队伍编号	MC2311173
题号	（A）

量子计算机在信用评分卡组合优化中的应用

摘 要

本文基于 QUBO 模型针对信用评分卡组合优化的问题进行了建模求解。

针对问题 1，在满足所有选择阈值数量总和为 1 的约束条件下，构建了 QUBO 模型的 Q 矩阵，并使用罚函数添加约束条件，采用 python3.9 中 neal 库的量子模拟退火算法进行求解，得到最优阈值为

针对问题 2，在满足所有选择阈值数量总和为 3 以及阈值所属信用评分卡不同的约束条件下，使用模拟退火算法对 QUBO 模型进行求解，得到最优阈值组合为

针对问题 3，提出了三种方法进行求解，方法一采用类似问题二中的常规解法进行求解。方法二提出了一种求解思路，使得求解模型可以转变为类似问题一中的求解模型，但是由于这种方法计算体积非常庞大，因此提出了方法三。方法三将求解目标降维转化为近似求解目标，将问题拆解为两步，第一步将通过求解最优二组合实现数据降维；第二步基于第一步筛选出的阈值进行三组合寻优求解，最终实现目标求解。虽然方法三可能得到局部最优解，但是他的计算效率更高，鲁棒性更好。

关键词：组合优化问题、二次无约束优化、梯度下降、模拟退火算法、数据降维

目录

1、背景简介

近年来，随着信息技术的不断发展，大量的数据产生和储存给金融行业带来了前所未有的机遇和挑战。在金融行业中，信用评分卡是一种常用的风险评估工具，用于评估客户的信用风险和财务状况。然而，由于评估的因素往往非常复杂，包括客户的个人信息、历史信用记录、还款能力等，因此对于大量客户的评估往往需要耗费大量时间和人力物力。

为了解决这个问题，研究者们提出了一种基于量子计算机的信用评分卡优化方法。这种方法利用了量子计算机处理大规模数据的高效性和并行处理的特点，通过建立数学模型，优化信用评分卡组合的方式，进一步提高评估效率和准确度。

在量子计算机的信用评分卡组合优化中，研究者们利用了量子计算机的特性，即量子叠加和量子纠缠，实现了对大量信用评分卡的优化。首先，研究者们将信用评分卡的各个因素抽象为数学模型，并利用量子比特进行编码和处理。然后，通过优化模型中的目标函数，研究者们得到了最优的信用评分卡组合方案。

QUBO（Quadratic Unconstrained Binary Optimization）是一种优化问题的数学模型[1]。它主要用于处理离散变量的优化问题，特别是处理布尔变量的优化问题。QUBO 模型的主要目标是最小化目标函数，而目标函数通常包含二次项和线性项。在 QUBO 模型中，变量的取值只能是 0 或 1，这使得它非常适用于解决组合优化问题[2]。

QUBO 模型被广泛应用于许多领域，例如计算机科学、物理学、金融学和生物学等。特别是在量子计算机领域，QUBO 模型被广泛应用于量子优化算法中，这些算法利用量子计算机的特性来解决大规模的优化问题[3]。

QUBO 模型通常可以表示为一个矩阵，其中矩阵的每个元素代表一个变量或变量之间的关系，矩阵的每一行或每一列表示一个约束条件。QUBO 模型的求解通常需要将其转化为一个关于二进制变量的二次规划问题，进而转化为一个关于二进制变量的整数规划问题，最终转化为关于 0 和 1 的布尔代数问题。

目前，一些经典的优化算法，例如模拟退火算法、遗传算法和梯度下降算法等，以及一些量子优化算法，例如量子近似优化算法（QAOA）和变分量子本征求解器（VQE）等，都可以用来解决 QUBO 模型的优化问题。

在量子计算机上求解 QUBO 模型通常需要将其转化为一个哈密顿量问题，并通过量子比特之间的纠缠关系，利用量子并行性来进行求解。当前，世界上唯一商业化生产的量子计算机 D-Wave 系统就采用了这种方法。D-Wave Systems（D-Wave）公司成立于 1999 年，是全球第一家商业化生产量子计算机的公司，它推出的第一款量子计算机于 2007 年上市。

2、赛题

在银行信用卡或相关的贷款等业务中，对客户授信之前，需要先通过各种审核规则对客户的信用等级进行评定，通过评定后的客户才能获得信用或贷款资格。规则审核过程实际是经过一重或者多重组合规则后对客户 进行打分，这些规则就被称为信用评分卡，每个信用评分卡又有多种阈值设置（但且只有一个阈值生效），这就使得不同的信用评分卡在不同的阈值 下，对应不同的通过率和坏账率，一般通过率越高，坏账率也会越高，反之，通过率越低，坏账率也越低。

对银行来说，通过率越高，通过贷款资格审核的客户数量就越多，相应的银行获得的利息收入就会越多，但高通过率一般对应着高坏账率，而 坏账意味着资金的损失风险，因此银行最终的收入可以定义为： 最终收入 = 贷款利息收入 — 坏账损失

信用评分卡 1			信用评分卡 2			信用评分卡 3		
阈值	通过率	坏账率	阈值	通过率	坏账率	阈值	通过率	坏账率
1	5%	0.50%	1	5%	0.50%	1	5%	0.50%
2	10%	1.00%	2	10%	1.00%	2	10%	1.00%
3	25%	1.50%	3	25%	1.50%	3	20%	1.70%
4	30%	2.00%	4	30%	2.00%	4	33%	2.00%
5	40%	2.50%	5	45%	2.50%	5	40%	2.70%
6	50%	3.00%	6	50%	2.70%	6	52%	3.00%
7	60%	3.50%	7	65%	3.50%	7	62%	3.70%
8	70%	4.00%	8	70%	4.00%	8	73%	4.00%
9	80%	4.50%	9	82%	4.70%	9	82%	4.70%
10	93%	5.00%	10	90%	5.00%	10	95%	5.00%

赛题说明 1 ：流程简化及示例

由于银行场景的复杂性，往往需要采用选择多个不同的信用评分卡进行组合来实现最佳的风险控制策略。而实际中的信用评分卡组合是一个非常复杂的过程，为便于建模，我们将该问题进行做如下简化（本简化只适用本次比赛赛题，不能完全代表实际场景）。

假设贷款资金为 1000000 元，银行贷款利息收入率为 8%，并以上面列举的三个信用评分卡作为选定的信用评分卡组合来测算银行最终收入。

由于每一信用评分卡有且只可选择 1 个阈值，假设信用评分卡 1 的阈 值设置为 8，则通过表格可知，对应通过率为 70%，坏账率为 4.00%，信用评分卡 2 的阈值设置为 6，则通过率为 50%，坏账率为 2.70%，信用评分卡 3 的阈值设置为 7，则通过率为 62%，坏账率为 3.70%。

