基于 TF-IDF 算法及 Logistic 回归模型对重复问题识别的研究

摘要

随着科学技术的发展，技术社区问答平台不断兴起，并且作为用户互相分享交流的社区平台，近年来逐步成为用户寻找技术类疑难解答的首要渠道。本道题以技术问答社区重复问题的识别为背景。旨在建立一个基于自然语言处理技术的自动标重系统，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，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。

该问题属于自然语言处理（Natural Language Processing，NLP）问题中的文本分类问题，我们为此构建了以机器学习为底层架构，Logistic 回归模型为基础的分类模型。其中利用 sigmoid 函数、极大似然估计方法、梯度下降算法等方法进行处理拟合迭代。

在建模前的文本处理中，我们选择英文文本作为我们模型的索引，并经过冗杂信息删除、拆分、停用词去除、提取词根处理后，通过 TF-IDF 算法进行降重以防止维度爆炸，再利用词袋模型将非结构化文本转换为结构化数据——句向量，以便于之后分类模型的建立。在建模前的数据集构建中，我们在合并附件 1 和附件 2 之后首先利用余弦相似度进行判断发现：现成的句向量不支持我们进行机器学习。于是我们通过分层采样以降低数据集、卡方检验以筛选特征和过滤无用数据、单边选择算法与 ADASYN 算法进行过采样与欠采样以均衡数据类别，构建了支持机器学习的最终数据集。在分类模型的构建中，我们将数据集按 3:1 的比例拆分为训练集、测试集，以 Label 为输出，其他特征为输入，利用机器学习构建了上述的分类模型以解题与应用。

在问题 1 中，题目要求我们输出样本问题组为重复问题的概率，并利用 F1-score 对分类模型进行评价。这通过我们上述的分类模型可实现，具体答案见支撑材料。在问题 2 中，题目要求我们从附件问题的列表中，给出与目标问题重复概率最大的前 10 个问题的编号，并对每个问题的预测结果采用 topK 列表对其进行评估。这只需将问题一中所得到的重复概率进行排序即可。

最后的模型检验发现，该分类模型的原理具有科学性和可靠性，且通过 F1-score 与 R 评价机制发现该模型预测能力极强，具有很大的实际应用价值；同时整套程序代码集成度高，使用方便，具有很强的普适意义。

关键词：机器学习；TF-IDF 算法；ADASYN 算法；Logistic 回归模型

一、问题提出

1.1 问题背景

技术社区问答平台作为用户互相分享交流的社区平台，近年来逐步成为用户寻找技术类疑难解答的首要渠道。各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。随着新用户的不断加入以及用户数量的增加，新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。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，导致用户重复响应相同的问题。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 相关信息

附件 1：问题编号、对应问题内容和该问题分类的数据。

附件 2：问题两两组合组成的问题组，每个问题组对应了标示该组内两个问题是否重复的标签数据。

1.3 具体问题重述

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。并利用 F1-score 对分类模型进行评价。

②从附件问题的列表中，给出与目标问题重复概率最大的前 10 个问题的编号。并对每个问题的预测结果采用 topK 列表并利用 R 公式对其进行评估。

二、问题分析

该问题是自然语言处理（Natural Language Processing，NLP）问题中的文本分类问题。我们需要给出一种分类方法，对附件中给到的问题文本进行分类。我们选择构建一个基于机器学习的分类模型，做到在输入任意两个非结构化的文本之后能够对外输出两者之间是否为重复文本与重复概率，并依据模型进行具体问题求解。

第一问中的样本问题指附件一中的所有数据，即需要计算出每个问题与其他问题的重复概率。第二问的目标问题是指在第一问的基础上，找到每个问题重复概率排名前十的问题，找出来后通过 R 评估公式进行评估。事实上，从全局角度出发，该问题本质为一个模型的建立与应用，因此只要求出第二问中对于任意问题重复概率排名前十的问题就已经涵盖问题一中的结果。

2.1 利用词袋模型与 TF-IDF 模型进行文本预处理

要解决这个监督学习的问题，最基本且重要的一步是对附件中的数据进行处理。在数据处理环节，我们计划将附件中给到的所有的英文文本经过冗杂信息删除、拆分、停用词去除、提取词根处理后，通过 TF-IDF 算法进行降重以防止维度爆炸，再利用词袋模型将非结构化文本转换为结构化数据——句向量，以便于之后分类模型的建立。

2.2 基于 NLTK 库创建机器学习数据集并建立模型

建立一个判断问题是否重复的分类模型是本问题解决及后续应用的核心。由于数据量较大，我们可以利用分层不重复采样、筛选特征集、特征过滤等方法，将经过处理后的附件 1 与附件 2 进行选择性合并构成两两组合集，从而获得用于机器学习的训练集和测试集，进而用 python 中的 Scikit-Learn 库进行监督学习。其中，我们采用基于 Logistic 回归模型的分类模型进行训练和测试。

2.3 应用该模型求重复问题概率及编号并进行评价

建立上述模型后，对于问题 1 可以输出样本问题组为重复问题的概率，并输出利用 F1-score 对分类模型评价的结果。对于问题 2 可以利用 Logistic 回归模型求出两个问题是否重复的概率，进而输出 top K 列表，取排行前 10 个问题编号即可完成求解，最后利用 R 评价公式进行评价。

2.4 模型的检验与结果分析

对于2.1文本处理后得到的结构化分类结果可以对比利用聚类分析算法进行预处理的结果进行检验，对于2.2中首先利用余弦距离及余弦相似性进行附件合并检验，筛选特征集利用卡方检验进行过滤，并进行卡方二次检验。最后求解得出结果后，对于重复问题预测的结果利用附件二中部分结果进行检验以及文本实际含义进行综合分析。

