的文本长度不超过 300，又考虑到问题的前半部分通常包括了问题的主要内容，故我们采取删除过长问题文本尾部的方式对其进行处理。

3：样本问题中包含了相当数量的特殊符号和一些不容易进行匹配的文本，比如一些代码中的关键词等，因此在测试数据前对样本内容进行预处理。具体处理方法为利用正则表达式去除特殊符号和难以匹配的问题文本，通过这样的处理可以较好的压缩文本的内容，提高文本处理识别效率。

Step3:经过以上的处理之后，我们利用训练后的 CNN-DSSM 模型计算问题样本之间的匹配程度，通过计算部分问题的匹配程度之后，通过比较发现其匹配程度较差，在对比样本问题组之后发现模型拟合程度并不好，同时计算得到的 F1-score 值较小，因此考虑对模型进一步优化处理，在对神经网络训练部分的模型进行调配后，发现拟合效果并没有得到改善。可以判断训练效果不好的因素除了模型本身的局限性之外，主要是训练集过少且与附件中的样本问题及相关性较低。

测试结果如表 2：

Query1	Query2	样本	预测
23129	10788	1	1
87000	86444	1	1
97776	8113	1	0
28260	28086	1	1
82598	21633	1	0
94706	33041	1	0
79328	28333	1	1
77184	664	1	1
72242	72051	1	1
112933	9380	1	0
54266	7199	1	1
92772	80270	1	1
112654	86960	1	0
50508	42776	1	0
81631	79730	1	0
59513	57394	1	1
…	…	…	…
71571	791	0	0
31769	15135	0	1
103393	82314	0	0
88298	77934	0	0
6252	839	0	0
45340	9542	0	1
5246	1444	0	0
17966	191	0	0
110492	91722	0	1
32931	4045	0	0
15811	1680	0	0
103017	56975	0	1
103016	71269	0	1

表 2: Step3 测试结果

数据集	GIS-1	GIS-2	ArcGIS-1	ArcGIS-2
准确率	0.7689			
Query1	Query2	样本	预测	
59740	18220	1	1	
20521	19872	1	1	
82415	33125	1	1	
10756	7839	1	0	
13107	12242	1	1	
97814	91537	1	1	
82931	33685	1	0	
59442	8178	1	1	
49854	1699	1	0	
44116	11667	1	1	
53785	15775	1	1	
60712	16479	1	0	
69361	20572	1	0	
17620	8794	1	1	
61909	61768	1	1	
82618	82081	1	1	
…	…	…	…	
86141	54777	0	0	
82498	7993	0	0	
100056	66892	0	0	
68438	2335	0	1	
44475	31171	0	0	
111879	54077	0	0	
62329	7038	0	1	
59042	30700	0	0	
74611	64150	0	0	
60977	58155	0	1	
99951	27733	0	0	
21908	3485	0	0	
102914	22918	0	0	
79622	4097	0	1	
28749	4469	0	0	

表 3：Step4 测试结果

Step4:为了进一步优化模型，提升其拟合效果，我们通过 Google 数据集搜索引擎以 GIS 和 ARCGIS 为关键词获取了四个相关性较强的数据集，同时选取了抽取了附件中 500 条样本作为验证集，再对利用新的数据集重新对 CNN-DSSM 模型进项训练，将验证集中的数据带入验证后的模型，通过人工调整超参数确定了最终的模型，代入附件中的样本问题及进行测试，利用 F1-score 公式对其进行评价，其计算公式如下：

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

$$P_i = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i + FP_i}$$

$$R_i = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i + FN_i}$$

其中，P_i 指被分类器判定正例中的正样本的比重，R_i 指被预测为正例的占总的正例的比重，TP_i 表示第 i 个样本预测答案正确，FP_i 第 i 个样本错将其他类预测为本类，FN_i 表示将第 i 个样本标签预测为其他类标签。

经过模型对问题一样本的测试后我们得到最终的测试结果如表 3 所示，其中准确率即为输出样本问题组为重复问题的概率，概率为：**76.89 %** 。接下来通过 F1-score 对分类模型进行评价，由于测试之后任意两个问题之间均有一组记录，因此总记录数为:

$$\frac{7295 \times 7294}{2} = 26604865\text{ 条}$$

得到 TP_i=12033211 ，TN_i=8423269 ，FP_i=2780520 ，FN_i=3367865 将这些参数代入公式计算得到结果如下：

$$P_i = 0.812301$$

$$R_i = 0.781323$$

$$F_1 = 0.796511$$

7 问题二的求解

问题二需要我们在问题一最终确立的模型基础上，以 Cosine 距离作为文本匹配相似程度数值，针对附件中的每一个目标问题通过排序得到与该问题相似程度数值最大的十个问题编号，并对每一个的问题的预测结果利用 Top K 列表进行评价，

