基于多属性量化综合评价的小学数学应用题相似性与难度度量研究

摘要

本文通过参考小学数学应用题实际背景，综合有关权威网站与文献，我们总结出小学数学应用题相似性与难度度量的多个角度，并进行对应的数据量化以实现问题的解决。

针对问题一，构建相似性度量方法时，我们的目的是为了能够精准地推荐给用户相关的练习题，提高其数学能力。为此，我们综合了多个有关权威网站的题型解析与相关文献的总结，确定了以下几个角度：应用题题型、标准解答是否需要画图解题、文本的相似性、应用真实场景进行相似性评估，并对其进行了数据量化，以实现相似性的精确度量。针对文本相似性，我们创新引入了 SimHash 算法进行高效率高精度的文本分析。通过搭建不同属性的量化矩阵，经过优化调整后的加权计算得到应用题相似性最终矩阵。

针对问题二，与问题一类似，我们通过结合小学生的学习实际与有关权威网站的数据，决定从多个角度进行小学数学应用题难度的评估，并进行数据量化，以实现难度的精确度量。难度评估属性囊括题目长度、题目长度、未知量个数、已知量个数、标准解答是否需要画图解题、涉及几位数的运算、加减号个数、乘号个数、除号个数、原子公式个数、方程元数、标准解答是否涉及分数、应用题题型。凭借收集到的小学数学应用题数据，我们进行属性的量化，并使用熵权法进行信息熵的分析，以使得难度评估更加客观。在此基础上结合 TopSis 方法，参考正理想解与负理想解，得出各应用题的最终难度评估。

针对问题三，我们针对相似度与难度两种分类情况，基于问题一与问题二所得到的两种不同结果形式采取针对性的分类算法。对于按相似度分类，我们采用 AGNES 算法进行分类；对于按难度分类，我们采用 K-means 算法进行分类。同时我们进行两类算法复杂度的评估，并讨论其是否能适用于更大规模的题库。

针对问题四，我们考虑将附件 2 中的 10 个应用题加入问题一的相似性度量模型，与附件 1 中的 100 个应用题共同进行相似性度量，对于输出的相似性度量结果再进行进一步的分析总结。同时我们进行相似性度量算法复杂度的评估，并讨论其是否能适用于更大规模的题库。

关键词：　多属性评估　相似性度量　SimHash 算法　熵权法　TopSis 方法　AGNES　K-means

目录

1 问题重述 …… 1
1.1 问题背景 …… 1
1.2 问题要求 …… 1
2 问题分析 …… 1
3 模型假设 …… 4
4 符号说明 …… 5
5 模型建立与求解 …… 5
5.1 问题一模型的建立与求解 …… 7
5.1.1 数据预处理 …… 7
5.1.2 应用题文本相似性度量矩阵建模 …… 8
5.1.3 应用题相似性最终矩阵建模 …… 9
5.2 问题二模型的建立与求解 …… 10
5.2.1 数据预处理 …… 10
5.2.2 难度评估属性权重量化建模 …… 11
5.2.3 应用题难度评估建模 …… 12
5.3 问题三模型的建立与求解 …… 14
5.3.1 按相似度分类 …… 14
5.3.2 按难度分类 …… 16
5.3.3 算法评估 …… 17
5.4 问题四模型的建立与求解 …… 18
5.4.1 求解过程 …… 18
5.4.2 评估算法 …… 19
6 模型评价和推广 …… 19
6.1 模型的评价 …… 19
6.1.1 模型的优点 …… 19
6.1.2 模型的缺点 …… 20
6.2 模型的推广 …… 20
7 参考文献 …… 20
8 附录 …… 21
8.1 支撑材料列表 …… 21
8.2 建模代码 …… 22

1 问题重述

1.1 问题背景

某 MOOC 在线教育平台希望通过实现个性化教学，让用户能够自主学习。在学习过程中，该平台从题库中随机抽取与例题同步的随堂测试题，并记录和分析学生的学习和答题信息。此外，该平台会定期回溯学生易错题所涉及的内容，自动推荐难度适中、题型相似的其他题目，供用户进行拓展练习。为了实现这些功能，该平台需要解决以下关键问题：如何度量题目之间的相似性，以及如何评估题目的难度。通过解决这些问题，该平台可以更好地满足用户的需求，提高用户的学习效果。

现有的度量题目相似性的方法包括题干文字和知识点标注，但它们的效果受到限制。题干文字的相似性可能无法捕捉到题意的差异，而知识点标注的方法则需要考虑知识点的粒度和划分。因此，需要采用更精确的方法来度量题目之间的相似性。评估题目难度的方法包括考试类型和教师主观判断，但它们都具有主观性和不确定性。因此，需要开发客观、可靠的方法来评估题目的难度。解决这些问题有助于提高在线教育平台的个性化教学效果和用户学习体验，帮助用户更好地自主学习，提高学习成效。

1.2 问题要求

根据以上背景信息，建立数学模型对以下具体问题进行讨论：

1. 设计刻画两道小学数学应用题之间相似性的度量方法。
2. 建立评估小学数学应用题难度的数学模型。
3. 将附件 1 中的题目，按相似性或难度分类（不限制某一道题目只能属于一个分类）。并评估算法的复杂度，能否适用于更大规模的题库。
4. 分析附件 2 中题目的难度，对于其中的每一道题目，在附件 1 中找出最相似的一道或若干道题目（没有相似题目写“无”）。并评估算法的复杂度，能否适用于更大规模的题库。

2 问题分析

对于问题一，在构建相似性度量方法时，我们的目的是为了能够精准地推荐给用户相关的练习题，提高其数学能力。为此，我们综合了多个有关权威网站[1] 的题型解析与相关文献[2] 的总结，确定了以下几个角度进行相似性评估，并对其进行了数据量化，以实现相似性的精确度量：

1. 应用题题型：小学数学应用题有多种不同的题型，例如工程问题，牛吃草问题等等。每种题型都需要不同的思考方式和解决方法。我们将有关权威网站的题型分类纳入小学数学应用题之间相似性的度量要素中，进而不仅考虑到了知识点的相似性比较，还考虑到了题目思考方式，逻辑方法的相似性比较，具有很强的可信性。

2. 标准解答是否需要画图解题：在小学数学应用题中，有些问题需要通过绘制图形才能更好地理解和解决。而需要画图解决的问题与不需要画图解决的问题，有较大的技术差异性。即对于那些需要画图才能解决的问题，学生需要具备良好的空间想象和准确的绘图技能。故而通过将此纳入小学数学应用题之间相似性的度量要素中，能起到较强的度量辅助作用。

3. 文本的相似性：题目中的文字描述也是小学数学应用题之间相似性比较的重要要素之一。在结合了小学数学应用题的内在技术要素后，我们仍然需要将文本相似性纳入小学数学应用题之间相似性的度量要素中。因为小学数学应用题的题目背景大概率与题目的内在技术有强联系关系，如工程问题中的绝大部分问题的文本叙述重点会极大相同。这样能够更全面地考虑到题目之间的相似性，增强相似性评估的精确性。

4. 应用真实场景：小学数学应用题往往与现实场景有很强的关联性，因此考虑到题目与现实场景的相似性比较有助于更好地评估题目之间的相似性，通过类似真实场景的题目推荐，也能帮助小学生对该类问题进行理解的加强。因此，我们将真实场景相似性度量纳入了小学数学应用题之间相似性的度量要素中，加以综合考虑题目的知识点、解题方式、文本相似性以及与现实场景之间的关联性，从而提高题目相似性评估的准确性和可靠性。

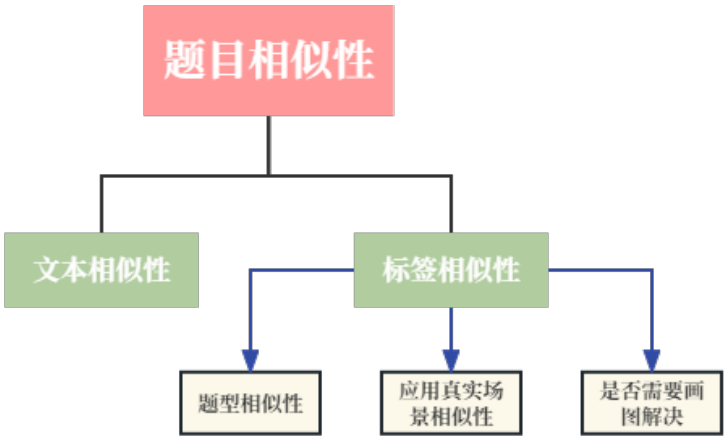

图 1: 题目相似性标签

对于问题二，针对问题二，需要建立评估小学数学应用题难度的数学模型。首先与问题一类似，我们通过结合小学生的学习实际与有关权威网站的数据，决定从以下角度进行小学数学应用题难度的评估，并进行数据量化，以实现难度的精确度量：

1. 题目长度：题目长度是一个影响小学数学应用题难度的重要因素。现在大多数学生对于应用题目的阅读通常是囫囵吞枣，当一遍阅读完成后即开始答题，最终所得到的答案大部分是错误的[3]。那么通常情况下，冗长和复杂的语句会造成学生的阅读困难，进而导致题目的难以理解，无法做出正确解答。我们通过计算小学数学应用题中的字符数来量化题目长度。

2. 未知量个数：未知量是数学应用题中需要被求解的数量，因此，未知量的个数直接影响着问题的难度，个数越多，意味着计算流程越复杂。在小学数学应用题中，未知量通常蕴含在问句中，即题目要求得到的结果。我们可以通过计算题目中未知量的个数来量化问题的难度。

3. 已知量个数：已知量是数学应用题中给定的已知条件或信息，是解题的基础。已知量的数量也会影响到问题的难度，个数阅越多，意味着题目所给条件越复杂，需要较强的归纳与理解能力。在小学数学应用题中，我们定义一条已知信息为一个已知量。我们可以通过计算题目中已知量的个数来量化问题的难度。

4. 标准解答是否需要画图解题：在小学数学应用题中，有些问题需要通过绘制图形才能更好地理解和解决。对于那些需要画图才能解决的问题，学生需要具备良好的空间想象和准确的绘图技能。因此，需要画图解题的问题通常比较困难。我们可以通过判断题目是否需要画图来量化问题的难度。

5. 涉及几位数的运算：在小学数学应用题中，涉及加、减、乘、除四则运算，其中四则运算涉及的位数也会影响问题的难度。例如，涉及两位数的乘法比涉及一位数的乘法要难得多。我们可以通过计算题目中涉及运算的最大位数来量化问题的难度。

6. 加减号个数：加减号的数量是影响小学数学应用题难度的因素之一。加减法是小学数学应用题中最基础的运算，加减号的数量越多，说明计算步骤越多，难度也就越大。因此，我们可以通过计算题目中加减号的数量来量化问题的难度。

7. 乘号个数：乘号的数量是影响小学数学应用题难度的因素之一。乘法是小学数学应用题中一个相对较难的运算，乘号的数量越多，说明计算步骤越多，难度也就越大。因此，我们可以通过计算题目中乘号的数量来量化问题的难度。

8. 除号个数：除号的数量是影响小学数学应用题难度的因素之一。除法是小学数学应用题中最难的四则运算，由于除法的运算方式与加减乘有很大差异. 除号的数量越多，说明计算步骤越多，难度也就越大。因此，我们可以通过计算题目中除号的数量来量化问题的难度。

9. 原子公式个数：基于小学数学应用题的逻辑步骤难以量化，我们通过收集官方解答步骤的数据，从中提取原子公式的个数以反映逻辑步骤，同时也反映了计算的复杂程度。原子公式是指数学应用题中的最基本的、不可分解的数学表达式，例如 $2x+3$、$4y-5$ 等。原子公式的数量也会影响问题的难度。我们可以通过计算题目中原子公式的数量来量化问题的难度。