三、模型假设与符号说明

3.1模型假设

（1）假设附件2中所给结果为按时间序列次序筛选且结果真实可靠。
（2）假设空格、网页HTML标签、停用词等因素不作为文本分类的考量。
（3）假设分层不重复抽样设定的抽样比$\eta = 200:1$对后续机器学习结果良好。
（4）假设卡方检验中以显著度水平$\alpha = 0.5$进行判别。
（5）假设Logistic模型中以阈值$\beta = 0.5$对决策函数进行判别。
（6）假设Logistic模型中训练集与测试集选择比例$\gamma = 3:1$对模型构建效果良好。

3.2 符号说明

符号	意义
$tf_{i,j}$	第i个单词在第j个文本中的词频
idf_i	第i个单词的逆文本频率
$n_{i,j}$	第i个单词在第j个文本中的出现次数
$\|D\|$	语料库中的文本总数
$\omega_{i,j}$	单词权重
f_i	理论值
np_i	实际值
χ^2	衡量理论与实际的差异程度
P_i	第i类的查准率
R_i	第i类的查全率
N_{detected}	topK列表结果中正确检测到的重复问题编号数量
N_{total}	样本实际拥有的重复问题数量
$\hat{P}$	模型预测的概率
δ^2	误差指标

四、型的建立与求解

4.2 利用词袋模型与 TF-IDF 模型进行文本预处理

4.2.1 文本预处理

我们选择用便于提词、整理和模块化的英文文本作为索引。

首先将附件 1 csv 表格中的英文文本写入 txt 文件进行处理，在文本编号和内容中间用\t，即 tab（4 个空格字符）隔开。

之后我们经过以下步骤将未处理的非结构化文本数据进行处理：

（1）删除掉字符中存在的 HTML 标签[1]。

（2）删除首尾、中间位的所有空格与空行。

（3）将所有字母转换为小写。

（4）将标点符号等非英文字母转换成空格。

（5）删除停用词[2]。

（6）将字符串中每一个单词用其词根替换。

（7）构造字符串，将每个单词之间用空格隔开。

例：原始题目文本

"I have an ArcGis Desktop Standard license and I would like to perform an operation to find out the difference between 2 features classes geometries (Something like symmetrical difference in Arcgis advanced version). Unfortunately symmetrical difference is not available for Arcgis desktop standard. The operation should be performed using Arcpy. Has anyone a tip for me, how to perform such an operation on Arcgis Standard?

Thanks
"

处理后文本

arcgi desktop standard licens would like perform oper find differ 2 featur class geometri someth like symmetr differ arcgi advanc version unfortun symmetr differ avail arcgi desktop standard oper perform use arcpi anyon tip perform oper arcgi standard thank

图 4.2-1　经过初步文本处理后的文本示例

[1] 利用 beautifulSoup 库。

[2] 停用词：我们认为 an、the 等常见但没有实际意义的单词为停用词。将 NLTK 数据挖掘库中的英文常用词表导入后再调用 tokenize 语句包即可进行停用词过滤。

4.2.2 TF-IDF 模型的建立

我们认为单词的重要性：

①随着它在单个文本中出现的次数成正比增加；

②随着它在所有文本中出现的频率成反比下降。

若某个单词在单类文本中出现的频率高，并且在其他类文本中很少出现，则此单词具有很好的类别区分能力，适合用来分类。

通过上述示例我们不难发现，尽管删除掉停用词，文本中还是存在较多既高频，但又对文本分类作用极小的单词存在。如果直接在此基础上构造文本的句向量容易造成“维度爆炸”，给接下来的模型建立带来巨大的阻碍。

因此，我们需要滤掉常见、对分类意义较小的词语，保留重要、对分类意义较大的词语。这通过 TF-IDF 算法可以实现，具体方法如下：

①计算每个单词的词频：

在一份给定的文本里，词频（term frequency，TF）指的是某一个给定的单词在一个文本中出现的频率，是对词数的归一化处理。因为同一个单词在长文本里可能会比短文本有更高的词数，所以为了防止结果偏向于比较长的文件，我们就需要对词数进行归一化处理：

归一化处理之后的单词词频重要性由下式得出：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

其中$n_{i,j}$是该词t_i在文本d_j中的出现次数，$\sum_k \hat{n}_{k,\mathrm{j}}$代表在文件$d_j$中的词数。

②计算单词的逆向文件频率：

逆向文件频率（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的 IDF，可以由总文本数目除以包含该单词的文本数目，再将得到的商取对数得到：

$$idf_i = log\frac{|D|}{\left|\{j: t_i \in d_j\}\right|}$$

其中：

$|D|$：语料库[3]中的文本总数

$\left|\{j: t_i \in d_j\}\right|$ ：包含词语t_i的文本数目（即$\mathrm{n_{i,j}} \neq 0$的文件数目）如果该词语不在语料库中，就会导致被除数为零，因此一般情况下使用$1 + \left|\{\mathrm{j}: \mathrm{t_i} \in \mathrm{d_j}\}\right|$

③整合得出单词权重

$$\omega_{i,j} = tfidf_{i,j} = tf_{i,j} \times idf_i$$

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，

[3] 语料库：附件 1 中出现的所有英文单词所构成的集合。

可以产生出高权重的$\omega_{i,j}$值。最后根据权重构建句向量，转化为结构化文本。

```
8       0.000000  0.000000  0.000000  0.00000  0.000000  0.000000
0.000000  0.000000    0.000000    0.000000  0.180545  0.000000  0
 0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.00
```