例如如果我们选择三重信用卡组合策略，那么这三种信用评分卡组合后的总通过率为所有信用评分卡通过率相乘，即：

0.7×0.5×0.62 = 0.217

总坏账率为三种信用评分卡对应坏账率的平均值，即：

1/3×(0.04+0.027+0.037) = 0.0367

基于以上条件可求得，本次贷款利息收入为：

贷款资金×利息收入率×总通过率×（1－总坏账率），即：

1000000×0.08×(0.7×0.5×0.62) ×(1－1/3×(0.04+0.027+0.037)) 16758.18(元)

由坏账带来的坏账损失为： 贷款资金×总通过率×总坏账率，即：

1000000×(0.7×0.5×0.62) ×(1/3×(0.04+0.027+0.037))=7522.666(元)

那么银行的 最终收入 为： 贷款利息收入 － 坏账损失 ，即

16758.18－7522.666 = 9235.514 (元)

由此可见，选择不同的信用评分卡，不同的阈值组合，会给银行带来不同的收入与损失，由此决定银行最终收入。因此，银行的目标是 选择最 合理的信用评分卡组合以及其阈值，使得银行最终收入最多。

赛题说明 2：QUBO 模型简介

QUBO 模型是指二次无约束二值优化（Quadratic Unconstrained Binary Optimization）模型，它是一种用于解决组合优化问题的数学模型。在 QUBO 模型中，需要将问题转化为一个决策变量为二值变量，目标函数是一个二次函数形式优化模型。

QUBO 模型可以运行在量子计算机硬件上，通过量子计算机进行毫秒 级的加速求解。这种模型和加速方式在未来各行业中将得到广泛的实际应用。因此现阶段研究基于 QUBO 模型的量子专用算法十分有应用价值。例 如典型的图着色、旅行商问题、车辆路径优化问题等，都可以转化为 QUBO 模型并借助于量子计算机求解。

赛题说明 3：赛题数据

附件 1 中共包含 100 张信用评分卡，每张卡可设置 10 种阈值之一，并对应各自的通过率与坏账率共 200 列，其中 t_1 代表信用评分卡 1 的通过 率共 10 项，h_1 代表信用评分卡 1 的坏账率共 10 项，依次类推 t_100 代表 信用评分卡 100 的通过率，h_100 代表信用评分卡 100 的坏账率。

根据上面的赛题说明及附件 1 中的数据，请你们团队通过建立数学模 型完成如下问题 1 至问题 3。

问题 1：在 100 个信用评分卡中找出 1 张及其对应阈值，使最终收入 最多，请针对该问题进行建模，将该模型转为 QUBO 形式并求解。

问题 2：假设赛题说明 3 目前已经选定了数据集中给出的信用评分卡 1、 信用评分卡 2、信用评分卡 3 这三种规则，如何设置其对应的阈值，使最 终收入最多，请针对该问题进行建模，将模型转为 QUBO 形式并求解。

问题 3：从所给附录中 100 个信用评分卡中任选取 3 种信用评分卡， 并设置合理的阈值，使得最终收入最多，请针对该问题进行建模，并将模 型转为 QUBO 形式并求解。

3、问题 1 求解

3.1 求解目标的建模

$x_{i,j}$为一个二值变量，其表示为

$$x_{i,j}=\begin{cases}0, & \text{阈值被选中}\\ 1, & \text{阈值未被选中}\end{cases}$$

其中 i 代表信用评分卡的编号，取值范围为[1, 100]，j 代表信用评分卡中的阈值编号，取值范围为[1, 10]

由于信用评分卡的编号取值为 1-100 的整数，阈值的编号取值为 1-10 的整数，因此最终收入可以表示为所有选定阈值对应的收入的总和，表示为：

$$f=\sum_{i=1}^{100}\sum_{j=1}^{10}x_{i,j}\cdot f_{i,j}$$

式中，$x_{i,j}$为每个阈值的选择状态，$f_{i,j}$为每个阈值对应的收入

$f_{i,j}$可以采用下式计算：

$$f_{i,j}=r\cdot t_{i,j}-(1+r)\cdot t_{i,j}\cdot h_{i,j}$$

式中，r 为利息，$t_{i,j}$为阈值对应的通过率，$h_{i,j}$为阈值对应的坏账率。

问题 1 的求解目标是从 1000 个阈值中选出一个最优阈值，因此选择的阈值总和为 1。约束条件建模如下

$$\sum_{i=1}^{100}\sum_{j=1}^{10}x_{i,j}\ =\ 1$$

因此目标函数可以用下式表示

$$Min\ H_1(x)=-\left(\sum_{i=1}^{100}\sum_{j=1}^{10}x_{i,j}\cdot f_{i,j}\right)+P\cdot\left[\left(\sum_{j=1}^{100}\sum_{i=1}^{10}x_{i,j}\right)-1\right]^2$$

其中，P 表示惩罚指数，如果阈值选择的数量不为 1，则惩罚指数会使得目标函数的取值陡增。

将目标函数写成 QUBO 形式，即为下式

$$\text{QUBO: Minimize } H_1(x)=\boldsymbol{x}^{\mathrm{t}}\,\boldsymbol{Q}\,\boldsymbol{x}$$

$\boldsymbol{x}$ 为阈值选择状态的二值变量的向量，$\boldsymbol{Q}$ 是一个常量矩阵，$\boldsymbol{Q}$ 的详细构建过程见代码部分。

3.2 求解算法

问题 1 算法： Minimize $H_1(x) = \boldsymbol{x}^{\mathrm{t}}\, \boldsymbol{Q}\, \boldsymbol{x}$

***输入：data_100.csv**

***用 DataFrame 读取数据，并转化为矩阵格式，用于并行计算**

***创建 $\boldsymbol{Q}$ 矩阵**

首先创建一个 1000*1000 的 0 矩阵，然后将最终收入作为权重写入矩阵：

Q[(i, i)] = -(interest_rate * t - (1 + interest_rate) * t * h)

***在 $\boldsymbol{Q}$ 矩阵中添加约束惩罚**

为了确保只选择一个阈值，因此不同阈值之间的权重应该添加惩罚：

Q[(i, j)] += penalty

***生成模拟退火初始状态用来加快计算速度**

随机选取阈值编号 0-1000 中的一个整数，作为被选中的阈值，变量记为 1，作为求解器的初始解。

***使用量子退火模拟算法求解**

sampler = SimulatedAnnealingSampler()

***输出：$\boldsymbol{x}$**

num_reads 是量子计算机中一个重要的参数，指的是在一个量子计算机的执行过程中，重复进行多少次相同的计算。每次计算都会得到一个可能的解，这些解会被汇总在一起，最后可以根据这些解的出现频率来得到最终的解。在 Simulated Annealing Sampler 这种经典计算机的算法中，num_reads 表示执行的次数，每次执行都会得到一个不同的解。而在量子计算机的量子退火算法中，num_reads 也表示重复执行的次数，每次执行会得到一个可能的解。num_reads 的值越大，得到的解的质量可能会越好，但计算时间也会越长。因此，选择合适的 num_reads 对于解决问题是非常重要的。为了探究 num_reads 的取值与求解结果的关系，将 num_reads 取值为 10，100，200，300，400，500，1000，2000，并运行代码，记录最终收入与计算时间，得到的关系如图 1 所示，可以发现 num_reads 越大，求解结果