10. 方程元数：方程元数指的是方程中未知数的数量。一个方程中未知数越多，就需要进行更多的计算才能得到正确答案，因此难度也就越大。我们考虑到部分应用题并非必须采用方程的方式解决，但方程的解题方式一般意味着原子公式个数的减少，即推理过程的难度下降。故而考虑方程元数这一因素可与原子公式个数相互协调，使得度量可信度得到提高。我们可以通过计算题目中未知数的数量来量化方程元数。

11. 标准解答是否涉及分数：小学数学中，涉及分数的题目往往会被认为是较难的题目之一，因为它需要学生有较强的分数概念和运算能力，以及抽象理解能力。并且约分，最小公分母的计算，会涉及到全部四则运算，因此涉及分数的应用题难度较大。我们可以通过检测应用题中是否涉及分数来量化问题的难度。

12. 应用题题型：小学数学应用题有多种不同的题型，例如工程问题，牛吃草问题等等。每种题型都需要不同的思考方式和解决方法，因此难度也会有所不同。我们可以根据应用题的题型进行分类，然后使用对各大权威网站的有关题型难度的数据统计与处理，对应用题难度进行量化评估。

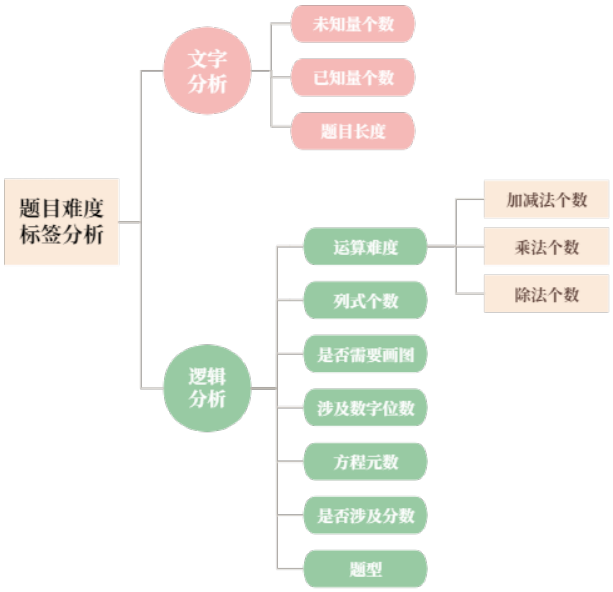

图 2: 题目难度评估标签

对于问题三，我们针对相似度与难度两种分类情况，基于问题一与问题二所得到的两种不同结果形式采取针对性的分类算法。对于按相似度分类，我们采用 AGNES 算法进行分类；对于按难度分类，我们采用 K-means 算法进行分类。同时我们进行两类算法的评估，并讨论其是否能适用于更大规模的数据集。

对于问题四，我们考虑将附件 2 中的 10 个应用题加入问题一的相似性度量模型，与附件 1 中的 100 个应用题共同进行相似性度量，对于输出的相似性度量结果再进行进一步的分析总结。

3 模型假设

1. 假设所有小学生的解题能力处于平均水平，即可以把所有小学生的数学能力视为同质的。
2. 假设题目的难度仅仅由题目本身的特征决定，而不受到小学生个体差异的影响。
3. 假设每个应用题目是独立的，即每个题目的难度与其他题目无关。

4. 假设不同小学生在理解题目和表达答案时使用的语言无差异性，而将其视为单纯的数学问题。

5. 假设不同小学生在解题时会按照标准的解题步骤进行操作，而不会使用其他不常见的解题方法，这些标准解题步骤是最容易被小学生理解和掌握的。

4 符号说明

表 1: 符号表

变量	说明
$feature$	各应用题特征词集合
$weight$	各应用题特征词权重集合
$a_{i,j}$	第 i 个应用题文本与第 j 个应用题文本对应的相似性
$b_{i,j}$	第 i 个应用题与第 j 个应用题题型对应的相似性
$c_{i,j}$	第 i 个应用题与第 j 个应用题标准解答对应的相似性
$d_{i,j}$	第 i 个应用题与第 j 个应用题所属真实场景对应的相似性
bw_i	第 i 个相似性度量矩阵对应的权重参数
$f_{i,j}$	第 i 个应用题与第 j 个应用题所属真实场景对应的相似性
$z_{i,j}$	第 i 个应用题的第 j 个难度评估属性值
$p_{i,j}$	第 i 个应用题的第 j 个难度评估属性值出现的频率
$e_{i,j}$	第 i 个应用题的第 j 个难度评估属性值对应的信息熵
d_j	第 j 个难度评估属性值的信息效用值
e_j	第 j 个难度评估属性所有应用题的信息熵之和
w_j	第 j 个难度评估属性对应的熵权
Z_j^+	第 j 个难度评估属性对应的正理想解
Z_j^-	第 j 个难度评估属性对应的负理想解
D_i^+	第 i 个应用题与正理想解的距离
D_i^-	第 i 个应用题与负理想解的距离
S_i	第 i 个应用题的难度评估分数
MS_i	与第 i 个应用题相似性最高的应用题

5 模型建立与求解

根据附件所提供的小学数学应用题，我们参考多个权威网站[1] 的相关解答，对每道题目的标准答案、所属题型、应用真实场景等所需要素进行了统计，以作为模型所需数据的有力参考，其中题型归类为：工程问题、和差问题、鸡兔同笼、逆推问题、年龄问题、牛吃草问题、浓度问题、容斥原理问题、相遇问题、其他问题；应用真实场景归类为：工程、化学、金融、军事、日常、物理、自然。部分展示如下：

序号	题目长度	未知量个数	已知量个数	标准解答是否需要画图解题	涉及几位数的运算	加减号个数	乘号个数	除号个数	原子公式个数	方程元数	是否涉及分数	题型	应用真实场景
1	67	2	2	1	2	2	1	3	4	1	1	盈亏问题	日常
2	67	1	3	2	2	1	2	1	4	1	1	追及问题	物理
3	135	1	3	2	2	4	4	4	4	1	2	相遇问题	物理
4	64	1	4	1	2	1	0	2	3	0	2	工程问题	工程
5	89	3	3	1	2	2	1	3	6	1	1	鸡兔同笼问题	军事
6	88	1	3	1	2	1	1	3	5	1	2	工程问题	工程
7	63	3	3	1	2	0	0	8	3	3	1	年龄问题	日常
8	105	3	2	1	1	1	4	2	4	3	1	牛吃草问题	日常
9	91	1	3	1	2	4	4	5	11	0	1	牛吃草问题	自然
10	30	2	1	1	2	0	0	1	1	1	1	年龄问题	日常
11	81	1	3	1	2	2	2	2	3	1	2	工程问题	工程
12	82	1	3	2	2	1	1	1	3	0	1	相遇问题	物理
13	64	1	3	2	1	0	1	1	2	0	1	追及问题	物理
14	117	1	3	2	2	0	2	2	6	1	2	追及问题	物理
15	55	1	4	2	2	0	0	3	2	0	1	容斥原理	日常
16	102	1	5	1	2	0	3	2	5	1	1	逆推问题	日常
17	53	1	3	1	3	0	2	2	3	0	1	逆推问题	日常
18	112	2	4	1	2	0	3	2	4	1	2	工程问题	工程
19	91	1	3	1	2	2	5	4	8	0	1	牛吃草问题	自然
20	68	1	3	2	3	1	0	1	2	0	1	追及问题	物理

图 3: 各属性标注部分展示

其中，特殊标注原则如下：

表 2: 特殊标注原则 1

题型	**对应的标号**
工程问题	1
和差问题	2
鸡兔同笼	3
逆推问题	4
年龄问题	5
牛吃草问题	6
浓度问题	7
容斥原理	8
相遇问题	9
盈亏问题	10
追及问题	11

表 3: 特殊标注原则 2

应用背景	**对应的标号**
工程	1
化学	2
金融	3
军事	4
日常	5
物理	6
自然	7

表 4: 特殊标注原则 3

画图解决/涉及分数	对应的标号
不需要/未涉及	1
需要/涉及	2

5.1 问题一模型的建立与求解

5.1.1 数据预处理

1. 文本相似性数据预处理

   针对各应用题的文本相似性衡量，我们需要采用更加复杂的文本预处理，以提取出应用题文本更加有效的特征。对于应用题文本相似性度量矩阵建模，我们采用 SimHash 算法用于判断两段文本的相似性，故而数据预处理阶段我们需要进行中文分词与提取特征词的工作

   (a) 中文分词

   中文分词是将一段中文文本按照词语划分成一个一个具有实际意义的词汇的过程。在自然语言处理中，分词是一项非常基础的任务，对于后续的文本处理和分析都有着重要的作用。中文分词算法有很多，其中 jieba 是一种比较流行的中文分词工具。基于小学数学应用题拥有相对具体的应用场景，故而中文分词的重要性可见一斑。

   (b) 提取特征词

   在 SimHash 算法中，对待处理文档进行中文分词后，得到有效的特征及其权重是非常关键的一步。在这一步骤中，我们可以使用 TF-IDF 方法获取一篇文章权重最高的前 topK 个词（$feature$）和权重（$weight$）。TF-IDF（Term Frequency-Inverse Document Frequency）是一种常用的文本相似性计算方法，它通过统计词频和逆文档频率来计算一个词语在整个文档集中的重要程相似性。在计算 TF-IDF 的过程中，词频（TF）指的是某个词在文章中出现的次数，逆文档频率（IDF）指的是包含该词的文档数与总文档数之间的比值的对数的负数。

   我们使用 jieba 中用于提取文章中关键词的函数，可以自动对文章中的词语进行权重排序，然后返回排名前 topK 的关键词及其权重。使用该函数，我们可以轻松地获取到一篇文章中的关键词及其权重，从而得到一组有效的特征。针对小学数学应用题的实际情况，我们选择的特征词数目为 4 个。这些特征可以作为后续 SimHash 算法的输入，用于计算文章的 SimHash 值，为应用题文本相似性度量矩阵建模做好准备。

2. 其他数据预处理

   根据附件 1 所提供的 100 道小学数学应用题，我们对每道题目的标准答案及其所属题型进行了统计，进而确定各应用题所属应用题题型，标准解答是否需要画图解答、应用题所属真实场景这三个角度信息的收集。在此基础上进行进行对应类别标签的标注。

5.1.2 应用题文本相似性度量矩阵建模

1. 对各应用题对应的特征词使用哈希操作
   对于数据预处理阶段得到的各应用题特征词集合 $feature$ 进行普通的哈希操作，计算 $hash$ 值，这样就得到一个长度为 n 位的二进制数，与数据预处理阶段得到的特征词集合对应权重集合 $weight$ 得到 $(hashweight)$ 的集合。只要有一个词，就能够生成长度为 64 的 hash 值。

2. 对哈希值进行加权
   根据对应的 weight 值进行加权，即 $W = hash * weight$。即 hash 为 1 则和 weight 正相乘，为 0 则和 weight 负相乘。例如一个词经过 hash 后得到 (0101115)，经过该步骤之后可以得到列表 $[-5, 5, -5, 5, 5, 5]$。

3. 加权结果的合并
   将上述得到的各个向量的加权结果进行求和,变成只有一个序列串。例如,将 $[-5, 5, -5, 5, 5, 5]$ $[-3, -3, -3, 3, -3, 3][1, -1, -1, 1, 1, 1]$ 进行列向累加得到 $[-7, 1, -9, 9, 3, 9]$。