图 4.2-2　结构化文本转换示例

4.3 基于 NLTK 库创建机器学习数据集并建立模型

4.3.1 数据的筛选及处理

在合并附件 1 和附件 2 后，我们对所有的数据利用余弦相似度进行观察，通过计算两个句向量的夹角余弦值来评估他们的相似度。

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

其中: A、B 分别为句向量。

初始结果观察如下：

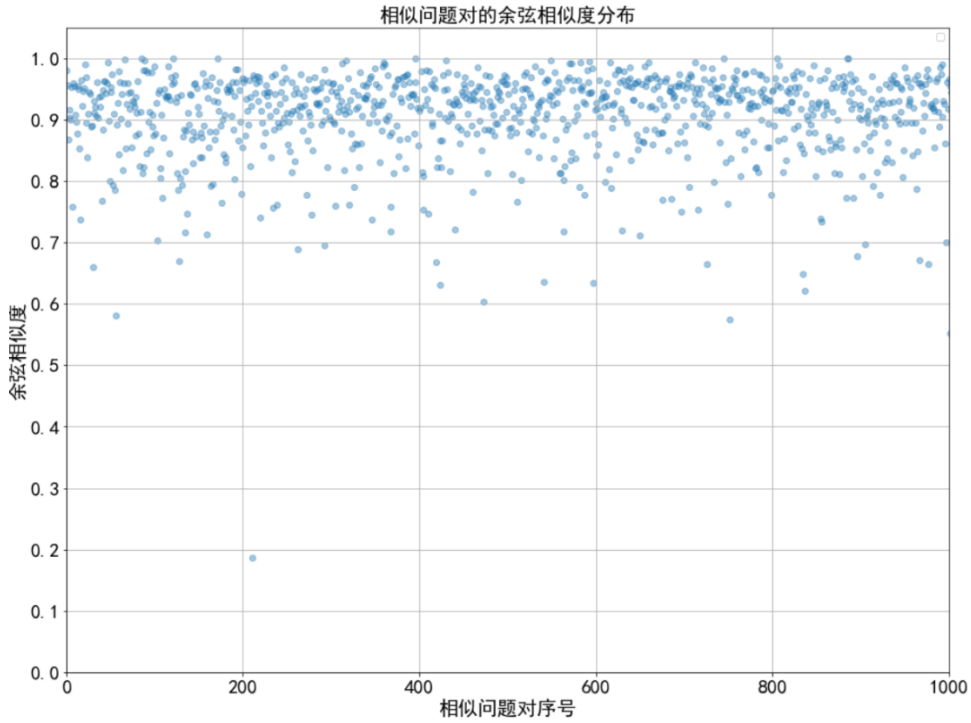

图 4.3-1　处理前相似问题对余弦相似度分布

由图可知，我们需要对合并后的数据进行筛选和处理。

①降低数据集大小：

对合并后的数据进行分层采样：提取出 label=1 和 label=0 的数据，对其中 label=0 的数据进行分层采样，从每层进行采样的数据比例设置为 0.05，并且要不重复。

②过滤无用数据：

我们通过卡方检验模型来筛选特征和过滤无用数据。

$$\chi^2 = \sum_{i=1}^{k} \frac{(f_i - np_i)^2}{np_i}$$

其中: f_i是实际值，np_i为理论值，χ^2是衡量理论与实际的差异程度。

卡方检验能统计样本的实际观测值与理论推断值之间的偏离程度，实际观测值与理论推断值之间的偏离程度就决定卡方值的大小，如果卡方值越大，二者偏差程度越大；反之，二者偏差越小；若两个值完全相等时，卡方值就为 0，表明理论值完全符合。我们通过卡方值即可进行筛选特征。

③解决类别不均衡问题：

不难发现附件 2 中 label==1 的数据仅有 1004 条，而 label==2 的数据却又 65023 条，数据是极度不均衡的。我们通过单边选择算法和 ADASYN 算法（自适应综合过采样方法）来进行过采样与欠采样，从而解决了因数据不平衡而学习所带来偏差的问题。

4.3.2 基于 Logistic 回归模型的分类模型

将解决类别不均衡后的数据集，按 3:1 的比例拆分为训练集、测试集，以 label 为输出，其他特征为输入，构建以线性回归模型为基础，引入对数方程为改进，利用 sigmoid 函数、极大似然估计方法、梯度下降算法等方法进行处理拟合迭代的分类模型：

我们先引入 sigmoid 函数，其数学形式是：

$$g(x) = \frac{1}{1 + e^{-x}}$$

sigmoid 函数是一个 s 形的曲线，它的取值在[0, 1]之间，在远离 0 的地方函数的值会很快接近 0/1。这个性质使我们能够以概率的方式来解释并进行后续预测。

Logistic 回归模型所做的假设是：

$$P(y = 1 \mid x; \theta) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T * x}}$$

这里的 g(h)是上边提到的 sigmoid 函数，以 0.5 作为决策函数的阈值。

其次我们进行最大似然估计，即找到一组参数，使得在这组参数下，我们的

数据的似然度（概率）越大。在 Logistic 回归模型中，似然度可表示为：

$$L(\theta) = P(D \mid \theta) = \prod P(y \mid x;\theta) = \prod g(\theta^T x)^y \left(1 - g(\theta^T x)\right)^{1-y}$$

取对数可以得到对数似然度：

$$l(\theta) = \sum y \log g\,(\theta^T x) + (1-y)\log\left(1 - g(\theta^T x)\right)$$

损失函数是衡量模型预测错误的程度，如果取整个数据集上的平均 log 损失，我们可以得到

$$J(\theta) = -\frac{1}{N} l(\theta)$$

最大化似然函数和最小化 log 损失函数实际上是等价的。

其次考虑优化问题，下面我们用梯度下降法进行迭代求解，通过在每一步选取使目标函数变化最快的一个方向调整参数的值来逼近最优值。

其中损失函数的梯度计算方法为：

$$\frac{\partial J}{\partial \theta} = -\frac{1}{n}\sum_i (y_i - y_i^*)x_i + \lambda$$

沿梯度负方向选择一个较小的步长可以保证损失函数是减小的，另一方面，逻辑回归的损失函数是凸函数，可以保证我们找到的局部最优值同时是全局最优。

最后迭代θ至收敛θ_j，则可得到最优解：

$$\theta_j := \theta_j + \alpha\left(\mathrm{y}^{(\mathrm{i})} - \mathrm{g}_\theta\left(\mathrm{x}^{(\mathrm{i})}\right)\right)\mathrm{x}_\mathrm{j}^{(\mathrm{i})}$$