越接近绝对正确解，同时计算消耗线性增加。当 num_reads 取值约 500 时，可以兼顾计算精度与计算消耗。最优 num_reads 的取值可能与求解目标 $\boldsymbol{x}$ 的长度有关，当 $\boldsymbol{x}$ 的长度越大时，出现局部最优解的概率也越大，因此需要增大 num_reads 的取值来提高计算精度，在本问题中，为了保证计算精度，设置 num_reads = 1000。

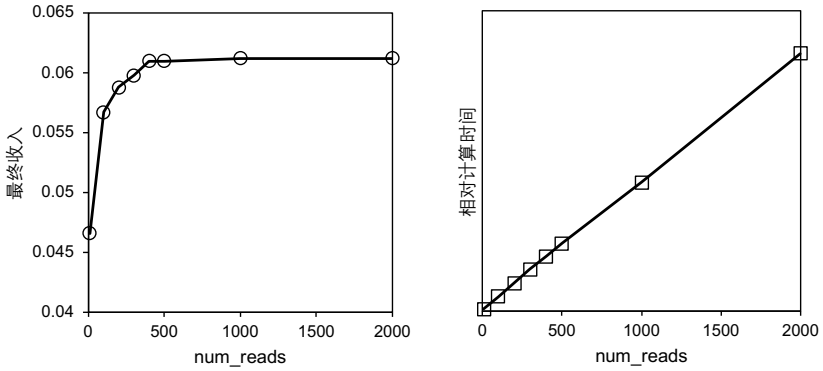

图 1　num_reads 取值与求解结果的关系

3.3 求解结果

在 python3.9 环境中使用上述算法进行求解，得到问题 1 的求解结果如下表所示：

问题 2 求解结果	
信用评分卡编号	**49**
阈值	**1**
通过率	0.82
坏账率	0.005
最终收入	**0.06117**

4、问题 2 求解

4.1 求解目标的建模

$x_{i,j}$为一个二值变量，其表示为

$$x_{i,j}=\begin{cases}0, & \text{阈值被选中}\\ 1, & \text{阈值未被选中}\end{cases}$$

其中 i 代表信用评分卡的编号，取值范围为$[1,3]$，j 代表信用评分卡中的阈值编号，取值范围为$[1,10]$

由于信用评分卡的编号取值为 1-3 的整数，阈值的编号取值为 1-10 的整数，因此最终收入可以表示为所有选定阈值对应的收入的总和，表示为：

$$f=\sum_{i=1}^{3}\sum_{j=1}^{10}x_{i,j}\cdot f_{i,j}$$

式中，$x_{i,j}$为每个阈值的选择状态，$f_{i,j}$为每个阈值对应的收入

$f_{i,j}$可以采用下式计算：

$$\begin{cases}f_{i,j}=r\cdot t_{i,j}-(1+r)\cdot t_{i,j}\cdot h_{i,j}\\ t_{i,j}=t_{1,j}\cdot t_{2,j}\cdot t_{3,j}\\ h_{i,j}=(h_{1,j}+h_{2,j}+h_{3,j})/3\end{cases}$$

式中，r 为利息，$t_{i,j}$为阈值对应的总通过率，$h_{i,j}$为阈值对应的总坏账率。

问题 2 的求解目标是分别从评分卡 1，评分卡 2，评分卡 3 中个阈值中选出一个最优阈值，因此选择的阈值总和为 3。因此问题 2 中的约束有两个：一是确保在所有阈值中选择的阈值数量为 3，二是确保每个评分卡只选择一个阈值。

约束条件建模如下

$$\begin{cases}\sum_{i=1}^{3}\sum_{j=1}^{10}x_{i,j}=3, & i=1,2,3\\ \sum_{j=1}^{10}x_{i,j}=1, & i=1,2,3\end{cases}$$

因此目标函数可以用下式表示

$$Min\ H_2(x) = -\left(\sum_{i=1}^{3}\sum_{j=1}^{10} x_{i,j}\cdot f_{i,j}\right) + P\cdot\left[\left(\sum_{i=1}^{3}\sum_{j=1}^{10} x_{i,j}\right) - 3\right]^2 + P\cdot\sum_{i=1}^{3}\left[\left(\sum_{i=1}^{10} x_{i,j}\right) - 1\right]^2$$

其中，P 表示惩罚指数，如果阈值选择的总数量不为 3，或者每个评分卡选择的阈值数量不为 1，则惩罚指数会使得目标函数的取值陡增。

将目标函数写成 QUBO 形式，即为下式

$$\text{QUBO: Minimize } H_2(x) = \boldsymbol{x}^{\mathrm{t}}\,\boldsymbol{Q}\,\boldsymbol{x}$$

$\boldsymbol{x}$ 为阈值选择状态的二值变量的向量，$\boldsymbol{Q}$ 是一个常量矩阵，$\boldsymbol{Q}$ 的详细构建过程见代码部分。

4.2 求解算法

问题 2 算法： Minimize $H_2(x) = \boldsymbol{x}^{\mathrm{t}}\,\boldsymbol{Q}\,\boldsymbol{x}$

***输入：data_100.csv**

***用 DataFrame 读取数据，只截取前 6 列，并转化为矩阵格式，用于并行计算**

***创建 $\boldsymbol{Q}$ 矩阵**

首先创建一个 30*30 的 0 矩阵，然后将最终收入作为权重写入矩阵：

Q[(i, j)] = -(interest_rate * t - (1 + interest_rate) * t * h)

***在 $\boldsymbol{Q}$ 矩阵中添加约束惩罚**

为了确保每个评分卡只选择一个阈值，因此同一个评分卡的不同阈值之间的权重应该添加惩罚

```
for i in range(3):
    for j in range(10):
        for k in range(j + 1, 10):
                Q[(i * 10 + j, i * 10 + k)] += penalty
```

***生成模拟退火初始状态用来加快计算速度**

分别从[0, 10], [10, 20], [20, 30]中随机选取一个阈值编号，作为被选中的阈值，变量记为 1，作为求解器的初始解。

***使用量子退火模拟算法求解**

```
sampler = SimulatedAnnealingSampler()
```

***输出：$\boldsymbol{x}$**

4.3 求解结果

在 python3.9 环境中使用上述算法进行求解，得到问题 2 的求解结果如下表所示：

问题 2 求解结果			
信用评分卡编号	**1**	**2**	**3**
阈值	**8**	**1**	**2**
通过率	0.93	0.72	0.82
坏账率	0.036	0.032	0.013
总通过率		0.5491	
总坏账率		0.0270	
最终收入		**0.0279**	