4. 合并结果降维
   对于上述步骤得到的 n 位向量的累加结果的每个值进行判断，大于 0 则置为 1，否则置为 0，从而得到该语句的 simhash 值。例如，$[-7, 1, -9, 9, 3, 9]$ 得到 010111，这样，我们就得到一个文档的 simhash 值，作为衡量文本相似性的一个关键参考指标。

5. 计算各应用题 simhash 值的汉明距离
   在 simHash 算法中，通过将文本转换为 simHash 值，并计算汉明距离来判断文本的相似性。汉明距离是指两个等长字符串对应位置上不同字符的个数，例如字符串"1011101" 和"1001001" 的汉明距离为 2，因为它们在第 2 和第 5 个位置上的字符不同。在 simHash 算法中，假设我们将文本转换为长度为 n 的二进制字符串，那么它们的汉明距离就是这两个二进制字符串不同位的数量。

   因为 simHash 值的长度是固定的 n 位，所以我们可以通过 1 减去汉明距离与 n-bit 的比值来计算相似性。例如，如果两个文本的 simHash 值的汉明距离为 d，那么它们的相似性就可以用以下公式来表示：

$$similarity = 1 - \frac{d}{n} \tag{1}$$

   其中，n 表示 simHash 值的长度。

6. 应用题文本相似性度量矩阵搭建
   根据各应用题之间的文本相似性，统计为如下的应用题文本相似性度量矩阵 A，以参与应用题相似性最终矩阵的建模，假设有 m 个应用题，其中 $a_{i,j}$ 代表第 i 个应用题文本与第 j 个应用题文本对应的相似性，为了减少数据计算量，我们只需要计算相似性矩阵的右上半

部分:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ & a_{2,2} & \cdots & a_{2,m} \\ & & \ddots & \vdots \\ & & & a_{m,m} \end{bmatrix} \tag{2}$$

5.1.3 应用题相似性最终矩阵建模

各应用题所属应用题题型，标准解答是否需要画图解答、应用题所属真实场景对应的相似性度量矩阵搭建较为简单，即将两道应用题的这些要素进行比较后，若相同则在对应的相似性度量矩阵位置填入 2，否则填入 1，即搭建了另外三个要素的相似性度量矩阵，分别为 B，C，D。

1. 各相似性度量矩阵归一化

   由于四个衡量要素的相似性分析方法不一，为了正确有效地进行最终相似性的评估，我们要对四个要素的相似性度量矩阵进行归一化处理，以便后续的权重求和得出最终结果。

   标准化过程我们采用的是向量归一化（Vector Normalization）方法。将相似性评估属性矩阵中的每一个元素除以其所在列的元素平方和的平方根，就是将该元素所在的列向量进行向量归一化的结果。这种方法的意义在于将不同列之间的元素量纲统一，从而可以消除属性之间的度量单位和大小的差异。以下是以应用题文本相似性度量矩阵 A 为例的示意:

$$a_{i,j} = \frac{a_{i,j}}{\sqrt{\sum_{k=1}^{m} a_{k,j}^2}} \tag{3}$$

2. 根据预设好的超参数权重构建应用题相似性最终矩阵

   在应用题相似性度量矩阵的构建中，超参数权重通常指的是对应于不同特征的权重系数。我们根据小学数学应用题的实际背景，总结为对于相似性的判断中，应用题的题型与文本相似性应占比较大，标准解答是否需要画图解题与应用题所属真实场景占比次之。经由此原则，我们进行权重参数的初步设定。经过多轮调试运行评估应用题之间的相似性，与有关权威网站的相似题推荐数据相互比较，最后得到相对最优的超参数权重设置，对应四个相似性度量矩阵 A，B，C，D 为 bw_1，bw_2，bw_3，bw_4。

   最终，经由权重处理，我们得到应用题相似性最终矩阵 F，其中 $f_{i,j}$ 代表第 i 个应用题与第 j 个应用题之间的相似性，$f_{i,j} = a_{i,j} + b_{i,j} + c_{i,j} + d_{i,j}$，应用题相似性评估部分展示如下:

表 5: 应用题相似性评估部分表

题目标号	1	2	3	4	5	6	7
1		**1.198393**	**1.172352**	**1.345488**	**1.267363**	**1.293404**	**1.32297**
2			**1.192762**	**1.120268**	**1.250477**	**1.224435**	**1.354643**
3				**1.094227**	**1.328602**	**0.99006**	**1.068185**
4					**1.189238**	**1.459287**	**1.293404**
5						**1.241321**	**1.267363**
6							**1.397571**
7							

5.2 问题二模型的建立与求解

5.2.1 数据预处理

根据附件 1 所提供的 100 道小学数学应用题，我们对每道题目的标准答案及其所属题型进行了统计。在此基础上，我们进行了各应用题对应的难度评估属性的收集整理，搭建了难度评估属性矩阵 Z。其中对于各题型对应的难度系数统计，我们参考了多个权威小学数学应用题统计网站，将不同题型的题目按照简单、中等、困难三个层次进行分类统计，以此进行题型对应难度系数的量化统计。本次数据收集对数据统计和分析方法进行了科学、系统的规划和设计，旨在提高研究的可信性和可靠性。

$$Z = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,n} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m,1} & z_{m,2} & \cdots & z_{m,n} \end{bmatrix} \tag{4}$$

为了实现后续的多属性决策类模型建立，需要将每个备选方案的指标进行正向化与标准化处理。在数据生成初期，我们已规定每个评估属性对应的量化值与对应题目的难度大小成正比，故正向化处理过程无需进行。

标准化过程我们采用的是向量归一化（Vector Normalization）方法。将难度评估属性矩阵中的每一个元素除以其所在列的元素平方和的平方根，就是将该元素所在的列向量进行向量归一化的结果。这种方法的意义在于将不同列之间的元素量纲统一，从而可以消除属性之间的度量单位和大小的差异，使得它们在后续的多属性决策分析中具有可比性。这是我们在后续建模中使用多属性决策分析方法时非常重要的一个前提。

同时，这样做的好处在于，标准化后的数据可以更好地反映不同属性在整个数据集中的分布情况，从而更加准确地确定属性权重，提高多属性决策的鲁棒性和可靠性。

$$z_{i,j} = \frac{z_{i,j}}{\sqrt{\sum_{k=1}^{m} z_{k,j}^2}} \tag{5}$$

此时得到的难度评估属性矩阵中不存在负数，无需进行下一步标准化处理转换负数数据，所得的难度评估属性矩阵即为数据预处理后的结果。

5.2.2 难度评估属性权重量化建模

通过问题解读与结合多属性决策相关经典文献知识，我们可以构建难度评估属性权重量化模型，该模型利用信息熵来计算每个难度评估属性的权重，即使用熵权法，以反映难度评估属性对各应用题难度评估的贡献相似性。以下是每个难度评估属性权重量化的详细步骤：

1. 计算各个难度评估属性值的概率分布，即每个难度评估属性值出现的频率
   假设有 m 个应用题，n 个难度评估属性。对于第 i 个应用题，第 j 个难度评估属性的属性值 $z_{i,j}$ 出现的频率 p_{ij} 可以计算如下：

$$p_{ij} = \frac{z_{i,j}}{\sum_{k=1}^{m} z_{i,j}} \tag{6}$$

2. 计算各个难度评估属性值对应的信息熵
   对于第 j 个难度评估属性，第 i 个应用题的难度评估属性值 $z_{i,j}$ 对应的信息熵 $e_{i,j}$ 可以计算如下：

$$e_{i,j} = -\frac{1}{\ln m} \sum_{i=1}^{m} p_{i,j} \ln(p_{i,j}) \tag{7}$$

   这个公式是信息熵的标准公式，用于表示随机变量的不确定性或信息量。

3. 计算各个难度评估属性值对应的信息效用值
   为了消除不同难度评估属性的量纲和取值范围的影响，可以对每个难度评估属性的信息熵进行归一化处理，计算出每个难度评估属性值的信息效用值 d_j，公式如下：

$$d_j = 1 - e_j \tag{8}$$

   其中，e_j 表示第 j 个难度评估属性的所有应用题的信息熵之和。

4. 计算各个难度评估属性对应的熵权
   最后，可以通过计算各个属性的熵权 w_j 来确定各个属性的权重，公式如下：

$$w_j = \frac{d_j}{\sum_{j=1}^{n} d_j} \tag{9}$$

   这个公式就是熵权法的核心公式，它根据信息熵的大小来确定各难度评估属性的权重。它根据信息熵的大小来确定各难度评估属性的权重。最终得到的 w_j 可以表示第 j 个难度评估属性对于整个应用题难度评估的重要程相似性。

   部分应用题难度评估后数据如图所示：

题目	属性1	属性2	属性3	属性4	属性5	属性6	属性7	属性8	属性9	属性10	属性11
1	0.580780381	0.084280864	0.119309997	0.053319117	0.077849894	0.088216218	0.103301186	0.105703285	0.078326045	0.079056942	0.179085409
2	0.739644289	0.084280864	0.059654999	0.079978675	0.155699789	0.088216218	0.075128135	0.105703285	0.078326045	0.079056942	0.176793283
3	0.562370628	0.16981965	0.059654999	0.079978675	0.155699789	0.088216218	0.225384406	0.105703285	0.078326045	0.158113883	0.38972912
4	0.556149614	0.080507094	0.059654999	0.106638234	0.077849894	0.088216218	0.046955085	0.079277463	0	0.158113883	0.034163973
5	0.697490113	0.111955177	0.178964996	0.079978675	0.077849894	0.088216218	0.103301186	0.158554927	0.078326045	0.079056942	0.267787408
6	0.556149614	0.110697254	0.059654999	0.079978675	0.077849894	0.088216218	0.075128135	0.132129106	0.078326045	0.158113883	0.180540795
7	0.634125762	0.07924917	0.178964996	0.079978675	0.077849894	0.088216218	0.075128135	0.079277463	0.234978135	0.079056942	0.630321542
8	0.796099068	0.13208195	0.178964996	0.053319117	0.077849894	0.044108109	0.12208322	0.105703285	0.234978135	0.079056942	0.707332262
9	0.796099068	0.114471024	0.059654999	0.079978675	0.077849894	0.088216218	0.234775423	0.290684032	0	0.079056942	0.35275206
10	0.634125762	0.0377377	0.119309997	0.026659558	0.077849894	0.088216218	0.009391017	0.026425821	0.078326045	0.079056942	0.084453731
11	0.556149614	0.10189179	0.059654999	0.079978675	0.077849894	0.088216218	0.112692203	0.079277463	0.078326045	0.158113883	0.175747968
12	0.562370628	0.103149714	0.059654999	0.079978675	0.155699789	0.088216218	0.056346101	0.079277463	0	0.079056942	0.048759766
13	0.739644289	0.080507094	0.059654999	0.079978675	0.155699789	0.044108109	0.028173051	0.052851642	0	0.079056942	0.035512037
14	0.739644289	0.14717703	0.059654999	0.079978675	0.155699789	0.088216218	0.056346101	0.158554927	0.078326045	0.158113883	0.228346902
15	0.559346591	0.069185783	0.059654999	0.106638234	0.155699789	0.088216218	0.028173051	0.052851642	0	0.079056942	0.035763235
16	0.586425261	0.12830818	0.059654999	0.133297792	0.077849894	0.088216218	0.075128135	0.132129106	0.078326045	0.079056942	0.195262076
17	0.586425261	0.066669937	0.059654999	0.079978675	0.077849894	0.132324327	0.056346101	0.079277463	0	0.079056942	0.029115055
18	0.556149614	0.140887414	0.119309997	0.106638234	0.077849894	0.088216218	0.075128135	0.105703285	0.078326045	0.158113883	0.188155074
19	0.796099068	0.114471024	0.059654999	0.079978675	0.077849894	0.088216218	0.187820338	0.211406569	0	0.079056942	0.229324221
20	0.739644289	0.085538787	0.059654999	0.079978675	0.155699789	0.132324327	0.037564068	0.052851642	0	0.079056942	0.04170794