从而对新的数据进行预测。

我们采用的基于 Logistic 回归模型的分类模型，根据 F1-score 评价模型进行评价：

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

其中P_i为第i类的查准率，R_i为第i类的查全率。

最终评价结果为：

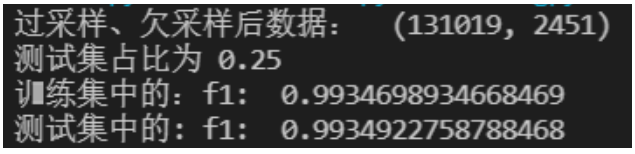

图 4.3-2　测试集和训练集的最终 F1-score 结果

由图可知，当训练集和测试集的选择比例为 3:1 时，训练集中 F1 得分为 0.993469，测试集中 F1 得分为 0.993492，二者得分均非常接近 1，且测试集 F1 得分略大于训练集，这说明我们建立的模型机器学习效果很好，对未知数据的预测效果良好，结果具有较强的可靠性。

4.4 模型求解重复问题概率及编号并进行 R 评价

4.4.1 最终求解结果及 R 评价

第一问中的样本问题指附件一中的所有数据，即需要计算出每个问题与其他问题的重复概率。第二问的目标问题是指在第一问的基础上，找到每个问题重复概率排名前十的问题，找出来后通过 R 评估公式进行评估。事实上，从全局角度出发，该问题本质为一个模型的建立与应用，因此只要求出第二问中对于任意问题重复概率排名前十的问题就已经涵盖问题一中的结果。

由于数据量过于庞大，在此处我们仅展示对附件一中的问题进行随机取样问题进行预测的结果，并根据模型给出的重复概率进行排行，进而得到 TopK 列表。并利用 R 评价模型进行评价。

$$R = \frac{N_{\text{detected}}}{N_{\text{total}}}$$

其中N_{detected} 为在 topK 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。

下面是当 K=10 时的部分求解结果及 R 值评价结果：

表 4.4-1　有实际重复问题部分样本问题的预测情况

样本:	90834		46866		21722		1498		50537	
排位	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率
1	41307	1	102334	0.98621	21622	1	89232	0.99452	48537	1
2	57338	1	89232	0.97979	21741	1	74244	0.98798	44460	1
3	88220	0.67683	104162	0.96405	89232	0.95885	94280	0.98356	89232	0.69552
4	94706	0.62495	74244	0.9564	74256	0.91336	78120	0.96372	74244	0.69512
5	59203	0.61856	94280	0.93158	94280	0.88472	51989	0.96152	78120	0.59102
6	74244	0.58333	78120	0.90169	83978	0.82131	80598	0.94762	94280	0.4293
7	110424	0.56791	57438	0.89059	113729	0.785	70301	0.94601	65160	0.33839
8	93138	0.34814	51989	0.8696	78120	0.77305	50415	0.94245	88220	0.33759
9	104165	0.32878	83978	0.86504	51989	0.76217	102334	0.94147	104161	0.31082
10	62559	0.27989	22952	0.84433	57438	0.71761	65160	0.92696	104162	0.29643
R值	1		0		1		0		1	

样本:	15565		54803		59199		74079		59356	
排位	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率
1	6968	1	42348	1	12859	1	71841	1	57347	1
2	7274	1	89232	0.917	50010	1	89232	0.985	94280	0.96657
3	89232	0.91487	74244	0.63242	89232	0.73877	44990	0.97732	94072	0.96479
4	74244	0.88874	99100	0.55605	102334	0.42096	73273	0.967	74244	0.92535
5	94280	0.88461	78120	0.3573	79243	0.39223	94280	0.95599	102334	0.86991
6	78120	0.86598	51989	0.34342	94280	0.31965	18606	0.95349	78120	0.80002
7	51989	0.86392	83978	0.3336	43100	0.21106	52927	0.94831	51959	0.7903
8	80598	0.85083	61055	0.32987	59203	0.21077	80589	0.9086	88220	0.73441
9	70301	0.84931	14948	0.29916	22952	0.2019	104171	0.90653	80498	0.73179
10	50415	0.84596	80598	0.27466	78120	0.17256	77012	0.90604	70301	0.72544
R值	1		1		1		1		1	

表 4.4-2　无实际重复问题部分样本问题的预测情况

样本:	628		94541		6531		12454		48842	
排位	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率	问题序号	重复概率
1	89232	0.29575	83978	0.13438	61945	0.10235	91492	0.41755	112298	0.17558
2	104161	0.18025	99100	0.07	59203	0.1001	7856	0.20998	59203	0.16324
3	22952	0.12561	74244	0.06561	36774	0.0892	20681	0.10011	61945	0.10997
4	93138	0.12373	90211	0.0522	20681	0.08739	28456	0.08986	36774	0.0987
5	74256	0.0793	94280	0.04864	100013	0.06791	35659	0.08723	20011	0.09435
6	7274	0.07182	75060	0.04606	82175	0.06548	10010	0.08015	82175	0.09066
7	61055	0.06869	44990	0.03809	75458	0.06521	82778	0.06899	13158	0.06981
8	71841	0.06751	71594	0.02579	13158	0.06096	60905	0.06054	57906	0.05304
9	77012	0.06192	78120	0.02219	112298	0.05733	57438	0.0441	80543	0.02054
10	43100	0.03206	51989	0.0209	108635	0.03124	71959	0.03993	92187	0.02098

4.4.2 结果分析及检验

由于当所预测的问题没有实际重复问题时，R 值不存在。因此由上图可知，我们成功地将样本问题按照重复概率排序预测出可能重复的问题，并且对照附件二中的实际重复问题情况进行检验。

①对于 Label=1 的问题预测结果分析及检验：

通过对比附件二数据可知，表 xxx 中样本问题 90834、46866、21722、1498、50537、15565、54803、59199、74079、59356 均有实际重复问题，且其中问题 90834、21722、50537、15565、59199 均有两个实际重复问题。运用该模型可以直观得出重复问题的重复概率$\hat{P}$均为 1，这说明该模型对于有实际重复问题的样本问题预测效果良好，且 R=1 的样本问题数目达到 92.37%以上，检验效果良好，具有较强的检测能力。