5、问题 3 求解

5.1 求解方法 1

5.1.1 求解目标的建模

$x_{i,j}$为一个二值变量，其表示为

$$x_{i,j}=\begin{cases}0, & \text{阈值被选中}\\ 1, & \text{阈值未被选中}\end{cases}$$

其中 i 代表信用评分卡的编号，取值范围为[1, 100]，j 代表信用评分卡中的阈值编号，取值范围为[1, 10]

由于信用评分卡的编号取值为 1-100 的整数，阈值的编号取值为 1-10 的整数，因此最终收入可以表示为所有选定阈值对应的收入的总和，表示为：

$$f=\sum_{i=1}^{100}\sum_{j=1}^{10}x_{i,j}\cdot f_{i,j}$$

式中，$x_{i,j}$为每个阈值的选择状态，$f_{i,j}$为每个阈值对应的收入

$f_{i,j}$可以采用下式计算：

$$\begin{cases}f_{i,j}=r\cdot t_{i,j}-(1+r)\cdot t_{i,j}\cdot h_{i,j}\\ t_{i,j}=t_{1,j}\cdot t_{2,j}\cdot t_{3,j}\\ h_{i,j}=(h_{1,j}+h_{2,j}+h_{3,j})/3\end{cases}$$

式中，r 为利息，$t_{i,j}$为阈值对应的总通过率，$h_{i,j}$为阈值对应的总坏账率。

问题 3 的求解目标是从所给附录中 100 个信用评分卡中任选取 3 种信用评分卡，并设置合理的阈值，使得最终收入最多。因此选择的阈值总和为 3。因此问题 2 中的约束有两个：一是确保在所有阈值中选择的阈值数量为 3，二是确保每个评分卡只选择一个阈值。

约束条件建模如下

$$\begin{cases}\sum_{i=1}^{100}\sum_{j=1}^{10}x_{i,j}=3, & i=1,2,3,\ldots,100\\ \sum_{j=1}^{10}x_{i,j}=1, & i=1,2,3,\ldots,100\end{cases}$$

因此目标函数可以用下式表示

$$Min\ H_3(x) = -\left(\sum_{i=1}^{100}\sum_{j=1}^{10} x_{i,j}\cdot f_{i,j}\right) + P\cdot\left[\left(\sum_{i=1}^{100}\sum_{j=1}^{10} x_{i,j}\right) - 3\right]^2 + P\cdot\sum_{i=1}^{100}\left[\left(\sum_{i=1}^{10} x_{i,j}\right) - 1\right]^2$$

其中，P 表示惩罚指数，如果阈值选择的总数量不为 3，或者每个评分卡选择的阈值数量不为 1，则惩罚指数会使得目标函数的取值陡增。

将目标函数写成 QUBO 形式，即为下式

$$\text{QUBO: Minimize } H_3(x) = \boldsymbol{x}^{\mathrm{t}}\,\boldsymbol{Q}\,\boldsymbol{x}$$

$\boldsymbol{x}$ 为阈值选择状态的二值变量的向量，$\boldsymbol{Q}$ 是一个常量矩阵，$\boldsymbol{Q}$ 的详细构建过程见代码部分。

5.1.2 求解算法

问题 3 方法 1 算法： Minimize $H_3(x) = \boldsymbol{x}^{\mathrm{t}}\,\boldsymbol{Q}\,\boldsymbol{x}$

***输入：data_100.csv**

***用 DataFrame 读取数据，并转化为矩阵格式，用于并行计算**

***创建 $\boldsymbol{Q}$ 矩阵**

首先创建一个 1000*1000 的 0 矩阵，然后将最终收入作为权重写入矩阵：

Q[(i, i)] = -(interest_rate * t - (1 + interest_rate) * t * h)

***在 $\boldsymbol{Q}$ 矩阵中添加约束惩罚**

为了确保每个评分卡只选择一个阈值，因此同一个评分卡的不同阈值之间的权重应该添加惩罚

Q[(i, j)] += penalty

***生成模拟退火初始状态用来加快计算速度**

从[0, 10], [10, 20], …, [990, 1000]的区间中任选 3 个区间，并分别从 3 个区间中随机选择一个阈值编号，作为被选中的阈值，变量记为 1，作为求解器的初始解。

***使用量子退火模拟算法求解**

sampler = SimulatedAnnealingSampler()

***输出：$\boldsymbol{x}$**

5.1.3 求解结果

在 python3.9 环境中使用上述算法进行求解，得到问题 3 的求解结果如下表所示：

问题 3 求解结果			
信用评分卡编号	**8**	**33**	**49**
阈值	**2**	**6**	**3**
通过率	0.88	0.93	0.85
坏账率	0.013	0.024	0.01
总通过率		0.6957	
总坏账率		0.0157	
最终收入		**0.04383**	

5.2 求解方法 2

5.2.1 求解目标的建模

从 100 个信用评分卡中任选 3 个信用评分卡，并选择合理的阈值，其形成的组合数量 m 为：

$$m = \mathrm{C}_{100}^{3} \cdot 10 \cdot 10 \cdot 10 \ = \ 161700000$$

只需要在这 m 的组合中选出一个最优组合，使得最终收入最大即可，这就将问题转化为与问题 1 类似的问题。具体的建模过程简述如下（许多参数可以与问题 1 中的参数共用，在这里不再赘述）：

最终收入可以表示为所有选定组合对应的收入的总和，表示为：

$$f = \sum_{i=1}^{m} x_i \cdot f_i$$

式中，x_i为每个组合的选择状态，f_i为每个组合对应的最终收入

f_i可以采用下式计算：

$$f_i = r \cdot t_i - (1 + r) \cdot t_i \cdot h_i$$

式中，r 为利息，t_i为组合对应的总通过率，h_i为组合对应的总坏账率。

因此求解目标是从 m 个组合中选出一个最优组合，因此选择的组合总和为 1。约束条件建模如下

$$\sum_{i=1}^{m} x_i \ = \ 1$$

因此目标函数可以用下式表示

$$Min\ H_3(x) = -\left(\sum_{i=1}^{m} x_i \cdot f_i\right) + P \cdot \left[\left(\sum_{i=1}^{m} x_i\right) - 1\right]^2$$

其中，P 表示惩罚指数，如果组合选择的数量不为 1，则惩罚指数会使得目标函数的取值陡增。

将目标函数写成 QUBO 形式，即为下式

$$\text{QUBO: Minimize } H_3(x) = \boldsymbol{x}^{\mathrm{t}}\, \boldsymbol{Q}\, \boldsymbol{x}$$

$\boldsymbol{x}$ 为组合选择状态的二值变量的向量，$\boldsymbol{Q}$ 是一个常量矩阵，$\boldsymbol{Q}$ 的详细构建过程见代码部分。

5.2.2 求解算法

问题 3 方法 2 算法：Minimize $H_1(x) = \boldsymbol{x}^{\mathrm{t}}\, \boldsymbol{Q}\, \boldsymbol{x}$