最终得到与目标问题重复概率最大的问题编号。

Top K 计算公式如下：

$$R_{Top} = \frac{N_{\text{detected}}}{N_{\text{total}}}$$

下面进行求解：

Step1:根据现有的模型和 Cosine 距离公式 (公式 1) 计算出文本匹配相似程度数值，部分结果如下表所示：

Query1	Query2	余弦距离
35624	14323	0.64
38855	21761	0.64
44502	22303	0.6
74453	27457	0.68
72097	48364	0.63
74692	37405	0.6
31147	16691	0.64
77184	664	0.64
25727	14323	0.62
49001	1553	0.61
59869	9896	0.64
84063	36724	0.68
60848	50423	0.65
48899	40660	0.66
31900	5999	0.65
90859	33629	0.61
67887	44526	0.59
78378	31261	0.64
37474	37453	0.67
75176	4906	0.61
94874	86802	0.66
88827	5821	0.62
74284	74071	0.53
73032	71841	0.55
3301	1987	0.58
107251	57561	0.52
70622	61499	0.5
73399	50415	0.22
89370	61411	0.26
11543	10444	0.24
89375	75316	0.23
89377	83316	0.21
89379	63683	0.25
5986	4927	0.24
…	…	…

表 4：文本匹配相似程度部分数值

由于计算得出的全部数据共有两千余万条，故我们将其余未展示的数据结果附在支撑材料中进行展示。

Step2:根据 Cosine 距离公式得到的结果，我们对每一个问题的与其它所有问题的文本匹配程度数值按降序进行排序。

Step3:根据 Top K 列表进行评估，K 取 10，又根据在问题一模型中求得的每一个问题与其它问题语义匹配的结果得到 $N_{detected}$，根据 Top K 评估公式得到部分与目标问题重复概率最大的前 10 个问题的编号与 Top K 列表的评估值如下所示：

列名编号	与之重复概率最大前十个问题编号	top K评估值
12904	110237	0.4
	55542	
	36812	
	29281	
	22446	
	87749	
	81819	
	79981	
	75930	
	64055	

列名编号	与之重复概率最大前十个问题编号	top K评估值
15607	90290	0.8
	81495	
	49080	
	54588	
	23763	
	94847	
	88399	
	61084	
	58028	
	37248	

列名编号	与之重复概率最大前十个问题编号	top K评估值
14919	36891	0.8
	72906	
	39349	
	82444	
	50682	
	92174	
	17916	
	108299	
	102336	
	87429	

表 5：结果示例

由于全部数据较多，在论文中不便展示，我们仍将其放置于支撑材料中进行展示。

8 模型的优缺点和推广

8.1 模型的优点

1.DSSM 用字向量作为输入，减少了对切词的依赖，并提高模型的范化能力，因为每个汉字所能表达的语义是可以复用的。

2. 传统的输入层是用 Embedding 模型直接做词的映射，再将各词的向量累加，由于 Word2Vec 和 LDA 都是无监督的训练，这样会给整个模型引入误差。本文的 DSSM 采用统一的有监督训练，不需要在中间过程做无监督模型的映射，因此精准度较高。

3.CNN-DSSM 通过卷积层提取了滑动窗口下的上下文信息，又通过池化层提取了全局的上下文信息，上下文信息得到较为有效的保留。

4. 通过池化层提取了全文的上下文信息。

5. 共享卷积核，对高维数据处理无压力。

6. 特征工程无需人工操作。

8.2 模型的缺点

1. 由于 CNN-DSSM 滑动窗口大小的限制，对于文本中间隔较远的上下文信息，难以有效保留。

2.DSSM 是端到端的模型，虽然省去了人工特征转化、特征工程和特征组合，但端到端的模型有个问题就是效果不可控。

3. Word hashing 的处理方式可能造成冲突。

9 参考文献

[1]Po-Sen Huang,Xiaodong He,Jianfeng Gao,Li Deng,Alex Acero,Larry Heck. Learning deep structured semantic models for web search using clickthrough data[P]. Information & Knowledge Management,2013.