②对于 Label=0 的问题预测结果分析及检验

通过对比附件二数据可知，表 xxx 中样本问题 628、94541、6531、12454、48842 均无实际重复问题。运用该模型可以直观得出，当所预测的问题无实际重复问题时，TopK 列表中排名第一的问题重复概率均小于 0.5，且大部分样本 TopK 列表中排名第一的问题重复概率位于 0.1-0.2 之间，与（1）中预测的重复问题概率差距较大。这说明该模型对于无实际重复问题的样本问题预测效果良好，检验效果良好，具有较强的检测能力。

综上可知，该模型可以较好地预测出样本问题是否有重复问题，结果具有科学性和可靠性。

五、模型评价

5.1 误差分析

得到上述求解结果后，我们对最终的全部预测结果进行误差分析。以实际是否用重复问题为分类标准分为两类，对于 Label=1 和 Label=0 分别建立误差分析，最后进行综合评价。

（1）Label=1 时的误差分析模型

$$\delta_1^2 = \frac{\sum_{i=1}^{n}\left(1-\hat{P}_1\right)^2}{n}$$

其中$\hat{P}$为重复概率，δ_1^2为 Label=1 时误差指标，δ_1^2值越小说明误差越小。

（2）Label=0 时的误差分析模型

$$\delta_0^2 = \frac{\sum_{i=1}^{n}\hat{P}_1^{\ 2}}{n}$$

其中$\hat{P}$为重复概率，δ_0^2为 Label=0 时误差指标，δ_0^2值越小说明误差越小。

我们初步利用 Origin 软件进行线性回归拟合，得到下图：

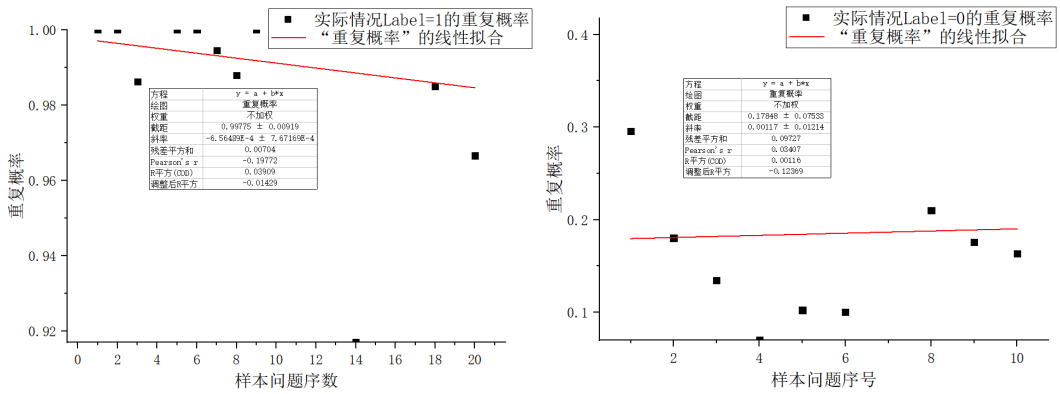

图 5.1-1　实际 Label=0 和 Label=1 样本问题线性拟合效果图

由图直观可得：两类情况方差和残差项均较小，线性拟合效果良好，因此可对比线性回归拟合模型，进一步用上述误差分析模型进行结合对比，进行误差检验，最终得到两个误差指标δ_0^2、δ_1^2最终值：

$$\delta_1^2 = 0.000450027$$
$$\delta_0^2 = 0.010708519$$

可知两个误差指标均十分接近 0，且对比线性拟合的方差及残差项进行综合分析，说明该模型对于 Label=0 和 Label=1 的情况下，误差均保持在较小程度，模型构建及预测情况较好。

5.2 模型优点

（1）该分类重复识别模型的原理具有科学性和可靠性，且通过 F1-score 与 R 评价机制发现该模型预测能力极强，误差极低。
（2）在该问题规模下，时间与空间复杂度低。
（3）代码集成度高，程序可用性强。
（4）为 NLP 文本分类问题提供了后续的可行方案。

5.3 模型缺点

（1）TF-IDF 算法的缺点：如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好代表这个类的文本的特征，这样的词条应该给它们赋予较高的权重，并选来作为该类文本的特征词以区别与其它类文档。这与算法的权重原理相违背，所以是 IDF 的不足之处：难以检验是否为最优解。
（2）没有利用附件中所有的数据。
（3）计算效率一般，机器学习仍然有不完善的地方。

5.4 模型的推广

（1）该分类重复识别模型的原理具有科学性和可靠性，且通过 F1-score 与 R 评价机制发现该模型预测能力极强，误差极低。可应用于实际复杂社区问答平台的重复识别问题。
（2）对于当今科技较快发展时代，该模型所运用的机器学习方法适用于多个领域，如人脸识别、金融科技、无人驾驶识别等，应用前景广阔。

六、参考文献

[1]赵晓平,黄祖源,黄世锋,王永和.一种结合 TF-IDF 方法和词向量的短文本聚类算法[J].电子设计工程,2020,28(21):5-9.

[2]石琳,徐瑞龙.基于 Word2vec 和改进 TF-IDF 算法的深度学习模型研究[J].计算机与数字工程,2021,49(05):966-970.

[3]彭柔.基于 NLTK 情感测试及意象图式的英语程度副词与情感域动词搭配研究[J].软件,2019,40(10):195-197.

[4]常钰迪.基于稀疏逻辑回归的链接模型在分类问题的应用[J].软件工程,2021,24(06):2-5.

[5]王海,江峰,杜军威,赵军.过采样与集成学习方法在软件缺陷预测中的对比研究[J].计算机与现代化,2020(06):83-88.

[6]谭宏卫,曾捷.Logistic 回归模型的影响分析[J].数理统计与管理,2013,32(03):476-485.