图 4: 部分应用题难度评估后数据

5.2.3 应用题难度评估建模

在计算完各个难度评估属性的熵权值后，我们使用 TOPSIS 方法，计算个应用题具有的各难度评估属性的与正负理想解的距离，以评估各应用题的难度，以下是应用题难度评估的详细步骤：

1. 计算正负理想解

   正理想解是指各属性中导致应用题难度最大的取值组合，意味着难度最大的应用题，而各属性对应的负理想解是指各属性中导致应用题难度最小的取值，意味着难度最小的应用题。

   我们使用数据预处理步骤中得到的标准化难度评估属性矩阵，得到每个难度评估属性对应的正负理想解。

   正理想解：

   $$Z_j^+ = \max_{i=1}^{m} Z_{i,j} \tag{10}$$

   负理想解：

   $$Z_j^- = \min_{i=1}^{m} Z_{i,j} \tag{11}$$

2. 计算各个应用题到正理想解和负理想解的相对距离矩阵

   我们使用使用欧氏距离距离度量方法，计算各个应用题到正理想解和负理想解的距离，分别得到两个相对距离矩阵 D^+ 和 D^-。

   对于第 $i(i = 1, 2, 3, ..., n)$ 个应用题与正理想解的距离：

   $$D_i^+ = \sqrt{\sum_{j=1}^{n} w_j (Z_j^+ - z_{i,j})^2} \tag{12}$$

   对于第 $i(i = 1, 2, 3, ..., n)$ 个应用题与负理想解的距离：

   $$D_i^- = \sqrt{\sum_{j=1}^{n} w_j (Z_j^- - z_{i,j})^2} \tag{13}$$

3. 计算各应用题的难度评估分数

至此，我们可以得出第 i 个应用题的难度评估分数 (位于 $[0,1]$ 之间)：

$$S_i = \frac{D_i^-}{D_i^- + D_i^+} \tag{14}$$

题目	难度评分
1	0.707332262
2	0.684846684
3	0.666801495
4	0.630321542
5	0.619588232
6	0.617906474
7	0.562392207
8	0.46824288
9	0.464352825
10	0.455058971
11	0.451853672
12	0.422782903
13	0.41972179
14	0.402686816
15	0.38972912
16	0.389519837
17	0.387504242
18	0.377838561
19	0.375546994
20	0.371893428

图 5: 部分应用题难度评估分数

5.3 问题三模型的建立与求解

5.3.1 按相似度分类

基于问题一所得的应用题相似性度量矩阵，我们采用聚类方法中的 AGNES 算法进行分类，用于将各个应用题对应的数据点分成不同的群组，其中具有较高相似性的应用题被归为一组。其中 AGNES 算法的基本思想是将每个数据点视为一个单独的簇，然后逐步合并具有最小距离的簇，直到所有数据点都被合并到一个簇中。这个过程形成了一个树形结构。

在 AGNES 算法中，距离度量是非常重要的，因为它影响着聚类的结果。我们在运用此算法的过程中，采用的是问题一所得的相似性度量矩阵，对应各数据点之间的距离。

如果簇 C1 中的一个对象和簇 C2 中的一个对象之间的距离是所有属于不同簇的对象间欧式距离中最小的，C1 和 C2 可能被合并。这是一种单连接方法，其每个簇可以被簇中的所有对象代表，两个簇之间的相似性由这两个簇中距离最近的数据点对的相似性来确定。

算法具体流程如下：

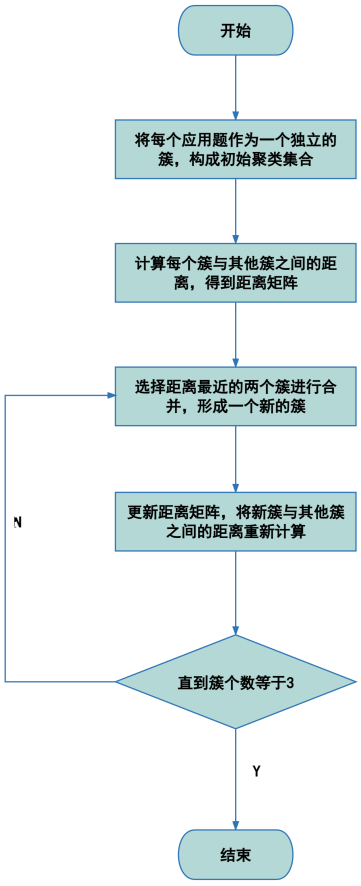

图 6: AGNES 分类流程图

分类后的结果如图：

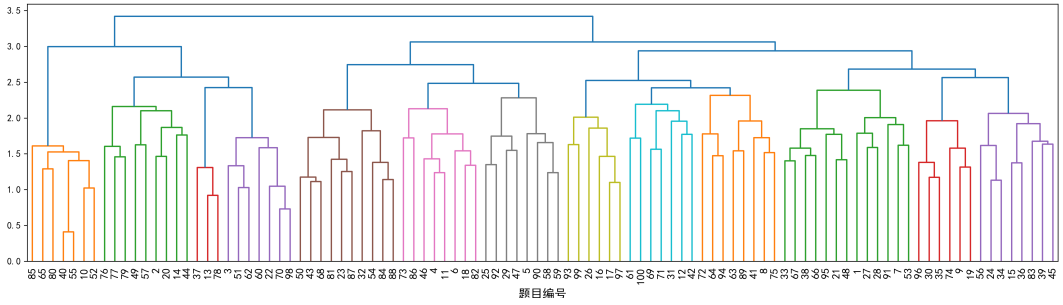

图 7: 按相似性分类结果树状图

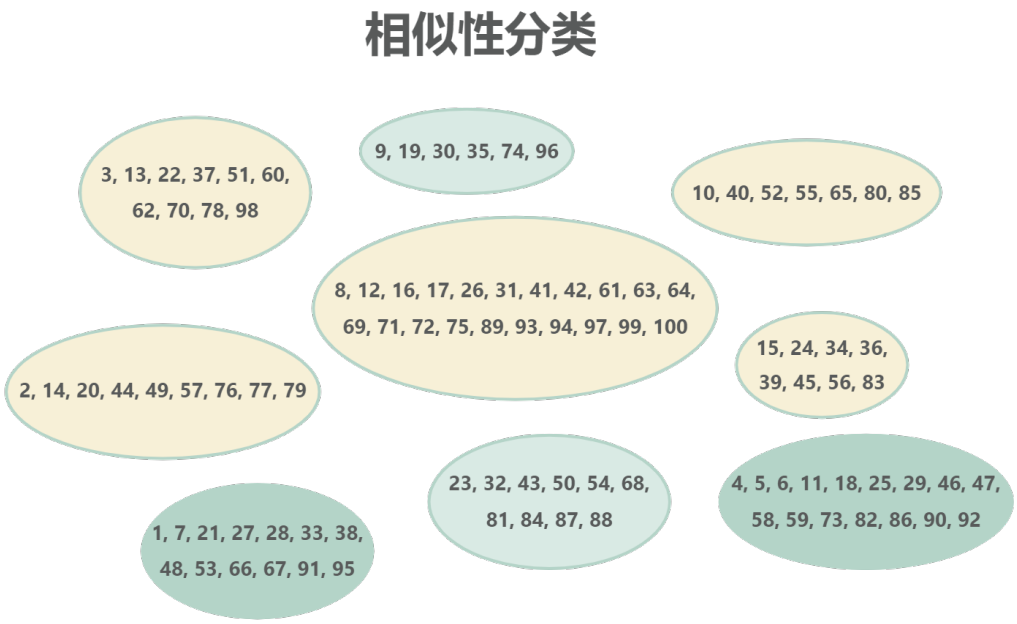

图 8: 按相似性分类结果图

5.3.2 按难度分类

基于问题二所得的应用题难度评估列表，我们采用聚类方法中的 K-means 算法进行分类，其目标是将数据集划分为 k 个簇（k 为预设的簇数），每个簇具有相似的特征。该算法通过迭代来优化每个簇的质心，使得簇内数据点的距离之和最小化。我们选择 $k = 3$。

算法具体流程如下：

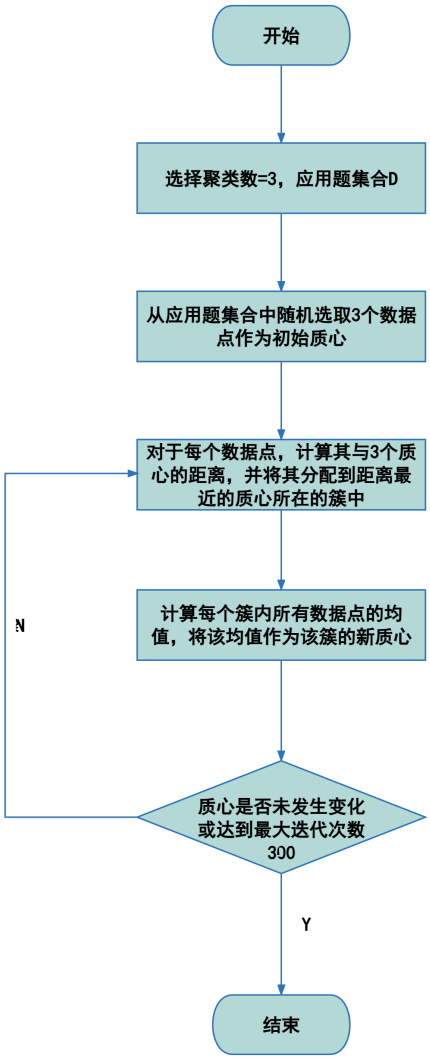

图 9: K-means 分类流程图

分类后的结果如图：

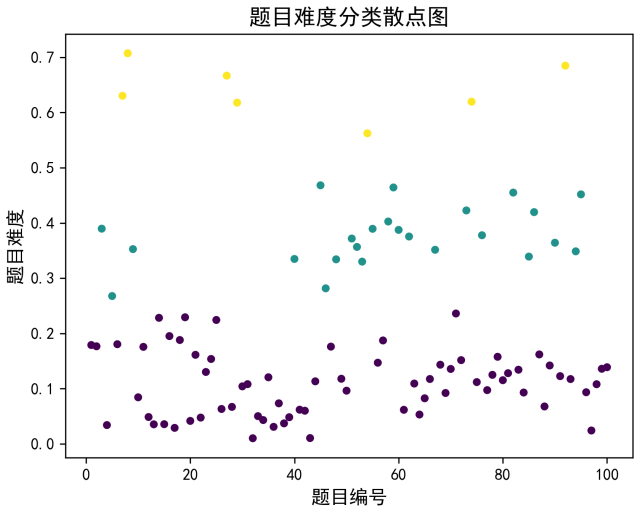

图 10: 按难度分类结果图

5.3.3 算法评估

AGNES (Agglomerative Nesting) 和 K-means 聚类算法是常用的聚类算法之一。AGNES 算法是一种层次聚类算法，而 K-means 算法是一种迭代聚类算法。两种算法在聚类任务中有不同的优点和缺点，并且在处理大规模数据集时的可扩展性表现也有所不同。

AGNES 算法采用自下而上的策略，从每个数据点作为单独的簇开始，逐步将相似的簇合并为更大的簇，直到所有数据点都被合并为一个簇。AGNES 算法的优点在于可以处理非凸形状的簇，并且可以提供聚类结果的层次结构。然而，其时间复杂度较高，尤其是在处理大规模数据集时，计算相似性矩阵的时间会非常长。