[2]Ali Mamdouh Elkahky,Yang Song,Xiaodong He. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems[P]. World Wide Web,2015.

[3]Yelong Shen,Xiaodong He,Jianfeng Gao,Li Deng,Grégoire Mesnil. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval[P]. Conference on Information and Knowledge Management,2014.

[4]H. Palangi1, L. Deng2, Y. Shen2, J. Gao2, X. He2, J. Chen2, X. Song2, R. Ward1.SEMANTIC MODELLING WITH LONG-SHORT-TERM MEMORY FOR INFORMATION RETRIEVAL[P].arXiv preprint arXiv:1412.6629 (2014).

[5]CSDN. 深度学习解决 NLP 问题：语义相似度计算——DSSM[EB/OL]. http://blog.csdn.net/u013074302/article/details/76422551.2021.05.02

附录

transform.py

将附件数据转化为标准格式

```
import csv
import re

map = []
dict = {}
judge = False
ccc = 3
with open('附件1.csv', 'r', encoding='utf-8') as fr:
reader = csv.reader(fr)
for i in reader:
if not judge:
judge = True
continue
i[1] = i[1].replace('\r', '').replace('\n', '').replace('\t'
    , '').replace(',', ' ').lstrip()
i[1] = re.sub("\[", "", i[1])
i[1] = re.sub("]", "", i[1])
i[1] = re.sub("<", "", i[1])
i[1] = re.sub(">", "", i[1])
i[1] = ' '.join(i[1].split())
i[1] = i[1][0:300]

map.append({'id': i[0], 'context': i[1]})
dict.update({i[0]: i[1]})

judge = False
map2 = []

with open('附件2.csv', 'r', encoding='utf-8') as fr:
```

```
reader = csv.reader(fr)
for i in reader:
if not judge:
judge = True
continue
left = i[0]
right = i[1]
flag = i[2]
if i[2] == '1':
right = re.sub("\'", "", right)
right = re.sub("\[", "", right)
right = re.sub("]", "", right)
map2.append({'left': left, 'right': right, 'flag': flag})
else:
for j in map:
if int(j['id']) < int(left):
map2.append({'left': left, 'right': j['id'], 'flag': flag})

f = "train_data.csv"
with open(f, "w", encoding='utf-8')as file:
for i in map2:
file.write(
i['flag'] + ',' + dict[i['left']] + ',' + dict[i['right']] +
    ',' + i['left'] + ',' + i['right'] + '\n'
)
```

问题一的求解

args.py

配置模型参数

```
TRAIN_DATA = './MRPC/train_data.csv'
TEST_DATA = './MRPC/test_data.csv'

```

```
VOCAB_FILE = './vocab_mrpc.txt'
MODEL_FILE = './dssm_pr.pkl'

N = 3
SENTENCE_MAXLEN = 800
CHAR_SIZE = 10041
EMBEDDING_SIZE = 300

EPOCH = 100
BATCH_SIZE = 50
LR = 0.0005
```

data_load.py

加载数据

```
import numpy as np
import pandas as pd
from make_vocab import lst_gram
import args


def load_vocab():
vocab = open(args.VOCAB_FILE, encoding='utf-8').readlines()
slice2idx = {}
idx2slice = {}
cnt = 0
for char in vocab:
char = char.strip('\n')
slice2idx[char] = cnt
idx2slice[cnt] = char
cnt += 1
return slice2idx, idx2slice


```

```
20 def padding(text, maxlen=args.SENTENCE_MAXLEN):
21 pad_text = []
22 for sentence in text:
23 pad_sentence = np.zeros(maxlen).astype('int64')
24 cnt = 0
25 for index in sentence:
26 pad_sentence[cnt] = index
27 cnt += 1
28 if cnt == maxlen:
29 break
30 pad_text.append(pad_sentence.tolist())
31 return pad_text
32
33
34 def char_index(text_a, text_b):
35 slice2idx, idx2slice = load_vocab()
36 a_list, b_list = [], []
37
38 for a_sentence, b_sentence in zip(text_a, text_b):
39 a, b = [], []
40
41 for slice in lst_gram(a_sentence):
42
43 if slice in slice2idx.keys():
44 a.append(slice2idx[slice])
45 else:
46 a.append(1)
47
48 for slice in lst_gram(b_sentence):
49 if slice in slice2idx.keys():
50 b.append(slice2idx[slice])
51 else:
52 b.append(1)
53
```

```
54  a_list.append(a)
55  b_list.append(b)
56
57  a_list = padding(a_list)
58  b_list = padding(b_list)
59
60  return a_list, b_list
61
62
63  def load_char_data(filename):
64  df = pd.read_csv(filename, sep='\t')
65  text_a = df['#1 string'].values
66  text_b = df['#2 string'].values
67  label = df['quality'].values
68  a_index, b_index = char_index(text_a, text_b)
69  return np.array(a_index), np.array(b_index), np.array(label)
```