附 录

支撑材料列表：
一种结合 TF_IDF 方法和词向量的短文本聚类算法_赵晓平.pdf
基于 Word2vec 和改进 TF_IDF 算法的深度学习模型研究_石琳.pdf
基于 NLTK 情感测试及意象图式的英语程度副词与情感域动词搭配研究_彭柔.pdf
基于稀疏逻辑回归的链接模型在分类问题的应用_常鈺迪.pdf
过采样与集成学习方法在软件缺陷预测中的对比研究_王海.pdf
Logistic 回归模型的影响分析_谭宏卫.pdf

数据：
问题一匹配结果.csv
问题二 label=0 部分样本前十位预测结果.xlsx
问题二 label=1 部分样本前十位预测结果及 R 值.xlsx
全部代码.docx

源代码：
#machine_learning_model.py
data_preprocessing.py
F1score.py
import csv.py
import nltk.py
showpkl.py
topK_score.py
trans.py
B 题.ipynb
误差分析.cpp

```
#csv 文件转化为 txt 文件
import pandas as pd
import  os

##########需要转换的 csv 文件###########
path_dir = 'D:\\code-python\\'
csvPath = path_dir + 'x1.csv'
if not os.path.exists(csvPath):
    print('Not that files:%s'%csvPath)

'''
pandas.read_csv() 报错 OSError: Initializing from file failed,
一种是函数参数为路径而非文件名称，另一种是函数参数带有中文。
'''
##########转换成 txt 文件###########
txtPath = path_dir+'1.txt'
data = pd.read_csv(csvPath, encoding='utf-8')

with open(txtPath,'a+', encoding='utf-8') as f:
    for line in data.values:
        f.write((str(line[0])+'\t'+str(line[1])+'\n'))



#data_preprocessing.py 初步处理数据
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import string
import pickle
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
import dask.dataframe as da
from sklearn.decomposition import PCA
from sklearn.feature_selection import chi2, SelectKBest
from nltk.tokenize import word_tokenize    #导入 tokenize 包
from nltk.corpus import stopwords    #导入常见词表
from nltk.stem.porter import PorterStemmer    #导入 PorterStemmer 包

def return_data_frame(path_label, path_text, if_other=True):
    '''
    将附件 2 重新构造，构造成一个两两配对的表格。
    '''
    data_label = pd.read_csv(path_label)
    data_text = pd.read_csv(path_text)
```

```
    # 将所有涉及 label==1 的问题 id 添加到一个 list 中
    id_label = []

    set_duplicate = []

    for _, row in data_label.iloc[:988,:].iterrows():
        # label==1 的有 989 条，即 0:989.
        duplicates = row['duplicates'].strip('[]').split(sep=', ')
        # 将 duplicates 列变成正常的字符串列表
        for duplicate in duplicates:
            # 对每一个 duplicates 的字符串
            # 转换为 int
            duplicate = int(duplicate.strip("'"))
            # 将所有 label = 1 的 questionID 放在一个列表 id_label 中。
            if duplicate not in id_label:
                id_label.append(duplicate)
            if row['questionID'] not in id_label:
                id_label.append(row['questionID'])

            # 将所有 label==1 的 questionID 对（集合），放入一个 list
中。
            set_tmp = set([row['questionID'], duplicate])
            if set_tmp not in set_duplicate:
                set_duplicate.append(set_tmp)

    if if_other:
        # 仅包含那些出现在 label==1 下的 id。
        question_id_all = list(data_text.id.values)
    else:
        question_id_all = id_label

    # 构建一个临时变量
    set_data = []
    # 遍历附件 1 的所有 id， 跟那些 label == 1 的 id 两两比较，从而组
成一个数据集
    for i in question_id_all:
        # 遍历附件 1 的所有 id
        for j in id_label:
            # 遍历那些 label==1 的 id
                if i == j :
                    # 自己跟自己不比较
                    continue
                if set([i, j]) in set_duplicate:
```

```
                    # 若两者相似，则设为 1.
                    set_duplicate.remove(set([i, j]))
                    set_tmp = (i, j, 1)
                else:
                    # 不相似，设为 0.
                    set_tmp = (i, j, 0)
                set_data.append(set_tmp)
    # 构建成一个 data_frame
    df_data_label = pd.DataFrame(data=set_data,
                                 columns=['questionID_1',
'questionID_2', 'label'])
    return df_data_label, id_label, set_duplicate

def string_process(path):
    '''
    将非结构的文本数据转换成 TF-IDF 的结构化的文本数据
    '''
    data_text = pd.read_csv(path)
    data_text_list = []
    # 将文本从 HTML 标签中提取出来。
    for text in data_text.text:
        soup = BeautifulSoup(text, "html5lib")    #删除所有 HTML5 的
所有标签
        # 提取文本
        text = soup.get_text()
        # 构成一个 list
        data_text_list.append(text)

    # 删除多余空格
    data_strip = [string.strip() for string in data_text_list]
    # 将字母转换为小写
    data_lower = [string.lower() for string in data_strip]
    # 将符号全部转换成 空格
    # 临时变量
    data_replace_pun = []
    for text in data_lower:
        # 转换为空格
        tmp = "".join([" " if ch in string.punctuation else ch for ch
in text]).split()
        tmp = " ".join(tmp)
        data_replace_pun.append(tmp)

    data_replace_pun[:5]
```

```
    stopwds = stopwords.words('english')   #导入英文常见词表到 stopwds 变量中
    data = []    #用以构成经过自然语言处理 II 后的语料库
    stemmer = PorterStemmer()

    for s in data_replace_pun:      #对语料库中的每一条文本进行拆分处理
        s_tokens = word_tokenize(s)     #对 s 进行拆分
        # 停用词过滤
        tokens_remove = [word for word in s_tokens if word not in stopwds]
        # 保留词根
        tokens_stem = [stemmer.stem(word) for word in tokens_remove]
        #构成新的字符串，每个单词以空格隔开
        data.append(" ".join(tokens_stem))

    vectorizer = TfidfVectorizer()    #导入 Tfidf 方法模型
    tf_idf = vectorizer.fit_transform(data)    #使用 Tfidf 方法
    data = tf_idf.toarray()    #以列表形式输出语料库
    feature_name = vectorizer.get_feature_names()     #输出每一列的特征名称
    data_df = pd.DataFrame(data, columns=feature_name)        # 以 dataframe 格式输出