相比之下，K-means 算法可以在大规模数据集上高效地处理聚类任务，因为它采用迭代方式逐步优化簇的质心，并且可以使用分布式计算框架进行并行计算。K-means 算法的缺点在于需要事先指定簇的个数，并且对于非凸形状的簇表现不佳。此外，K-means 算法对初始质心的选择非常敏感，需要多次运行算法以选择最优结果。

在处理大规模数据集时，K-means 算法通常比 AGNES 算法更为适用，因为它具有较好的可扩展性。但是，如果数据集具有非凸形状的簇或需要得到聚类结果的层次结构，则 AGNES 算法可能更适合。此外，对于需要对簇的解释和可视化的应用，AGNES 算法提供了更好的结果，因为它能够提供聚类结果的层次结构。

综上所述，AGNES 算法和 K-means 算法在聚类任务中具有各自的优缺点和适用场景。需要根据数据集的特点和需求进行选择。在处理大规模数据集时，K-means 算法通常更为适用，但在处理具有非凸形状的簇或需要得到聚类结果的层次结构时，AGNES 算法更合适。同时，选择算法时应该考虑算法的时间复杂度和可扩展性，并进行算法参数的调整和多次运行，以获得更好的聚类结果。

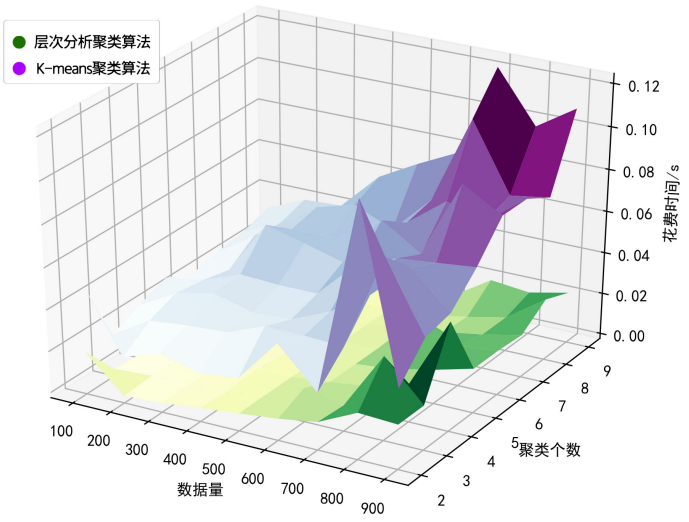

图 11: 分类算法复杂度分析

5.4 问题四模型的建立与求解

5.4.1 求解过程

我们考虑将附件 2 中的 10 个应用题加入问题一的相似性度量模型，与附件 1 中的 100 个应用题共同进行相似性度量。即使问题一的应用题相似性矩阵得到扩充，对于处理后的最终矩阵，我们重点提取观察附件 2 中的 10 个应用题。如对应的应用题编号为 i，共有 n 个应用题，我们需要在矩阵中提取 (i,j), 其中 $j=i,i+1...n$ 与 (j,i),$j=1,2...i-1$ 对应的相似性数值。对于提取出的相似性数值进行排序，最大值即为与该题最相似的应用题。输出结果如下，表格中越靠左的应用题意味着与对应附件 2 中的应用题越相似, 即为与第 i 题最相似的题 MS_i:

表 6: 附件 2 题目相似性结果

应用题	**附件 1 应用题题号)**								
Q001	**3**	12	64	57	60	76	14	20	49	98
Q002	**39**	31	1	27	66	56				
Q003	**86**	96	73	40	55	37				
Q004	**23**	84	88	70	71	68				
Q005	**11**	30	6	18	35					
Q006	**51**	16	3							
Q007	**80**	40	10	55	65	52	53			
Q008	**64**	59	25	27	72	92				
Q009	**6**	82	28	11	30	96				
Q010	**37**	78	13	77	79	14	49	20		

其中我们会基本忽略相似性在 1.3 以下的选择，视作该题无相似题目。我们考虑将附件 2 中的 10 个应用题加入问题二的难度度量模型，与附件 1 中的 100 个应用题共同进行难度度量。输

出结果即可得到这 10 个应用题的难度得分。输出结果如下：

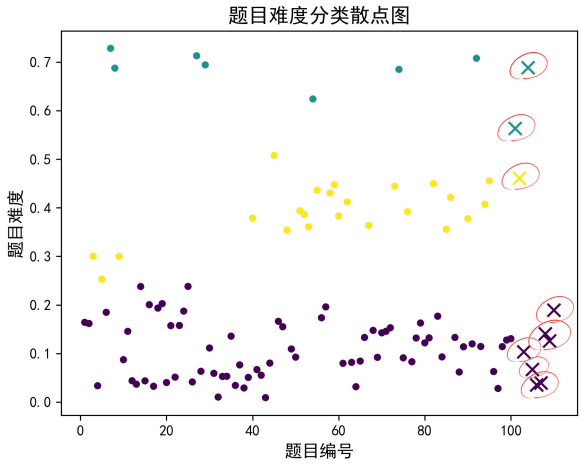

图 12: 附件 2 题目相似性难度评估结果

5.4.2 评估算法

我们使用的相似性度量算法重点分为数据预处理，各属性相似性度量矩阵的搭建，最终的矩阵搭建三步，由于矩阵的搭建需要的时间复杂度与空间复杂度均为 $O(n^2)$，那么针对更大的题库，会导致算法运行时间过长，比较出最相似应用题的效果不佳。

基于熵权法与 TOPSIS 方法的底层原理，多属性评估需要一定量数据支撑，故而无法进行进一步的剪枝优化。

6 模型评价和推广

6.1 模型的评价

6.1.1 模型的优点

1. 本模型从小学数学学习现状入手，确定了小学数学应用题相似性度量的多方面因素，为实际应用中相似题目的推荐提供了重要的指导。

2. 采用多角度量化的小学数学应用题相似性度量，使得原本抽象的各方面数据得到量化，并重点引入了 simhash 算法，进行归一化与加权操作，有效地衡量了小学数学应用题的相似性。

3. 本模型基于熵权法与 TOPSIS 方法进行小学数学应用题难度评估，通过综合评价多方面影响题目难度的因素，从原始数据入手，依据信息熵对属性权重进行量化，无需主观评分，具有应用题难度评估的绝对客观性。这种方法的优点在于它是基于数据本身进行分析，不受主观因素影响，因此结果更加客观准确。

4. 本模型引入了 AGNES 与 K-means 算法进行分类，有利于结合应用场景，发挥对应算法的优势，以应用于多种分类场景。这样的方法可以使得分类更加准确，能够根据实际需求灵活地调整分类方式，提高模型的实用性和适用性。

6.1.2 模型的缺点

1. 采用多角度量化的小学数学应用题相似性度量，各要素的权重参数仍待优化，需要依据更加丰富的实际数据来进行权重调整。

2. 虽然基于熵权法与 TOPSIS 算法的难度评估能够量化多方面因素并降低主观评分的影响，但其准确性仍然受到数据采集和处理的影响，需要保证数据的准确性和完整性，否则评估结果可能会出现偏差。

3. AGNES 算法在处理大规模数据集时会面临较大的计算复杂度和存储空间问题；K-means 算法的分类效果受超参数设置的影响较大，如聚类数量等。

6.2 模型的推广

本文主要设计了多角度量化的小学数学应用题相似性度量模型，基于熵权法与 TOPSIS 方法的小学数学应用题难度评估模型，在此基础上衍生出对应的 AGNES 与 K-means 分类模型。在教育领域，可以应用于小学数学应用题的智能评估和分类，帮助教师更好地了解学生的学习情况和进度，为个性化教学提供支持。在人工智能领域，可以应用于 MOOC 平台的智能推荐系统，为学生提供个性化的学习内容和适当难度的题目。在企业管理领域，可以用于企业招聘和人才选拔中，帮助企业更加准确地评估应聘者的数学能力，提高招聘效率。此外，本文设计的模型还可以应用于各类数学竞赛和考试中，提高题目的难度分级和分类，为参赛者提供更加公平的竞争机会。综上所述，本文设计的模型具有广泛的实际应用价值，将为相关领域的教育、人才选拔、智能推荐和竞赛评估等方面提供有效的支持和参考。

7 参考文献

参考文献

[1] 作业帮. 作业帮小学数学试卷库. https://www.zujuan.com/question?tree_type=knowledge&xd=1&chid=3. Accessed: 2023-05-04.

[2] 王莉. 小学数学应用题解析与拓展. *数学学习与研究*, 22(137–139), 2023.

[3] 洪敏. 小学数学应用题教学的现状及解题策略. In *社会发展——跨越时空经济基础论文集（一）*, pages 2797–2800, 2023.

8 附录

8.1 支撑材料列表

以下是本文模型搭建涉及的全部代码与数据支撑材料列表:

表 7: 支撑材料列表

文件名	作用描述
标签标注	标签原始数据
standard.py	对难度标签数据进行标准化
standard.xls	standard.py 的结果文件
题型难度分值计算.xlsx	计算不同题型的难度评分
cal_weight.py	利用熵权法计算不同标签的权重
weight.xls	cal_weight.py 的结果文件
cal_difficult_score.py	利用 TOPSIS 算法计算所有题目的难度评分
score.xls	cal_difficult_score.py 的结果文件
110score.xls	修改一些的 cal_difficult_score.py 的结果文件
前 100 道题总难度评分.xls	整合 standard.xls 、题型难度分值计算.xlsx 、score.xls 的数据
kmeans.py	利用 Kmeans 算法对前 100/110 题进行分类
time.py	计算 Kmeans 和 AGNES 的时间复杂度
110 题标准化数据	standard.py 的结果文件和题型难度分值结算.xlsx 的结合
cluster.py	利用层次聚类算法对 100 道题的相似性进行分类
initz.xlsx	100 道题的原始数据
110 道题原始数据.xlsx	对 110 道题的原始数据的四则符号数据进行处理
text.py	使用 SimHash 算法计算两个题目之间的文本相似性
tag.py	自动标注每两道题之间的标签相似性
tag_stand.py	将标签相似性标准化
similar.py	将文本相似性和标签相似性赋予权重并统计在一起，计算总的题目
result.xls	text.py 生成的文本相似性矩阵
result2.xls	tag.py 生成的未经处理的标签相似性
result3.xls	tag_stand.py 生成的标准化数据后的标签相似性
similarity.xls	similar.py 生成的题目相似性矩阵
textsimilar.xlsx	100 道题目列表，用于 text.py 文件文本相似性的计算
tagsimilar.xlsx	100 道题目列表以及其所对应的题型、应用背景和画图需求性，以计算标签相似性
stopwords.txt	自然语言处理所用的停用词表
search.py	输入需要计算相似性的题目标号，并计算其与 100 道题目的相似性
tagsimilar.xlsx	100 道题目列表以及其所对应的题型、应用背景和画图需求性，用于计算标签相似性
question.xlsx	10 道需要计算相似性的题目
stopwords.txt	自然语言处理所需要的停用词表

表 8: 续表: 支撑材料列表

文件名	作用描述
Q001.xls	计算出的题目 1 与其它 100 道题目的相似性数值
Q002.xls	计算出的题目 2 与其它 100 道题目的相似性数值
Q003.xls	计算出的题目 3 与其它 100 道题目的相似性数值
Q004.xls	计算出的题目 4 与其它 100 道题目的相似性数值
Q005.xls	计算出的题目 5 与其它 100 道题目的相似性数值
Q006.xls	计算出的题目 6 与其它 100 道题目的相似性数值
Q007.xls	计算出的题目 7 与其它 100 道题目的相似性数值
Q008.xls	计算出的题目 8 与其它 100 道题目的相似性数值
Q009.xls	计算出的题目 9 与其它 100 道题目的相似性数值
Q010.xls	计算出的题目 10 与其它 100 道题目的相似性数值
B_tex.zip	论文原始 tex 文件