make_vocab.py

word hashing 处理

```
1  import args
2
3  def n_gram(word, n=args.N):
4  s = []
5  word = '#' + word + '#'
6  for i in range(len(word) - 2):
7  s.append(word[i:i + 3])
8  return s
9
10
11 def lst_gram(lst, n=args.N):
12 s = []
13 for word in str(lst).lower().split():
14 s.extend(n_gram(word))
```

```
return s


vocab = []
file_path = './MRPC/'
files = ['train_data.csv', 'test_data.csv']

for file in files:
f = open(file_path + file, encoding='utf-8').readlines()
for i in range(1, len(f)):  # 从遍历因为下标为表头
   （10gold_label sentence1 ）sentence2
s1, s2 = f[i][2:].strip('\n').split('\t')
vocab.extend(lst_gram(s1))
vocab.extend(lst_gram(s2))

vocab = set(vocab)
vocab_list = ['[PAD]', '[UNK]']
vocab_list.extend(list(vocab))

vocab_file = args.VOCAB_FILE
with open(vocab_file, 'w', encoding='utf-8') as f:
for slice in vocab_list:
f.write(slice)
f.write('\n')
```

model.py

模型类和函数

```
import torch
import torch.nn as nn
import args

class DSSM(nn.Module):
def __init__(self):
super(DSSM, self).__init__()
```

```
8  self.embedding = nn.Embedding(args.CHAR_SIZE, args.
       EMBEDDING_SIZE)
9  self.linear1 = nn.Linear(args.EMBEDDING_SIZE, 256)
10 self.linear2 = nn.Linear(256, 128)
11 self.linear3 = nn.Linear(128, 64)
12 self.dropout = nn.Dropout(p=0.2)
13
14 def forward(self, a, b):
15 a = self.embedding(a).sum(1)
16 b = self.embedding(b).sum(1)
17
18 a = torch.tanh(self.linear1(a))
19 a = self.dropout(a)
20 a = torch.tanh(self.linear2(a))
21 a = self.dropout(a)
22 a = torch.tanh(self.linear3(a))
23 a = self.dropout(a)
24
25 b = torch.tanh(self.linear1(b))
26 b = self.dropout(b)
27 b = torch.tanh(self.linear2(b))
28 b = self.dropout(b)
29 b = torch.tanh(self.linear3(b))
30 b = self.dropout(b)
31
32 cosine = torch.cosine_similarity(a, b, dim=1, eps=1e-8)
33 return cosine
34
35 def _initialize_weights(self):
36 for m in self.modules():
37 if isinstance(m, nn.Linear):
38 torch.nn.init.xavier_uniform_(m.weight, gain=1)
```

train.py

训练模型

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from torch.autograd import Variable
from data_load import load_char_data
from model import DSSM
import args


class MRPCDataset(Dataset):
def __init__(self, filepath):
self.path = filepath
self.a_index, self.b_index, self.label = load_char_data(
    filepath)

def __len__(self):
return len(self.a_index)

def __getitem__(self, idx):
return self.a_index[idx], self.b_index[idx], self.label[idx]


if __name__ == '__main__':
train_data = MRPCDataset(args.TRAIN_DATA)
test_data = MRPCDataset(args.TEST_DATA)
train_loader = DataLoader(dataset=train_data, batch_size=
    args.BATCH_SIZE, shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=args.
    BATCH_SIZE, shuffle=True)