***输入：data_100.csv**

***用 DataFrame 读取数据，并转化为矩阵格式，用于并行计算**

***创建 $\boldsymbol{Q}$ 矩阵**

首先计算所有组合创建一个 161700000*161700000 的 0 矩阵，然后将每个组合的最终收入作为权重写入矩阵：

Q[(i, i)] = -(interest_rate * t - (1 + interest_rate) * t * h)

储存每个组合索引与评分信用卡的映射关系

***在 $\boldsymbol{Q}$ 矩阵中添加约束惩罚**

为了确保只选择一个组合，因此不同组合之间的权重应该添加惩罚：

Q[(i, j)] += penalty

***生成模拟退火初始状态用来加快计算速度**

随机选取组合编号 1-161700000 中的一个整数，作为被选中的组合，变量记为 1，作为求解器的初始解。

***使用量子退火模拟算法求解**

sampler = SimulatedAnnealingSampler()

***根据求解得到的组合索引值，利用映射关系得到信用评分卡与阈值**

***输出：信用评分卡与阈值的组合**

5.2.3 求解结果

本算法是为了提供一种求解思路，由于计算量太大，求解设备无法在短时间内得到最优解。

5.3 求解方法 3（方法 2 的改进）

5.3.1 求解目标的建模

在方法 2 中，由于组合的数量过于庞大，因此难以求解。在本方法中，希望寻找一种方法来减少数据体积。

由于最优三组合信用评分卡的编号与最优二组合信用评分卡的编号具有一定的相关性，因此可以先寻找排名靠前的 n 个最优的信用评分卡阈值二组合，并得到这些组合中的信用评分卡与阈值编号，然后再从这些阈值中寻找最优三组合，使得最终收入最大。具体建模过程如下：

第一步：

从 100 个信用评分卡中任选取 2 种信用评分卡，并设置合理的阈值，使得最终收入最多。因此选择的阈值总和为 2。因此此处的约束有两个：一是确保在所有阈值中选择的阈值数量为 2，二是确保每个评分卡只选择一个阈值。

约束条件建模如下

$$\begin{cases} \sum_{i=1}^{100}\sum_{j=1}^{10} x_{i,j} = 2, & i = 1,2,3,\ldots,100 \\ \sum_{j=1}^{10} x_{i,j} = 1, & i = 1,2,3,\ldots,100 \end{cases}$$

因此目标函数可以用下式表示

$$Min\ H_3(x) = -\left(\sum_{i=1}^{100}\sum_{j=1}^{10} x_{i,j}\cdot f_{i,j}\right) + P\cdot\left[\left(\sum_{i=1}^{100}\sum_{j=1}^{10} x_{i,j}\right) - 2\right]^2 + P\cdot\sum_{i=1}^{100}\left[\left(\sum_{i=1}^{10} x_{i,j}\right) - 1\right]^2$$

其中，P 表示惩罚指数，如果阈值选择的总数量不为 3，或者每个评分卡选择的阈值数量不为 1，则惩罚指数会使得目标函数的取值陡增。

将目标函数写成 QUBO 形式，即为下式

$$\text{QUBO: Minimize } H_3(x) = \boldsymbol{x}^{\mathrm{t}}\,\boldsymbol{Q}\,\boldsymbol{x}$$

$\boldsymbol{x}$ 为阈值选择状态的二值变量的向量，$\boldsymbol{Q}$ 是一个常量矩阵，$\boldsymbol{Q}$ 的详细构建过程见代码部分。**在这里需要指出的是，使用模拟退火方法求解 QUBO 时，得到的前 10 个最优解不一定是绝对的最优解，因为模拟退火方法是一种梯度优化算法，求得的 10 个最优解与求解过程中的梯度下降路径有关，但是梯度下降的幅度将随着计算过程越来越小，因此，在**

本方法中，将对目标 QUBO 函数使用模拟退火算法求解 10 次，每次获取前 5 个最优解，一共得到 50 个阈值组合，并期望得到其中的 100 个阈值，由于每次求解得到的结果可能重复，因此最终筛选得到的阈值数量可能小于 100 个，将筛选出的阈值数量记为 *n*。

第二步：

从 n 个阈值中任选 3 个形成组合，其形成的组合数量 m 为：

$$m = \mathrm{C}_n^3$$

只需要在这 m 的组合中选出一个最优组合，使得最终收入最大即可，这就将问题转化为与问题 1 类似的问题。具体的建模过程简述如下（许多参数可以与问题 1 中的参数共用，在这里不再赘述）：

最终收入可以表示为所有选定组合对应的收入的总和，表示为：

$$f = \sum_{i=1}^{m} x_i \cdot f_i$$

式中，x_i为每个组合的选择状态，f_i为每个组合对应的最终收入

f_i可以采用下式计算：

$$f_i = r \cdot t_i - (1 + r) \cdot t_i \cdot h_i$$

式中，r 为利息，t_i为组合对应的总通过率，h_i为组合对应的总坏账率。

因此求解目标是从 m 个组合中选出一个最优组合，因此选择的组合总和为 1。约束条件建模如下

$$\sum_{i=1}^{m} x_i \ = \ 1$$

因此目标函数可以用下式表示

$$Min\ H_3(x) = -\left(\sum_{i=1}^{m} x_i \cdot f_i\right) + P \cdot \left[\left(\sum_{i=1}^{m} x_i\right) - 1\right]^2$$

其中，P 表示惩罚指数，如果组合选择的数量不为 1，则惩罚指数会使得目标函数的取值陡增。

将目标函数写成 QUBO 形式，即为下式

$$\text{QUBO: Minimize } H_3(x) = \boldsymbol{x}^{\mathrm{t}}\, \boldsymbol{Q}\, \boldsymbol{x}$$

x 为组合选择状态的二值变量的向量，Q 是一个常量矩阵，Q 的详细构建过程见代码部分。

5.2.2 求解算法

问题 3 方法 3 算法（1）：

***输入：data_100.csv**

***用 DataFrame 读取数据，并转化为矩阵格式，用于并行计算**

***创建 Q 矩阵**

首先计算所有组合创建一个 1000*1000 的 0 矩阵，然后将每个组合的最终收入作为权重写入矩阵：

Q[(i, i)] = -(interest_rate * t - (1 + interest_rate) * t * h)

储存每个组合索引与评分信用卡的映射关系

***在 Q 矩阵中添加约束惩罚**

为了确保只选择一个组合，因此不同组合之间的权重应该添加惩罚：

Q[(i, j)] += penalty

***生成模拟退火初始状态用来加快计算速度**

从[0, 10], [10, 20], …, [990, 1000]的区间中任选 2 个区间，并分别从 2 个区间中随机选择一个阈值编号，作为被选中的阈值，变量记为 1，作为求解器的初始解。

***使用量子退火模拟算法求解**

sampler = SimulatedAnnealingSampler()