8.2 建模代码

文件名: search.py

```python
# -*- coding:utf-8 -*-
import jieba
import jieba.analyse
import numpy
import numpy as np
import pandas as pd
import openpyxl as op
import xlwt
import math


class SimHash(object):
    def simHash(self, content):
        seg = jieba.cut(content)
        jieba.analyse.set_stop_words('stopwords.txt')
        keyWords = jieba.analyse.extract_tags("|".join(seg), topK=10,
    withWeight=True)
        # print(keyWords)
        keyList = []
        for feature, weight in keyWords:
            # weight = math.ceil(weight)
            weight = int(weight * 10) + 1
            # print('weight: {}'.format(weight))
            binstr = self.string_hash(feature)
```

```
            temp = []
            for c in binstr:
                if (c == '1'):
                    temp.append(weight)
                else:
                    temp.append(-weight)
            keyList.append(temp)
        listSum = np.sum(np.array(keyList), axis=0)
        # print(listSum)
        if (keyList == []):
            return '00'
        simhash = ''
        for i in listSum:
            if (i > 0):
                simhash = simhash + '1'
            else:
                simhash = simhash + '0'

        return simhash

    def string_hash(self, source):
        if source == "":
            return 0
        else:
            x = ord(source[0]) << 7
            m = 1000003
            mask = 2 ** 128 - 1
            for c in source:
                x = ((x * m) ^ ord(c)) & mask
            x ^= len(source)
            if x == -1:
                x = -2
            x = bin(x).replace('0b', '').zfill(64)[-64:]
            # print('strint_hash: %s, %s' % (source, x))

            return str(x)

    def getDistance(self, hashstr1, hashstr2):
        '''
            计算两个simhash的汉明距离
        '''
```

```
        length = 0
        for index, char in enumerate(hashstr1):
            if char == hashstr2[index]:
                continue
            else:
                length += 1

        return length


if __name__ == '__main__':
    simhash = SimHash()
    file = pd.read_excel("tagsimilar.xlsx")
    questions = pd.read_excel("question.xlsx")
    mid5 = xlwt.Workbook(encoding='utf-8', style_compression=0)
    sheet = mid5.add_sheet('Sheet', cell_overwrite_ok=True)

    for i in range(1, 101):
        sheet.write(i, 0, i)
    sheet.write(0, 0, 'INDEX')
    sheet.write(0, 1, 'SUBJECT')
    sheet.write(0, 2, 'SIMILARITY')

    text_w = 10
    pic_w = 2
    typ_w = 15
    bac_w = 5

    pic_stand = 1 / (math.sqrt(14028))
    typ_stand = 1 / (math.sqrt(6339))
    bac_stand = 1 / (math.sqrt(8085))

    target = int(input("请输入需要检测相似度的题号: "))

    str1 = questions.values[target-1][1].replace("\n", "").replace("\t", "").
   replace(" ", "")  # 被测题目
    pic = questions.values[target-1][2]
    typ = questions.values[target-1][3]
    bac = questions.values[target-1][4]

    # similarity = np.zeros((100, 100), float)
```

```
105     # print(similarity[99][99])
106
107     for i in range(0, 100):
108         str2 = file.values[i][1].replace("\r", "").replace("\n", "").replace("\
    t", "").replace(" ", "")  # 竖行题目
109         sheet.write(i+1, 1, str2)
110
111         # print(str2)
112         # print("==========")
113
114         s1 = simhash.simHash(str1)
115         s2 = simhash.simHash(str2)
116         text_num = (64-simhash.getDistance(s1, s2))/64/6
117
118         if file.values[i][2] == pic:
119             pic_num = pic_stand * 2
120         else:
121             pic_num = pic_stand
122
123         if file.values[i][3] == typ:
124             typ_num = pic_stand * 2
125         else:
126             typ_num = pic_stand
127
128         if file.values[i][4] == bac:
129             bac_num = bac_stand * 2
130         else:
131             bac_num = bac_stand
132
133         similarity = text_num * text_w + pic_num * pic_w + \
134                      typ_num * typ_w + bac_num * bac_w
135
136         sheet.write(i+1, 2, similarity)
137
138     print("您要查找与该题相似的题目: ")
139     print(str1)
140     print("------------------------------")
141     print("查询成功, 请查看文件Search_result.xls! ")
142     print("------------------------------")
143     save_path = "Search_result.xls"
144     mid5.save(save_path)
```

文件名：similar.py

```
import numpy
import numpy as np
import pandas as pd
import openpyxl as op
import xlwt
import math
import xlrd

if __name__ == '__main__':
    text = pd.read_excel("result.xls")
    pic = pd.read_excel("result3.xls", sheet_name='Sheet1')
    typ = pd.read_excel("result3.xls", sheet_name='Sheet2')
    bac = pd.read_excel("result3.xls", sheet_name='Sheet3')

    mid3 = xlwt.Workbook(encoding='utf-8', style_compression=0)
    sheet = mid3.add_sheet('Sheet1', cell_overwrite_ok=True)

    for i in range(1, 101):
        sheet.write(0, i, i)
    for i in range(1, 101):
        sheet.write(i, 0, i)

    text_w = 10
    pic_w = 2
    typ_w = 15
    bac_w = 5

    for i in range(0, 100):
        for j in numpy.arange(i + 1, 100):
            text_num = text.values[i][j + 1]/6
            pic_num = pic.values[i][j + 1]
            typ_num = typ.values[i][j + 1]
            bac_num = bac.values[i][j + 1]
            similarity = text_num * text_w + pic_num * pic_w + \
                         typ_num * typ_w + bac_num * bac_w

            sheet.write(i+1, j+1, similarity)

    save_path = "similarity.xls"
    mid3.save(save_path)
```

文件名：tag.py

```
import numpy
import numpy as np
import pandas as pd
import openpyxl as op
import xlwt
import math
import xlrd

if __name__ == '__main__':
    last_res = pd.read_excel("result.xls")
    tag_file = pd.read_excel("tagsimilar.xlsx")
    mid1 = xlwt.Workbook(encoding='utf-8', style_compression=0)
    sheet1 = mid1.add_sheet('Sheet1', cell_overwrite_ok=True)
    sheet2 = mid1.add_sheet('Sheet2', cell_overwrite_ok=True)
    sheet3 = mid1.add_sheet('Sheet3', cell_overwrite_ok=True)
    for i in range(1, 101):
        sheet1.write(0, i, i)
        sheet2.write(0, i, i)
        sheet3.write(0, i, i)
    for i in range(1, 101):
        sheet1.write(i, 0, i)
        sheet2.write(i, 0, i)
        sheet3.write(i, 0, i)
    sum1 = 0
    sum2 = 0
    sum3 = 0
    count = 0
    # pic_res = np.ones((100, 100), int)
    # type_res = np.ones((100, 100), int)
    # back_res = np.ones((100, 100), int)
    # pd把result的第一行去除了
    for i in range(0, 100):
        for j in range(i+1, 100):
            count += 1
            pic1 = tag_file.values[i][2]
            pic2 = tag_file.values[j][2]
            if pic1 == pic2:
                sheet1.write(i+1, j+1, 2)
                sum1 += 4
            else:
```

```
                sheet1.write(i+1, j+1, 1)
                sum1 += 1

            type1 = tag_file.values[i][3]
            type2 = tag_file.values[j][3]
            if type1 == type2:
                sheet2.write(i+1, j+1, 2)
                sum2 += 4
            else:
                sheet2.write(i+1, j+1, 1)
                sum2 += 1

            back1 = tag_file.values[i][4]
            back2 = tag_file.values[j][4]
            if back1 == back2:
                sheet3.write(i+1, j+1, 2)
                sum3 += 4
            else:
                sheet3.write(i+1, j+1, 1)
                sum3 += 1

    print(sum1, sum2, sum3, count)

    # save_path = "result2.xls"
    # mid1.save(save_path)
```

文件名: tag_stand.py

```
import numpy
import numpy as np
import pandas as pd
import openpyxl as op
import xlwt
import math
import xlrd

# 14028 6339 8085

if __name__ == '__main__':
    text_res = pd.read_excel("result.xls")
    tag_mid_pic = pd.read_excel("result2.xls", sheet_name='Sheet1')
    tag_mid_type = pd.read_excel("result2.xls", sheet_name='Sheet2')
    tag_mid_back = pd.read_excel("result2.xls", sheet_name='Sheet3')
```

```
16
17      mid2 = xlwt.Workbook(encoding='utf-8', style_compression=0)
18      sheet1 = mid2.add_sheet('Sheet1', cell_overwrite_ok=True)
19      sheet2 = mid2.add_sheet('Sheet2', cell_overwrite_ok=True)
20      sheet3 = mid2.add_sheet('Sheet3', cell_overwrite_ok=True)
21      for i in range(1, 101):
22          sheet1.write(0, i, i)
23          sheet2.write(0, i, i)
24          sheet3.write(0, i, i)
25      for i in range(1, 101):
26          sheet1.write(i, 0, i)
27          sheet2.write(i, 0, i)
28          sheet3.write(i, 0, i)
29      pic_stand = 1 / (math.sqrt(14028))
30      type_stand = 1 / (math.sqrt(6339))
31      back_stand = 1 / (math.sqrt(8085))
32
33      # print(tag_mid_pic.values[1][3])
34      # print(tag_mid_type.values[1][3])
35      # print(tag_mid_back.values[1][3])
36
37      for i in range(0, 100):
38          for j in range(i+1, 100):
39              if tag_mid_pic.values[i][j+1] == 1:
40                  sheet1.write(i+1, j+1, pic_stand)
41              else:
42                  sheet1.write(i + 1, j + 1, pic_stand*2)
43
44              if tag_mid_type.values[i][j+1] == 1:
45                  sheet2.write(i+1, j+1, type_stand)
46              else:
47                  sheet2.write(i+1, j+1, type_stand*2)
48
49              if tag_mid_back.values[i][j+1] == 1:
50                  sheet3.write(i+1, j+1, back_stand)
51              else:
52                  sheet3.write(i+1, j+1, back_stand*2)
53
54      save_path = "../search_system/result3.xls"
55      mid2.save(save_path)
```

文件名：test.py

```
# -*- coding:utf-8 -*-
import jieba
import jieba.analyse
import numpy
import numpy as np
import pandas as pd
import openpyxl as op
import xlwt


class SimHash(object):
    def simHash(self, content):
        seg = jieba.cut(content)
        jieba.analyse.set_stop_words('stopwords.txt')
        keyWords = jieba.analyse.extract_tags("|".join(seg), topK=10,
    withWeight=True)
        # print(keyWords)
        keyList = []
        for feature, weight in keyWords:
            # weight = math.ceil(weight)
            weight = int(weight * 10) + 1
            # print('weight: {}'.format(weight))
            binstr = self.string_hash(feature)
            temp = []
            for c in binstr:
                if (c == '1'):
                    temp.append(weight)
                else:
                    temp.append(-weight)
            keyList.append(temp)
        listSum = np.sum(np.array(keyList), axis=0)
        # print(listSum)
        if (keyList == []):
            return '00'
        simhash = ''
        for i in listSum:
            if (i > 0):
                simhash = simhash + '1'
            else:
                simhash = simhash + '0'