device = torch.device("cuda:0" if torch.cuda.is_available()
    else "cpu")
```

```
29
30 dssm = DSSM().to(device)
31 dssm._initialize_weights()
32 optimizer = torch.optim.Adam(dssm.parameters(), lr=args.LR)
33 loss_func = nn.CrossEntropyLoss()
34
35 for epoch in range(args.EPOCH):
36 for step, (text_a, text_b, label) in enumerate(train_loader)
       :
37 a = Variable(text_a.to(device).long())
38 b = Variable(text_b.to(device).long())
39 l = Variable(torch.LongTensor(label).to(device))
40 pos_res = dssm(a, b)
41 neg_res = 1 - pos_res
42 out = torch.stack([neg_res, pos_res], 1).to(device)
43
44 loss = loss_func(out, l)
45 optimizer.zero_grad()
46 loss.backward()
47 optimizer.step()
48
49 if (step + 1) % 20 == 0:
50 total = 0
51 correct = 0
52 for (test_a, test_b, test_l) in test_loader:
53 tst_a = Variable(test_a.to(device).long())
54 tst_b = Variable(test_b.to(device).long())
55 tst_l = Variable(torch.LongTensor(test_l).to(device))
56 pos_res = dssm(tst_a, tst_b)
57 neg_res = 1 - pos_res
58 out = torch.max(torch.stack([neg_res, pos_res], 1).to(device
       ), 1)[1]
59 if out.size() == tst_l.size():
60 total += tst_l.size(0)
```

```
61 correct += (out == tst_l).sum().item()
62 print('[Epoch]:', epoch + 1, '训练loss:', loss.item())
63 print('[Epoch]:', epoch + 1, '测试集准确
      率: ', (correct * 1.0 / total))
64
65 torch.save(dssm, args.MODEL_FILE)
```

test.py

测试模型

```
1  import torch
2  from torch.utils.data import DataLoader, Dataset
3  from torch.autograd import Variable
4  from train import MRPCDataset
5  import args
6
7  if __name__ == '__main__':
8  test_data = MRPCDataset(args.TEST_DATA)
9  test_loader = DataLoader(dataset=test_data, batch_size=args.
      BATCH_SIZE, shuffle=False)
10
11 device = torch.device("cuda:0" if torch.cuda.is_available()
      else "cpu")
12 dssm = torch.load(args.MODEL_FILE).to(device)
13
14 total = 0
15 correct = 0
16 TP, TN, FP, FN = 0, 0, 0, 0
17 FLAG = True
18 for (test_a, test_b, test_l) in test_loader:
19 tst_a = Variable(test_a.to(device).long())
20 tst_b = Variable(test_b.to(device).long())
21 tst_l = Variable(torch.LongTensor(test_l).to(device))
22
23 pos_res = dssm(tst_a, tst_b)
```

```
neg_res = 1 - pos_res
out = torch.max(torch.stack([neg_res, pos_res], 1).to(device
    ), dim=1)[1]

total += tst_l.size(0)
correct += (out == tst_l).sum().item()

TP += ((out == 1) & (tst_l == 1)).sum().item()
TN += ((out == 0) & (tst_l == 0)).sum().item()
FN += ((out == 0) & (tst_l == 1)).sum().item()
FP += ((out == 1) & (tst_l == 0)).sum().item()

if FLAG == True:
for i in range(10, 20):
a, b, l = test_data[i][0], test_data[i][1], test_data[i][2]
print('标签：', l, '预测：', out[i].item())
FLAG = False

p = TP / (TP + FP)
r = TP / (TP + FN)

print('测试集准确率: ', (correct * 1.0 / total))
print('测试集精确率：', p)
print('测试集召回率：', r)
print('测试集f1-: score', 2 * r * p / (r + p))
```

问题二的求解

solve.py

进行排序

```
import csv

def takeSecond(elem):
```

```
4  return int(elem[0])
5
6  def takeThird(elem):
7  return elem[2]
8
9
10 map = []
11 with open('ans3.csv', 'r', encoding='utf-8') as fr:
12 reader = csv.reader(fr)
13 for i in reader:
14 map.append((i[0], i[1], i[2]))
15 map2.sort(key=takeSecond(), reverse=True)
16
17 # 组内排序
18 map2 = []
19 ans = []
20 for i in map:
21 if len(map2) != 0 and i[0] == map2[-1][0]:
22 map2.sort(key=takeThird, reverse=True)
23 for j in map2:
24 ans.append(j)
25 map2.clear()
26 else:
27 map2.append((i[0], i[1], i[2]))
28
29 with open('ans5.csv', "w", encoding='utf-8')as file:
30 for i in map:
31 file.write(str(i[0]) + ',' + str(i[1]) + ',' + str(i[2]) + '
   \n')
```