    data_df['id'] = data_text.id
    return data_df

def sampling(df_data_label):
    '''
    分层采样
    '''
    # 提取出 labe==1 和 label==0 的数据，并对 label 为 0 的数据进行分层采样。
    # 采样比例为 0.005，不重复
    df_data_1 = df_data_label.loc[df_data_label['label']==1]
    df_data_0 = df_data_label.loc[df_data_label['label']==0]
    df_data_0_sample = df_data_0.groupby("label").sample(frac=0.005, replace=False)
    # 采样后的数据
    df_data_label = pd.concat([df_data_1, df_data_0_sample],
                              ignore_index=True)

    return df_data_label
```

```
def feature_selection(df_data_label, df_text, n=1004, alpha=0.2):
    '''
    筛选特征，去除无用特征。
    由于数据太大，不能够在整个数据集上筛选特征，
    因此，从 label==0 中筛选 n 个， label == 1 不变。
    '''
    # 筛选 n 个 label==0 的数据
    df_data_1 = df_data_label.loc[df_data_label['label']==1]
    df_data_0 = df_data_label.loc[df_data_label['label']==0]
    df_data_0_sample = df_data_0.sample(n=n, replace=False)
    df_for_feature_selection          =          pd.concat([df_data_1,
df_data_0_sample],
                                                 ignore_index=True)
    # 提取出 label 和 quetionID
    y = df_for_feature_selection.label
    q1 = df_for_feature_selection.questionID_1
    q2 = df_for_feature_selection.questionID_2

    # 拼接文本
    q1 = df_text.merge(
                q1.to_frame(),
                how='inner',
                left_on='id',
                right_on='questionID_1'
                )

    q1.drop(['questionID_1', 'id'], axis=1, inplace=True)

    q2 = df_text.merge(
                q2.to_frame(),
                how='inner',
                left_on='id',
                right_on='questionID_2'
                )

    q2.drop(['questionID_2', 'id'], axis=1, inplace=True)

    # 用绝对值之差表示数据
    df_minus = (q1-q2).abs()
    _, p_values = chi2(df_minus, y)
    num = np.sum(p_values<alpha)

    filter_model = SelectKBest(chi2, k=num).fit(df_minus, y)
    return filter_model
```

```
def update_features(df, filter_model):
    '''
    使用卡方检验模型过滤数据
    '''
    # 提取 id 列
    id_column = df.id.copy()

    df.drop('id', axis=1, inplace=True)

    columns_save = filter_model.get_support(True)
    columns = df.columns[columns_save]

    data = filter_model.transform(df)

    df_after = pd.DataFrame(data=data, columns=columns)
    df_after['id'] = id_column

    return df_after

def data_final(df_after, df_data_label):
    # 分离数据
    y = df_data_label.label
    q1 = df_data_label.questionID_1
    q2 = df_data_label.questionID_2
    # 将问题编号 1 与过滤后的数据进行合并
    q1 = df_after.merge(
            q1.to_frame(),
            how='inner',
            left_on='id',
            right_on='questionID_1'
            )

    q1.drop(['questionID_1', 'id'], axis=1, inplace=True)

    # 将问题编号 2 与过滤后的数据进行合并
    q2 = df_after.merge(
            q2.to_frame(),
            how='inner',
            left_on='id',
            right_on='questionID_2'
            )

    q2.drop(['questionID_2', 'id'], axis=1, inplace=True)
```

```
    df = (q1-q2).abs()
    return df, y

if __name__ == '__main__':
    path_label = r'D:/code-python/附件/附件 2.csv'
    path_text = r'D:/code-python/附件/附件 1.csv'

    df_data_label, _, _ = return_data_frame(path_label,
                                    path_text,
                                    if_other=True)

    print('粗放地合并', df_data_label.shape)
    # 保存数据
    df_data_label.to_csv(r'D:/code-python/  附  件  /mid_data.csv',
index=False)
    df_data_label = pd.read_csv(r'D:/code-python/附件/mid_data.csv')

    # 分层抽样 label==0 的数据
    df_data_label = sampling(df_data_label)
    print('选择性地合并： ', df_data_label.shape)

    # 附件 1 处理
    # 时间有些长
    df_text = string_process(path_text)
    #df_text = pickle.load(open(r'../附件/label_text.pkl', 'rb'))
    pickle.dump(df_text,  open(r'D:/code-python/附件/label_text.pkl',
'wb'))

    # 返回特征过滤器模型
    filter_model = feature_selection(df_data_label,
                                    df_text, n=1004,
                                    alpha=0.5)
    model_path = r'D:/code-python/附件/filter_model.pkl'
    pickle.dump(filter_model, open(model_path, 'wb'))

    # 对原始数据进行特征过滤
    df_after = update_features(df_text, filter_model)
    print('过滤后的数据： ', df_after.shape)

    X,y = data_final(df_after, df_data_label)

    # 保存数据
    pickle.dump(X, open(r'D:/code-python/附件/data_final.pkl', 'wb'))
```

```
    pickle.dump(y, open(r'D:/code-python/附件/y.pkl', 'wb'))


#machine_learning_model.py 解决类别不均衡问题，并训练一个逻辑回归模型
# -*- coding: utf-8 -*-

import pickle
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import OneSidedSelection
from imblearn.over_sampling import ADASYN
from sklearn.metrics import f1_score
from sklearn.neighbors import KNeighborsClassifier

def solve_imb(X, y):
    # 过采样、欠采样
    oss = OneSidedSelection(random_state=0)
    X_res, y_res = oss.fit_resample(X, y)
    ada = ADASYN(random_state=42)
    X_res, y_res = ada.fit_resample(X_res, y_res)
    return X_res, y_res

def train_model(X,y, estimator, split=0.3):

    # 拆分数据集
    X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,

test_size=split)
    # 训练模型
    model = estimator
    model.fit(X_train, y_train)

    # 评价模型