```

```
41          return simhash
42
43      def string_hash(self, source):
44          if source == "":
45              return 0
46          else:
47              x = ord(source[0]) << 7
48              m = 1000003
49              mask = 2 ** 128 - 1
50              for c in source:
51                  x = ((x * m) ^ ord(c)) & mask
52              x ^= len(source)
53              if x == -1:
54                  x = -2
55              x = bin(x).replace('0b', '').zfill(64)[-64:]
56              # print('strint_hash: %s, %s' % (source, x))
57
58              return str(x)
59
60      def getDistance(self, hashstr1, hashstr2):
61          '''
62              计算两个simhash的汉明距离
63          '''
64          length = 0
65          for index, char in enumerate(hashstr1):
66              if char == hashstr2[index]:
67                  continue
68              else:
69                  length += 1
70
71          return length
72
73
74  if __name__ == '__main__':
75      simhash = SimHash()
76      file = pd.read_excel("textsimilar.xlsx")
77      res_file = xlwt.Workbook(encoding='utf-8', style_compression=0)
78      sheet = res_file.add_sheet('Sheet', cell_overwrite_ok=True)
79      for i in range(1, 101):
80          sheet.write(0, i, i)
81      for i in range(1, 101):
```

```
        sheet.write(i, 0, i)
    res = np.zeros((100, 100), int)
    count = 1
    print(file.values[1][1])
    for i in range(0, 100):
        for j in numpy.arange(i+1, 100):
            str1 = file.values[i][1].replace("\r", "").replace("\n", "").replace("\t", "").replace(" ", "")  # 横行题目
            str2 = file.values[j][1].replace("\r", "").replace("\n", "").replace("\t", "").replace(" ", "")  # 竖行题目
            s1 = simhash.simHash(str1)
            s2 = simhash.simHash(str2)
            dis = simhash.getDistance(s1, s2)
            # print(s1)
            # print(s2)
            # print('dis: {}'.format(dis))
            # res[i][j-1] = dis
            dis = (64-dis)/64
            sheet.write(i+1, j+1, dis)

            # count += 1
#         # print("==========================================")
# savepath = "result.xls"
# res_file.save(savepath)

    # str1 = """项目概况四川省""".replace("\r", "").replace("\n", "").replace("\t", "").replace(" ", "")
    # str2 = """项目概况""".replace("\r", "").replace("\n", "").replace("\t", "").replace(" ", "")
```

文件名：kmeans.py

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd

# 设置字体为楷体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']

# 生成数据
# x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
# y = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
FilePath = "../110score.xls"

data = pd.read_excel(FilePath)
data = data.values
y = np.array(data[0:110, 2])
x = range(1, 111)

added_y = np.array(data[100:110, 2])
added_x = range(101, 111)
# 计算频率分布
plt.figure()
n, bins, patches = plt.hist(y, bins=10)
plt.title("题目难度分布直方图", fontsize=14)
plt.xlabel("题目难度", fontsize=12)
plt.ylabel("题目个数", fontsize=12)
plt.savefig("../picture/34difficult_distribution.png", dpi=300)
plt.show()

# 使用K均值聚类算法将数据分成3类
kmeans = KMeans(n_clusters=3, random_state=0).fit(y.reshape(-1, 1))
labels = kmeans.fit_predict(y.reshape(-1, 1))
print(labels)
low = []
middle = []
high = []
for i in range(0, len(labels)):
    # print(data[i, 0])
    if labels[i] == 0:
        low.append(i)
    elif labels[i] == 1:
        middle.append(i)
    else:
        high.append(i)

plt.figure()
plt.scatter(x[0:100], y[0:100], c=labels[0:100], marker='.', linewidths=2)
plt.title("题目难度分类散点图", fontsize=14)
plt.xlabel("题目编号", fontsize=12)
plt.ylabel("题目难度", fontsize=12)

```

```
# plt.scatter(added_x, added_y, c=labels[100:110], marker='.', linewidths=2)
plt.scatter(added_x, added_y, c=labels[100:110], marker='x', s=80,linewidths=2)
# plt.scatter(added_x, added_y, c="none", marker='o', s=140,linewidths=2,
    edgecolors='r')
plt.savefig("../picture/34difficult_classify.png", dpi=300)

plt.show()
```

文件名：time.py

```
import matplotlib
import numpy as np
from bokeh.palettes import mpl
from matplotlib.lines import Line2D
from mpl_toolkits.mplot3d import Axes3D

import time
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering, KMeans

# 设置字体为楷体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']


# 绘制时间复杂度曲线
def get_time_complexity(algorithm, n_clusters, X):
    times = []
    start_time = time.time()
    algorithm(n_clusters=n_clusters).fit(X)
    end_time = time.time()
    return end_time - start_time


# 生成随机数据
np.random.seed(0)
# X = np.random.randn(1000, 2)

X = np.array(range(100, 1000, 100))
Y = np.array(range(2, 10))
XX, YY = np.meshgrid(X, Y)
ZZ_Agg = np.zeros((Y.size, X.size))
ZZ_Kmeans = np.zeros((Y.size, X.size))
for i in range(0, X.size):
```

```
    data_len = X[i]
    # for data_len in X:
    Matrix = np.random.randn(data_len, 2)
    for j in range(0, Y.size):
        clusters_num = Y[j]
        # for clusters_num in Y:
        spend_time_Agg = get_time_complexity(AgglomerativeClustering, clusters_num, Matrix)
        spend_time_Kmeans = get_time_complexity(KMeans, clusters_num, Matrix)
        ZZ_Agg[j, i] = spend_time_Agg
        ZZ_Kmeans[j, i] = spend_time_Kmeans
figure = plt.figure()
ax = Axes3D(figure)
ax.plot_surface(XX, YY, ZZ_Agg, rstride=1, cstride=1, cmap='YlGn', label='层次聚类算法')
ax.plot_surface(XX, YY, ZZ_Kmeans, rstride=1, cstride=1, cmap='BuPu', label='KMeans算法')

ax.set_xlabel('数据量')  # 为子图设置横轴标题
ax.set_ylabel('聚类个数')  # 为子图设置纵轴标题
ax.set_zlabel("花费时间/s")
# plt.legend()
plt.savefig("../picture/time.png", dpi=300)
y_unique = np.unique(2)   # 可以看作图例类型个数
color = ['p', 'g']  # 颜色集
legend_lines = [Line2D([0], [0], linestyle="none", marker='o', c=color[y]) for y in y_unique]
ax.legend(legend_lines, ['层次聚类算法', 'KMeans算法'], numpoints=1)

plt.show()

## 绘制AGNES算法的时间复杂度曲线
# n_clusters_range = range(2, 21)
# plot_time_complexity(AgglomerativeClustering, n_clusters_range, X)
# plt.title("AGNES算法时间复杂度曲线")
# plt.xlabel("分类数")
# plt.ylabel("时间 (s)")
# plt.legend()
# plt.savefig("../picture/KMeans.png", dpi=300)
# plt.show()
#
```

```
## 绘制K-means算法的时间复杂度曲线
# n_clusters_range = range(2, 21)
# plot_time_complexity(KMeans, n_clusters_range, X)
# plt.title("K-means算法时间复杂度曲线")
# plt.xlabel("分类数")
# plt.ylabel("时间 (s)")
# plt.legend()
# plt.savefig("../picture/AGNES.png", dpi=300)
# plt.show()
```

文件名：cal_difficult_score.py

```
import xlrd
import xlwt
import math
import pandas as pd
import numpy as np
from numpy import multiply


def load_data(FilePath, ExcelSheet):
    workbook = xlrd.open_workbook(str(FilePath)) # excel路径
    sheet = workbook.sheet_by_name(ExcelSheet) # sheet表
    return sheet


def normalization(sheet, StartCol, EndCol, StartRow, EndRow,
                  SavePath): # 起始列，终止列，起始行，终止行，逆向指标所在列(列表格式)，保存路径
    #######创建一个workbook########
    workbook = xlwt.Workbook(encoding='utf-8', style_compression=0) # 创建一个workbook 设置编码
    worksheet = workbook.add_sheet('Sheet') # 创建一个worksheet
    #######读取行列，并计算极差标准化值#######
    for j in range(StartCol - 1, EndCol): # 获取列代码
        Cols = sheet.col_values(j)
        # ColsName = Cols[0]          #获取列名并删除
        del Cols[0] # 删除列名
        ColsMax = max(Cols) # 获取整列的最大值，最小值
        ColsMin = min(Cols)
        ExtremeSub = ColsMax - ColsMin # 计算极差

        pingfang = [ele * ele for ele in Cols]
```

```
        pingfang_sum = sum(pingfang)
        sqrt_pingfang_sum = math.sqrt(pingfang_sum)
        for i in range(StartRow - 1, EndRow):  # 获取行代码
            PreValue = sheet.cell(i, j).value  # 获取对应行列单元格里数据
            Value = PreValue / sqrt_pingfang_sum
            # Value = (PreValue - ColsMin) / ExtremeSub
            # for Conve in ConverseCol:  # 计算逆向指标
            #     if j == Conve - 1:  # 如果是逆向指标列，采用另一种计算方法
            #         Value = (ColsMax - PreValue) / ExtremeSub  # 逆向指标值计算
            worksheet.write(i, j, label=Value)  # 写入对应单元格
    workbook.save(SavePath)  # 保存


FilePath = "110题标准化数据.xls"
# ExcelSheet = "Sheet"
# sheet = load_data(FilePath, ExcelSheet)
# StartCol = 1
# EndCol = 10
# StartRow = 2
# EndRow = 101
weight = np.matrix(
    [0.005501593, 0.047812898, 0.079703256, 0.054741035, 0.04680106,
     0.023046825, 0.135867117, 0.11588154, 0.453523048,
     0.037121628
     ]).T

data = pd.read_excel(FilePath)
data = data.values

maxrow = np.max(data, axis=0)
max_all = np.expand_dims(maxrow, 0).repeat(110, axis=0)
minrow = np.min(data, axis=0)
min_all = np.expand_dims(minrow, 0).repeat(110, axis=0)

Dadd_mat = np.multiply(max_all - data, max_all - data)
# Dadd_sum = np.sum(Dadd_mat, axis=1)
Dadd = np.dot(Dadd_mat, weight)

Dsub_mat = np.multiply(data - min_all, data - min_all)
# Dsub_sum = np.sum(Dsub_mat, axis=1)
```

```
Dsub = np.dot(Dsub_mat, weight)

Dadd_sum = np.sum(Dadd, axis=1)
Dsub_sum = np.sum(Dsub, axis=1)
S = Dsub_sum / (Dadd_sum + Dsub_sum)
print(S)
writer = pd.DataFrame(S)
# index参数设置为False表示不保存行索引,header设置为False表示不保存列索引
writer.to_excel("110score.xls", index=False, header=False)
```

文件名: cal_weight.py

```
import tkinter as tk
from tkinter.filedialog import *
from tkinter.messagebox import *
import xlrd
import xlwt
from math import log


# 熵值法处理数据部分
##########1.读取Excel文件##########
def load_data(FilePath, ExcelSheet):
    workbook = xlrd.open_workbook(str(FilePath))  # excel路径
    sheet = workbook.sheet_by_name(ExcelSheet)  # sheet表
    return sheet


##########2.熵值法运算########
def get_entropy(sheet, StartCol, EndCol, StartRow, EndRow, SavePath):  # 起始列, 终止列, 起始行, 终止行, 保存路径
    #######创建一个workbook########
    workbook = xlwt.Workbook(encoding='utf-8')  # 创建一个workbook 设置编码
    worksheet = workbook.add_sheet('Sheet')  # 创建一个worksheet
    ######获取样本数和指标数######
    m = EndCol + 1 - StartCol  # 指标数
    n = EndRow + 1 - StartRow  # 样本数
    ColBlank = StartCol - 1  # 前面的空列数
    #####获取计算熵值法需要的元素#######
    AbovePartList = []
    LCLSBVS = []
    for j in range(StartCol - 1, EndCol):  # 获取列代码
        ColLSBV = []
```