***求解得到前 5 个最优的阈值二组合**

***重复 10 次求解一共得到 50 个最优的阈值二组合**

***输出：筛选出 n 个最优阈值**

问题 3 方法 2 算法（2）：

***输入：筛选出的 n 个最优阈值**

***创建 *Q* 矩阵**

首先计算所有组合创建一个 *m*m* 的 0 矩阵，然后将每个组合的最终收入作为权重写入矩阵：

Q[(i, i)] = -(interest_rate * t - (1 + interest_rate) * t * h)

储存每个组合索引与阈值的映射关系

***在 *Q* 矩阵中添加约束惩罚**

为了确保只选择一个组合，因此不同组合之间的权重应该添加惩罚：

Q[(i, j)] += penalty

***使用量子退火模拟算法求解**

sampler = SimulatedAnnealingSampler()

***根据求解得到的组合索引值，利用映射关系得到信用评分卡与阈值**

***输出：信用评分卡与阈值的组合**

第一步的部分输出如下所示：

组合 1: 从信用评分卡 8 中选择阈值 2，通过率为 0.8800，坏账率为 0.0130。从信用评分卡 54 中选择阈值 5，通过率为 0.8500，坏账率为 0.0130。收入为 0.049338080000000006

组合 2: 从信用评分卡 8 中选择阈值 3，通过率为 0.8900，坏账率为 0.0160。从信用评分卡 44 中选择阈值 3，通过率为 0.8200，坏账率为 0.0110。收入为 0.047743516

组合 3: 从信用评分卡 49 中选择阈值 3，通过率为 0.8500，坏账率为 0.0100。从信用评分卡 56 中选择阈值 5，通过率为 0.8400，坏账率为 0.0140。收入为 0.047866559999999996

组合 4: 从信用评分卡 4 中选择阈值 2，通过率为 0.8000，坏账率为 0.0070。从信用评分卡 19 中选择阈值 4，通过率为 0.8600，坏账率为 0.0160。收入为 0.04649504

组合 5: 从信用评分卡 54 中选择阈值 1，通过率为 0.7500，坏账率为 0.0030。从信用评分卡 73 中选择阈值 2，通过率为 0.8400，坏账率为 0.0100。收入为 0.0459774

组合 1: 从信用评分卡 33 中选择阈值 3，通过率为 0.8900，坏账率为 0.0190。从信用评分卡 73 中选择阈值 2，通过率为 0.8400，坏账率为 0.0100。收入为 0.048100583999999995

组合 2: 从信用评分卡 19 中选择阈值 4，通过率为 0.8600，坏账率为 0.0160。从信用评分卡 73 中选择阈值 2，通过率为 0.8400，坏账率为 0.0100。收入为 0.047649503999999995

组合 3: 从信用评分卡 8 中选择阈值 2，通过率为 0.8800，坏账率为 0.0130。从信用评分卡 61 中选择阈值 5，通过率为 0.8700，坏账率为 0.0200。收入为 0.047605008

组合 4: 从信用评分卡 49 中选择阈值 1，通过率为 0.8200，坏账率为 0.0050。从信用评分卡 81 中选择阈值 2，通过率为 0.7900，坏账率为 0.0080。收入为 0.047276444

组合 5: 从信用评分卡 33 中选择阈值 6，通过率为 0.9300，坏账率为 0.0240。从信用评分卡 68 中选择阈值 4，通过率为 0.8500，坏账率为 0.0160。收入为 0.0461652

第一步重复运算 10 次后，将结果整理并筛选出 38 个阈值，如下表

评分卡	3	4	4	4	8	8	8	19	19	19
阈值	2	1	2	4	2	3	4	3	4	5
评分卡	19	33	33	39	41	44	49	49	49	54
阈值	6	3	6	2	1	5	1	2	3	1
评分卡	54	54	56	59	59	61	61	68	68	68
阈值	2	5	5	4	9	2	5	1	2	3
评分卡	68	68	73	73	73	81	91	91		
阈值	6	4	1	2	3	2	2	4		

再从 38 个阈值组成的三阈值组合中，使用第二步算法求解得到最优组合，使得最终收入最大。**由于方法 3 中经历了数据筛选，并基于已筛选的数据进行求解，因此得到的解可能是一个近似解。**

在 python3.9 环境中使用上述算法进行求解，得到问题 3 的求解结果如下表所示：

问题 3 方法 3 求解结果			
信用评分卡编号	**8**	**33**	**49**
阈值	**2**	**6**	**3**
通过率	0.88	0.93	0.85
坏账率	0.013	0.024	0.01
总通过率		0.6957	
总坏账率		0.0157	
最终收入		**0.04383**	

参考文献

[1] Glover, F., Kochenberger, G., Hennig, R., & Du, Y. (2022). Quantum bridge analytics I: a tutorial on formulating and using QUBO models. Annals of Operations Research, 314(1), 141-183.

[2] Morstyn, T. (2022). Annealing-based Quantum Computing for Combinatorial Optimal Power Flow. IEEE Transactions on Smart Grid.

[3] Papalitsas, C., Andronikos, T., Giannakis, K., Theocharopoulou, G., & Fanarioti, S. (2019). A QUBO model for the traveling salesman problem with time windows. Algorithms, 12(11), 224.

附录一（问题 1 求解代码）

运行环境为：python 3.9

```
import numpy as np
from neal import SimulatedAnnealingSampler
import pandas as pd
import time
import random

# 载入数据
df = pd.read_csv('./附件 1：data_100.csv')
data = df.values
# 定义参数
num_credit_cards = 100
num_thresholds = 10
interest_rate = 0.08  # 利息
penalty = 100000  # 罚函数系数
start_time = time.perf_counter()
# 初始化 QUBO 字典
Q = {(i, j): 0 for i in range(1000) for j in range(1000)}

# 定义目标函数和约束条件
for card in range(num_credit_cards):
    for threshold in range(num_thresholds):
        i = card * num_thresholds + threshold
        t, h = data[threshold, 2 * card], data[threshold, 2 * card + 1]
        Q[(i, i)] = -(interest_rate * t - (1 + interest_rate) * t * h)

# 添加罚函数以确保只选择一个阈值
for i in range(1000):
    for j in range(i + 1, 1000):
        Q[(i, j)] += penalty


# 使用 neal 库的 SimulatedAnnealingSampler 来解决 QUBO 问题
init = {i: 0 for i in range(1000)}
random_indices = random.sample(range(100), 1)
for i in range(1):
    init[random_indices[i] * 10] = 1


sampler = SimulatedAnnealingSampler()
response = sampler.sample_qubo(Q, num_reads=2000,initial_state = init)
```

```
# response = sampler.sample_qubo(Q)
# 获取最优解
best_solution = response.first.sample

# 计算运行时间
end_time = time.perf_counter()
run_time = (end_time - start_time) * 1000

# 输出最优解
print("Best solution:", best_solution)

# 输出结果
selected_thresholds = [i for i, value in best_solution.items() if value == 1]
for threshold in selected_thresholds:
    card = threshold // 10
    idx = threshold % 10
    print(f"从信用评分卡： {card + 1} 中选择阈值 ： {idx+1}，通过率为 {data[idx, 2 *
card]:.4f}， "
          f"坏账率为 {data[idx, 2 * card + 1]:.4f},"
          f"总收入为 {(interest_rate * data[idx, 2 * card] - (1 + interest_rate)
* data[idx, 2 * card] * data[idx, 2 * card + 1]):.4f}")
print("程序运行时间为: {:.2f}ms".format(run_time))
```