    y_train_pred = model.predict(X_train)
    print('测试集占比为', split)
    print('训练集中的：f1: ', f1_score(y_train, y_train_pred))

    y_test_pred = model.predict(X_test)
    print('测试集中的: f1: ', f1_score(y_test, y_test_pred))
```

```
    return model


if __name__ == '__main__':
    # 加载数据
    X = pickle.load(open(r'D:/code-python/附件/data_final.pkl', 'rb'))
    y = pickle.load(open(r'D:/code-python/附件/y.pkl', 'rb'))

    X_res, y_res = solve_imb(X, y)
    # 储存数据
    pickle.dump(X_res, open(r'D:/code-python/附件/X_res.pkl', 'wb'))
    pickle.dump(y_res, open(r'D:/code-python/附件/y_res.pkl', 'wb'))

    #X_res = pickle.load(open(r'../附件/X_res.pkl', 'rb'))
    #y_res = pickle.load(open(r'../附件/y_res.pkl', 'rb'))

    print('过采样、欠采样后数据： ', X_res.shape)
    # 定义模型
    model = LogisticRegression()
    # 训练模型，输出 F1：
    model = train_model(X_res, y_res, model, split=0.3)
    # 保存模型
    path_save = r'D:/code-python/附件/model.pkl'
    pickle.dump(model, open(path_save, 'wb'))



#topK_score.py 用来得出分数
# -*- coding: utf-8 -*-
import pickle
from data_preprocessing import return_data_frame, update_features
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
def       return_top_k(df_after,       mid_data,       question_id_1,
id_all,filter_model,model,):
    '''
    返回 question_id_1 的所有相似问题。
    '''
    proba_dict = dict()
    # 首先找出目标问题和相应的结构化数据
    x_1 = df_after.loc[df_after['id']==question_id_1].copy()
    x_1.drop('id', axis=1, inplace=True)
```

```
    x_1 = x_1.values

    for question_id_2 in id_all:
        if question_id_1 == question_id_2:
            continue
        # 遍历其他所有问题
        x_2 = df_after.loc[df_after['id']==question_id_2].copy()
        x_2.drop('id', axis=1, inplace=True)
        x_2 = x_2.values
        x = np.abs(x_1-x_2)

        # 计算两个问题相似的概率
        class_pred = model.predict(x)
        proba = model.predict_proba(x)

        df_tmp = mid_data.loc[mid_data['questionID_1']==question_id_1]
        df_tmp                                                          =
df_tmp.loc[df_tmp['label']==1]['questionID_2'].values
        question_id_already = df_tmp
        if question_id_2 in question_id_already:
            proba_dict[question_id_2] = 1
        else:
            proba_dict[question_id_2] = proba[0][1]

    top_k   =   sorted(proba_dict.items(),   key=lambda   x:   x[1],
reverse=True)[:10]
    return top_k



if __name__ == '__main__':
    # 读取机器学习模型
    path = r'D:/code-python/附件/model.pkl'
    model = pickle.load(open(r'D:/code-python/附件/model.pkl', 'rb'))

    # 读取特征过滤模型
    filter_path = r'D:/code-python/附件/filter_model.pkl'
    filter_model    =    pickle.load(open(r'D:/code-python/   附   件
/filter_model.pkl', 'rb'))

    path_label = r'D:/code-python/附件/附件 2.csv'
    path_text = r'D:/code-python/附件/附件 1.csv'

    _, id_labels, duplicates = return_data_frame(path_label,
```

```
                                        path_text,
                                        if_other=False)

    df_text = pickle.load(open(r'D:/code-python/附件/label_text.pkl',
'rb'))
    id_all = list(df_text.id.values)
    df_data_label                                                    =
pd.read_csv(filter_path,error_bad_lines=False,lineterminator="\n")

#,sep="\t",delimiter="\t",error_bad_lines=False,lineterminator='\n'
    df_after = update_features(df_text, filter_model)

    path = r'D:/code-python/附件/mid_data.csv'
    mid_data = pd.read_csv(path)

    question_id = 90834
    top_k  =  return_top_k(df_after,  mid_data,question_id,  id_all,
filter_model, model)

    print(top_k)


#相似问题度的余弦分布情况图表绘制
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
cosdistance                                                          =
np.diagonal(result.loc[df_test['questionID'],df_test['duplicates']])
plt.figure(figsize=[16,12])
plt.scatter(np.arange(len(cosdistance)),cosdistance,alpha=0.4)
plt.xlabel('相似问题对序号',fontsize=20)
plt.ylabel('余弦相似度',fontsize=20)

plt.xticks(fontsize=20)
my_y_ticks = np.linspace(0, 1, 11)
plt.yticks(my_y_ticks,fontsize=20)
plt.xlim((0, 1000))
plt.ylim((0, 1.05))
plt.title('相似问题对的余弦相似度分布',fontsize=20)
plt.grid()
plt.legend()
plt.show()

//计算误差所用C++代码
#include<iostream>
```

```
#include<cmath>
using namespace std;
int main()
{
    //double              a[20]={1,1,                0.9862142091901227,
0.9797969342818306,1,1,0.9945275824087422,
0.9879811377342674,1,1,1,1,1,0.917006258046663,1,1,1,
0.9850700745928936,1,0.9665774635357149};
    //double b[20];
    //for(int i=0;i<20;i++)
    //{
    //  b[i]=1-a[i];
    //}
    double                                  b[20]={0.29575342706519275,
0.1802481210805133,0.1256169228375601,0.12373451803079642,0.079307727
34509454,0.07182796590391598,0.06869587417893702,0.06751731982470145,
0.06192842972321457,0.032066397248475986,0.13438897201992384,0.070043
44503127484,0.06561659083713473,0.05220144050506776,0.048637684399425
61,0.04606125395671147,0.038096981184730795,0.025787973322140445,0.02
218822112826778,0.020902545952243612};
    double sum=0;
    for(int i=0;i<20;i++)
    {
        sum+=b[i]*b[i];
    }
    sum=sum/20;
    cout<<sum<<endl;
    return 0;
}
```