```
        Cols = sheet.col_values(j)
        # ColsName = Cols[0]          #获取列名并删除
        del Cols[0]  # 删除列名
        EachSampleInThisIndicatorSum = sum(Cols)  # 每个样本的该列指标求和
        for i in range(StartRow - 1, EndRow):
            PreValue = sheet.cell(i, j).value
            if PreValue == 0:
                PreValue = 0.000000001
            LnSumBracketVar = (PreValue / EachSampleInThisIndicatorSum) * log(
    PreValue / EachSampleInThisIndicatorSum)
            ColLSBV.append(LnSumBracketVar)
        ColLSBVSum = sum(ColLSBV)  # 计算括号内的和
        LnCLSBVS = ColLSBVSum / log(n)  # 除以 ln(n)
        LCLSBVS.append(LnCLSBVS)
        #########熵值法公式上下部分######
        AbovePart = 1 + LnCLSBVS  # 上半部分
        AbovePartList.append(AbovePart)  # 把上半部分存为列表
    LCLSBVSSum = sum(LCLSBVS)
    BelowPart = m + LCLSBVSSum  # 下半部分固定值
    ##########写入#############
    for j in range(0, m):  # 指标数
        Value = AbovePartList[j] / BelowPart  # 值等于上半部分除以下半部分
        worksheet.write(StartRow - 1, j + ColBlank, label=Value)  # 写入
    workbook.save(str(SavePath))  # 保存


# GUI相关函数
# 初始文件路径
def Get_FilePath():
    file_get = askopenfilename()  # 选择打开什么文件，返回文件名
    file_path.set(file_get)  # 设置变量filename的值


# 获取保存路径
def Get_SavePath():
    save_get = asksaveasfilename(defaultextension='.xls')  # 设置保存文件，并返
    回文件名，指定文件名后缀为.xls
    save_path.set(save_get)  # 设置变量filename的值


# 极差标准化执行模块
```

```
70  def exe_norma():
71      try:
72          Sheet = load_data(str(file_path.get()), str(read_sheet_name.get()))  #
        读取Sheet
73          get_entropy(Sheet, int(start_col.get()), int(end_col.get()), int(
        start_row.get()), int(end_row.get()),
74                      str(save_path.get()))  # 极差标准化与保存Sheet
75          # 从上面读取的sheet, 起始列, 终止列, 起始行, 终止行, 逆向指标所在列（列
        表格式）, 保存路径
76          showinfo(title='执行完毕', message=f'熵值法处理数据成功, 结果保存在{str
        (save_path.get())}')
77      except:
78          showwarning(title='执行出错', message='无法执行熵值计算, 请检查所输入的
        信息')
79
80
81  # GUI界面部分
82  root = tk.Tk()
83  root.title('熵值法处理工具')
84  root.geometry('800x330')
85  file_path = tk.StringVar()
86  save_path = tk.StringVar()
87
88  # 选择文件、获取路径
89  tk.Label(root, text='选择文件: ').place(x=10, y=20)
90  tk.Entry(root, textvariable=file_path, width=90).place(x=80, y=20)
91  tk.Button(root, text='打开文件', command=Get_FilePath).place(x=730, y=15)
92
93  # 读取的sheet名称
94  tk.Label(root, text='读取sheet: ').place(x=10, y=70)
95  read_sheet_name = tk.Entry(root, width=90)
96  read_sheet_name.place(x=80, y=70)
97
98  # 起始、终止行列
99  tk.Label(root, text='起始列: ').place(x=10, y=120)
100 start_col = tk.Entry(root, width=13)
101 start_col.place(x=80, y=120)
102 tk.Label(root, text='终止列: ').place(x=190, y=120)
103 end_col = tk.Entry(root, width=13)
104 end_col.place(x=260, y=120)
105 tk.Label(root, text='起始行: ').place(x=370, y=120)
```

```
start_row = tk.Entry(root, width=13)
start_row.place(x=440, y=120)
tk.Label(root, text='终止行: ').place(x=550, y=120)
end_row = tk.Entry(root, width=13)
end_row.place(x=620, y=120)

# 保存路径
tk.Label(root, text='保存路径: ').place(x=10, y=170)
tk.Entry(root, textvariable=save_path, width=90).place(x=80, y=170)
tk.Button(root, text='选择目录', command=Get_SavePath).place(x=730, y=165)

# 运行按钮
tk.Button(root, text='开始进行熵值法处理计算', command=exe_norma, width=30,
    height=2).place(x=290, y=230)

root.mainloop()
```

文件名: cluster.py

```
import matplotlib
import numpy as np
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage, fcluster, dendrogram
import matplotlib.pyplot as plt
import pandas as pd

matplotlib.rcParams['font.sans-serif'] = ['SimHei']
FilePath = "similarity.xls"

data = pd.read_excel(FilePath)
data.fillna(0, inplace=True)
data = data.values
data = data + data.transpose()
max_value = np.max(data)
data = max_value - data
for i in range(0, 100):
    data[i][i] = 0
# 生成距离矩阵
# dist_matrix = np.array([[0, 1, 2, 3],
#                         [1, 0, 4, 5],
#                         [2, 4, 0, 6],
#                         [3, 5, 6, 0]])
dist_matrix = data
```

```
# 使用pdist函数将距离矩阵转换为压缩的距离矩阵
dist_condensed = pdist(dist_matrix)

# 使用linkage函数计算簇之间的距离
Z = linkage(dist_condensed, method='complete',)

# 使用fcluster函数获取簇标签
labels = fcluster(Z, t=2.55, criterion='distance')
rst = []
for i in range(0, np.max(labels)):
    rst.append([])

for i in range(0, len(labels)):
    label = labels[i]
    rst[label - 1].append(i + 1)

for i in range(0, np.max(labels)):
    print(rst[i])

# 可视化结果
plt.figure(figsize=(20, 5))
dendrogram(Z,labels=range(1,101),leaf_font_size=12)
plt.xlabel("题目编号", fontsize=13)
plt.savefig("picture/similarity.png", dpi=300)
plt.show()

# plt.figure()
# plt.scatter(range(len(labels)), [0] * len(labels), c=labels, cmap='rainbow')
# plt.show()
```

文件名：standard.py

```
import math
import tkinter as tk
from tkinter.filedialog import *
from tkinter.messagebox import *
import xlrd
import xlwt


# 极差标准化部分
##########1.读取Excel文件##########
def load_data(FilePath, ExcelSheet):
```

```
    workbook = xlrd.open_workbook(str(FilePath))  # excel路径
    sheet = workbook.sheet_by_name(ExcelSheet)  # sheet表
    return sheet


##########2极差标准化运算########
def normalization(sheet, StartCol, EndCol, StartRow, EndRow, ConverseCol,
                  SavePath):  # 起始列，终止列，起始行，终止行，逆向指标所在列（列表格式），保存路径
    #######创建一个workbook########
    workbook = xlwt.Workbook(encoding='utf-8', style_compression=0)  # 创建一个workbook 设置编码
    worksheet = workbook.add_sheet('Sheet')  # 创建一个worksheet
    #######读取行列，并计算极差标准化值#######
    for j in range(StartCol - 1, EndCol):  # 获取列代码
        Cols = sheet.col_values(j)
        # ColsName = Cols[0]          #获取列名并删除
        del Cols[0]  # 删除列名
        ColsMax = max(Cols)  # 获取整列的最大值，最小值
        ColsMin = min(Cols)
        ExtremeSub = ColsMax - ColsMin  # 计算极差

        pingfang = [ele * ele for ele in Cols]
        pingfang_sum = sum(pingfang)
        sqrt_pingfang_sum = math.sqrt(pingfang_sum)
        for i in range(StartRow - 1, EndRow):  # 获取行代码
            PreValue = sheet.cell(i, j).value  # 获取对应行列单元格里数据
            Value = PreValue / sqrt_pingfang_sum
            # Value = (PreValue - ColsMin) / ExtremeSub
            for Conve in ConverseCol:  # 计算逆向指标
                if j == Conve - 1:  # 如果是逆向指标列，采用另一种计算方法
                    Value = (ColsMax - PreValue) / ExtremeSub  # 逆向指标值计算
            worksheet.write(i, j, label=Value)  # 写入对应单元格
    workbook.save("110题标准化数据.xls")  # 保存


# GUI相关函数
# 初始文件路径
def Get_FilePath():
    file_get = askopenfilename()  # 选择打开什么文件，返回文件名
    file_path.set(file_get)  # 设置变量filename的值
```

```
51
52
53  # 获取保存路径
54  def Get_SavePath():
55      save_get = asksaveasfilename(defaultextension='.xls')  # 设置保存文件，并返
        回文件名，指定文件名后缀为.xls
56      save_path.set(save_get)  # 设置变量filename的值
57
58
59  # 极差标准化执行模块
60  def exe_norma():
61      try:
62          Sheet = load_data(str(file_path.get()), str(read_sheet_name.get()))  #
        读取Sheet
63          normalization(Sheet, int(start_col.get()), int(end_col.get()), int(
        start_row.get()), int(end_row.get()),
64                        eval(converse_col.get()),
65                        str(save_path.get()))  # 极差标准化与保存Sheet
66          # 从上面读取的sheet，起始列，终止列，起始行，终止行，逆向指标所在列（列
        表格式），保存路径
67          showinfo(title='执行完毕', message=f'极差标准化计算成功，结果保存在{str
        (save_path.get())}')
68      except:
69          showwarning(title='执行出错', message='无法执行极差标准化计算，请检查所
        输入的信息')
70
71
72  # GUI界面部分
73  root = tk.Tk()
74  root.title('极差标准化工具')
75  root.geometry('800x380')
76  file_path = tk.StringVar()
77  save_path = tk.StringVar()
78
79  # 选择文件、获取路径
80  tk.Label(root, text='选择文件: ').place(x=10, y=20)
81  tk.Entry(root, textvariable=file_path, width=90).place(x=80, y=20)
82  tk.Button(root, text='打开文件', command=Get_FilePath).place(x=730, y=15)
83
84  # 读取的sheet名称
85  tk.Label(root, text='读取sheet: ').place(x=10, y=70)
```

```
read_sheet_name = tk.Entry(root, width=90)
read_sheet_name.place(x=80, y=70)

# 起始、终止行列
tk.Label(root, text='起始列: ').place(x=10, y=120)
start_col = tk.Entry(root, width=13)
start_col.place(x=80, y=120)
tk.Label(root, text='终止列: ').place(x=190, y=120)
end_col = tk.Entry(root, width=13)
end_col.place(x=260, y=120)
tk.Label(root, text='起始行: ').place(x=370, y=120)
start_row = tk.Entry(root, width=13)
start_row.place(x=440, y=120)
tk.Label(root, text='终止行: ').place(x=550, y=120)
end_row = tk.Entry(root, width=13)
end_row.place(x=620, y=120)

# 逆向指标所在列
tk.Label(root, text='逆向指标列: ').place(x=10, y=170)
converse_col = tk.Entry(root, width=30)
converse_col.place(x=80, y=170)
tk.Label(root, text='用中括号"[ ]"括起, 列号用英文输入下的逗号","分隔, 如无逆向
    指标输入""').place(x=310, y=170)

# 保存路径
tk.Label(root, text='保存路径: ').place(x=10, y=220)
tk.Entry(root, textvariable=save_path, width=90).place(x=80, y=220)
tk.Button(root, text='选择目录', command=Get_SavePath).place(x=730, y=215)

# 运行按钮
tk.Button(root, text='开始进行极差标准化计算', command=exe_norma, width=30,
    height=2).place(x=290, y=280)

root.mainloop()
```