附录二（问题 2 求解代码）

运行环境为：python 3.9

```
    import pandas as pd
import numpy as np
from neal import SimulatedAnnealingSampler
from dimod import BinaryQuadraticModel
from itertools import product
from itertools import combinations
import itertools
import time
import random

# 载入数据
df = pd.read_csv('./附件 1：data_100.csv')
data = df.iloc[:, :6].values

# 定义利息
interest_rate = 0.08
penalty = 100000
def final_income(t1, h1, t2, h2, t3, h3):
    total_pass_rate = t1 * t2 * t3
    total_bad_rate = (h1 + h2 + h3) / 3
    return interest_rate * total_pass_rate - (1 + interest_rate) *
total_pass_rate * total_bad_rate

start_time = time.perf_counter()

# 构建 QUBO
Q = {(i, j): 0 for i in range(30) for j in range(30)}

for i, j in product(range(30), repeat=2):
    card_i = i // 10
    card_j = j // 10

    if i == j:
        t = data[i % 10, 2 * card_i]
        h = data[i % 10, 2 * card_i + 1]
        Q[(i, j)] = -final_income(t, h, t, h, t, h)
    elif card_i != card_j:
        card1_i, card1_j = i % 10, j % 10
        t1, h1 = data[card1_i, 2 * card_i], data[card1_i, 2 * card_i + 1]
        t2, h2 = data[card1_j, 2 * card_j], data[card1_j, 2 * card_j + 1]
```

```
        for k in range(10):
            card3 = 3 - card_i - card_j
            t3, h3 = data[k, 2 * card3], data[k, 2 * card3 + 1]
            Q[(i, j)] = -final_income(t1, h1, t2, h2, t3, h3)

# 添加约束以确保每个信用评分卡只选择一个阈值
for i in range(3):
    for j in range(10):
        for k in range(j + 1, 10):
            Q[(i * 10 + j, i * 10 + k)] += penalty

bqm = BinaryQuadraticModel.from_qubo(Q, offset=0.0)

# 求解 QUBO 问题

init = {i: 0 for i in range(30)}
random_indices = random.sample(range(3), 3)
for i in range(3):
    init[random_indices[i] * 10] = 1

sampler = SimulatedAnnealingSampler()
response = sampler.sample(bqm, num_reads=200,initial_state = init)
# response = sampler.sample(bqm, num_reads=300)

# 获取最优解
best_solution = response.first.sample


# 计算运行时间
end_time = time.perf_counter()
run_time = (end_time - start_time) * 1000

# 输出结果
selected_thresholds = [i for i, value in best_solution.items() if value == 1]
for threshold in selected_thresholds:
    card = threshold // 10
    idx = threshold % 10
    print(f"从信用评分卡： {card + 1} 中选择阈值 ： {idx+1}，通过率为 {data[idx, 2 *
card]:.4f}，坏账率为 {data[idx, 2 * card + 1]:.4f}")

sample = best_solution
# 计算总通过率和总坏账率
total_pass_rate = 1
```

```
total_bad_debt_rate = 0

for i, value in sample.items():
    if value == 1:
        card_idx = i // 10
        threshold_idx = i % 10
        total_pass_rate *= data[threshold_idx, 2 * card_idx]
        total_bad_debt_rate += data[threshold_idx, 2 * card_idx + 1]

total_bad_debt_rate /= 3

# 计算最终收入
final_income = interest_rate * total_pass_rate - (1 + interest_rate) *
total_pass_rate * total_bad_debt_rate
print("最终收入:", final_income)
print("程序运行时间为: {:.2f}ms".format(run_time))
```

附录三（问题 3 方法 1 求解代码）

运行环境为：python 3.9

```
import pandas as pd
import numpy as np
from neal import SimulatedAnnealingSampler
from dimod import BinaryQuadraticModel
import itertools
import time
import random

# 载入数据
df = pd.read_csv('./附件 1：data_100.csv')
data = df.values

# 定义利息和罚款项
interest_rate = 0.08
penalty = 100000

# 定义目标函数和约束条件
def objective_function(x):
    # 计算最终收入
    total_pass_rate = 1
    total_bad_debt_rate = 0
    for i, value in enumerate(x):
        if value == 1:
            card_idx = i // 10
            threshold_idx = i % 10
            total_pass_rate *= data[threshold_idx, 2 * card_idx]
            total_bad_debt_rate += data[threshold_idx, 2 * card_idx + 1]
    total_bad_debt_rate /= 3
    final_income = interest_rate * total_pass_rate - (1 + interest_rate) *
total_pass_rate * total_bad_debt_rate
    return final_income


start_time = time.perf_counter()

# 构建 QUBO 矩阵
Q = {(i, j): 0 for i in range(1000) for j in range(1000)}
print("程序运行时间为：{:.2f}ms".format(time.perf_counter()-start_time))

# 计算目标函数
```

```
for i, j in itertools.combinations(range(1000), 2):
    if i // 10 != j // 10:
        t1, h1 = data[i % 10, 2 * (i // 10)], data[i % 10, 2 * (i // 10) + 1]
        t2, h2 = data[j % 10, 2 * (j // 10)], data[j % 10, 2 * (j // 10) + 1]
        for k in range(1000):
            if k != i and k != j and k // 10 != i // 10 and k // 10 != j // 10:
                t3, h3 = data[k % 10, 2 * (k // 10)], data[k % 10, 2 * (k // 10)
+ 1]
                w = interest_rate * t1 * t2 * t3 - (1 + interest_rate) * (t1 *
h1 + t2 * h2 + t3 * h3) / 3
                Q[(i, j)] -= w
                Q[(j, i)] -= w

# 添加约束条件
for i in range(1000):
    Q[(i, i)] -= penalty
    for j in range(i + 1, 1000):
        if i // 10 == j // 10:
            Q[(i, j)] += penalty
        else:
            Q[(i, j)] += 2 * penalty

bqm = BinaryQuadraticModel.from_qubo(Q, offset=0.0)
for i in range(10):
    bqm.add_linear_constraint([(i * 10 + j, 1) for j in range(10)],
constant=3)

# 求解 QUBO 问题

init = {i: 0 for i in range(1000)}
random_indices = random.sample(range(100), 3)
for i in range(3):
    init[random_indices[i] * 10] = 1

sampler = SimulatedAnnealingSampler()
response = sampler.sample(bqm, num_reads=300 ,initial_state = init)

end_time = time.perf_counter()
run_time = (end_time - start_time) * 1000

# 获取最优解
best_solution = response.first.sample
selected_thresholds = [i for i, value in best_solution.items() if value == 1]
```

```
# 输出结果
for threshold in selected_thresholds:
    card = threshold // 10
    idx = threshold % 10
    print(f"从信用评分卡： {card + 1} 中选择阈值 ： {idx+1}，通过率为 {data[idx, 2 *
card]:.4f}，坏账率为 {data[idx, 2 * card + 1]:.4f}")
print("最终收入:", objective_function(best_solution))
print("程序运行时间为: {:.2f}ms".format(run_time))
```

附录四（问题 3 方法 3 求解代码）

第一步

运行环境为：python 3.9

```
import pandas as pd
import numpy as np
from itertools import combinations
from dimod import BinaryQuadraticModel
from neal import SimulatedAnnealingSampler
import random

# 载入数据
df = pd.read_csv('./附件 1：data_100.csv')
data = df.values

def final_income(t1, h1, t2, h2):
    interest_rate = 0.08
    total_pass_rate = t1 * t2
    total_bad_rate = (h1 + h2) / 2
    return interest_rate * total_pass_rate - (1 + interest_rate) *
total_pass_rate * total_bad_rate

# 创建 Q 矩阵
num_credit_cards = 100
num_thresholds = 10
penalty = 100000  # 罚函数系数
Q = {(i, j): 0 for i in range(1000) for j in range(1000)}

for i, j in combinations(range(num_credit_cards * num_thresholds), 2):
    if i // num_thresholds != j // num_thresholds:
        t1, h1 = data[i % num_thresholds, 2 * (i // num_thresholds)], data[i %
num_thresholds, 2 * (i // num_thresholds) + 1]
        t2, h2 = data[j % num_thresholds, 2 * (j // num_thresholds)], data[j %
num_thresholds, 2 * (j // num_thresholds) + 1]
        Q[(i, j)] = -final_income(t1, h1, t2, h2)

for i in range(100):
    for j in range(10):
        for k in range(j + 1, 10):
            Q[(i * 10 + j, i * 10 + k)] += penalty

for i in range(1000):
    Q[(i, i)] -= 2 * penalty
```

```
    for j in range(i + 1, 1000):
        Q[(i, j)] += penalty
        Q[(j, i)] += penalty
# 使用 SimulatedAnnealingSampler 求解 QUBO 问题

init = {i: 0 for i in range(1000)}
random_indices = random.sample(range(100), 2)
for i in range(2):
    init[random_indices[i] * 10] = 1

bqm = BinaryQuadraticModel.from_qubo(Q)
sampler = SimulatedAnnealingSampler()
sampleset = sampler.sample(bqm, num_reads=2000 ,initial_state = init)

# 输出结果
top_solutions = sorted(sampleset.data(), key=lambda x: x.energy)[:10]
print(top_solutions)
for idx, solution in enumerate(top_solutions):
    sample = solution.sample
    selected_indices = [i for i, value in sample.items() if value == 1]
    t1, h1 = data[selected_indices[0] % num_thresholds, 2 *
(selected_indices[0] // num_thresholds)], data[selected_indices[0] %
num_thresholds, 2 * (selected_indices[0] // num_thresholds) + 1]
    t2, h2 = data[selected_indices[1] % num_thresholds, 2 *
(selected_indices[1] // num_thresholds)], data[selected_indices[1] %
num_thresholds, 2 * (selected_indices[1] // num_thresholds) + 1]
    print(f"组合 {idx + 1}: 从信用评分卡 {selected_indices[0] // num_thresholds +
1} 中选择阈值 {selected_indices[0] % num_thresholds + 1}，通过率为 {t1:.4f}，坏账率
为 {h1:.4f}。从信用评分卡 {selected_indices[1] // num_thresholds + 1} 中选择阈值
{selected_indices[1] % num_thresholds + 1}，通过率为 {t2:.4f}，坏账率为
{h2:.4f}。收入为 {final_income(t1, h1, t2, h2)}")
```

第二步

运行环境为：python 3.9

```
import numpy as np
from neal import SimulatedAnnealingSampler
import pandas as pd

# 载入数据
df = pd.read_csv('./selected_t.csv')
data = df.values
# 定义参数

num_thresholds = 38
num_combinations = 8436
interest_rate = 0.08  # 利息
penalty = 100000  # 罚函数系数


# 定义目标函数
def final_income(t1, h1, t2, h2, t3, h3):
    total_pass_rate = t1 * t2 * t3
    total_bad_debt_rate = (h1 + h2 + h3) / 3
    final_income = interest_rate * total_pass_rate - (1 + interest_rate) *
total_pass_rate * total_bad_debt_rate
    return final_income

# 初始化 QUBO 字典
Q = {(i, j): 0 for i in range(num_combinations) for j in
range(num_combinations)}
combinations_list = list(combinations(range(num_thresholds), 3))

for i in range(num_combinations):
    for j in range(i, num_combinations):
        t1, h1 = data[combinations_list[i][0], 0],
data[combinations_list[i][0], 1]
        t2, h2 = data[combinations_list[i][1], 2],
data[combinations_list[i][1], 3]
        t3, h3 = data[combinations_list[i][2], 4],
data[combinations_list[i][2], 5]
        income_j = final_income(t1, h1, t2, h2, t3, h3)
        Q[(i, j)] = -income_i

# 添加罚函数以确保只选择一个阈值
for i in range(num_combinations):
```

```
    for j in range(i + 1, num_combinations):
        Q[(i, j)] += penalty

# 使用 neal 库的 SimulatedAnnealingSampler 来解决 QUBO 问题

sampler = SimulatedAnnealingSampler()
response = sampler.sample_qubo(Q, num_reads=2000)

# 获取最优解
best_solution = response.first.sample

# 输出最优解
print("Best solution:", best_solution)

# 输出结果
top_solution = sampleset.first.sample
selected_combination = [i for i, value in top_solution.items() if value ==
1][0]

selected_indices = combinations_list[selected_combination]
t1, h1 = data[selected_indices[0], 0], data[selected_indices[0], 1]
t2, h2 = data[selected_indices[1], 2], data[selected_indices[1], 3]
t3, h3 = data[selected_indices[2], 4], data[selected_indices[2], 5]

print(f"最优组合：选择阈值 {selected_indices[0] + 1}，通过率为 {t1:.4f}，坏账率为
{h1:.4f}")
print(f"最优组合：选择阈值 {selected_indices[1] + 1}，通过率为 {t2:.4f}，坏账率为
{h2:.4f}")
print(f"最优组合：选择阈值 {selected_indices[2] + 1}，通过率为 {t3:.4f}，坏账率为
{h3:.4f}")

income_optimal = final_income(t1, h1, t2, h2, t3, h3)
print(f"最终收入：{income_optimal:.4f}")
```