队伍编号	MC2305153
题号	A

基于模拟退火算法的信用评分卡组合优化 QUBO 模型

摘 要

银行在信用卡或贷款业务中，需要对信用评分卡及阈值进行组合，从而使收入最大。本文以最终收入为目标函数、信用评分卡及阈值的选择为变量来建立数学模型，并将其转化为 QUBO 形式进行求解，得到使银行收入最大的最优组合方式。

针对问题一，100 张信用卡中选择一张并确定其阈值，我们对是否选择信用评分卡和对应阈值，分别引入 0-1 变量，并将约束条件转变为哈密顿运算符，构建目标函数对应的 QUBO 模型，并采用模拟退火法和 Python 工具进行求解，得到使得目标函数值最小的变量 D 矩阵，从而得到的最优组合，即信用评分卡 48 及其阈值设置为 1。

针对问题二，对给定的 3 张信用评分卡设置阈值，我们三张卡阈值的选择引入 0-1 变量，同样将约束条件转变为哈密顿运算符，得到目标函数，但此时得到的目标函数为四次函数，我们通过对三张信用评分卡的阈值组合进行枚举，引入枚举组合选择的 0-1 变量，完成函数的降维，使其转换为 QUBO 形式，最后利用模拟退火法和 Python 工具进行求解，得到使目标函数值最小的矩阵 D，从而得到的最优组合，即信用评分卡 1 的阈值设置为 5，信用评分卡 2 的阈值设置为 9，信用评分卡 3 的阈值设置为 10。

针对问题三，从 100 张信用评分卡中选择 3 张并分别设置阈值，通过对 3 种信用评分卡组合进行枚举，引入枚举组合选择的 01 变量，并利用问题二的阈值枚举组合 01 变量，构建出任意信用卡组合和阈值组合下的目标函数，构建目标函数对应的 QUBO 模型，并采用模拟退火法和 Python 工具进行求解，由于变量数量级过大，计算过程中需耗费大量算力，根据每一个信用评分卡与阈值组合对应的最终收入值离散图，我们发现收益率高的分布都在于阈值比较低的位置，为此我们做出一个合理的假设，即任何评分卡，都在 0-2 阈值之间取得较优解，如此得到最优组合为信用评分卡 7 及其阈值为 1，信用评分卡 32 及其阈值为 2，信用评分卡 48 及其阈值为 2。

本题目的难点在于约束条件的哈密顿算符构建，整体函数模型的构建以及 QUBO 模型函数二次形式的转换。本文的创新之处包括引入枚举组合的 01 变量对函数进行降维，实现高次函数和其对应 QUBO 形式的转换。

关键词：最优组合；QUBO；模拟退火法；0-1 变量；哈密顿运算符

目录

一、 问题重述

1.1 问题背景

银行在信用卡或贷款业务中需要对客户进行信用等级评定，这个过程实际上是根据一重或多重审核规则对客户进行打分，这些规则被称为信用评分卡。每个评分卡都有多个阈值设置，但只能有一个阈值生效。因此，在不同的阈值下，不同的信用评分卡对应着不同的通过率和坏账率。通常来说，通过率越高，坏账率也越高，反之亦然。银行在设置合理的信用评分卡和阈值时，需要平衡通过率和坏账率，以实现最大利息收入并降低资金损失风险。由于传统算法的计算量过大，普通计算机处理所需时间过长，而 QUBO[1]模型作为一种有效的数学模型，可以通过量子计算机硬件来快速求解组合优化问题。该模型的核心思想是将问题转化为决策变量为二值变量的形式，同时通过优化目标函数的二次函数形式进行求解。目前基于 QUBO 模型的量子专用算法具有广泛而深刻的实际应用前景。本文将把 QUBO 模型与模拟退火算法[3]结合求解组合优化问题。

1.2 目标任务

根据所给数据建立数学模型并将模型转化为 QUBO 形式解决以下问题：

问题一：从所给数据集的 100 张信用评定卡中选择 1 张并选择其合适阈值使银行最终收入最大。

问题二：假设已经选定所给数据集中的信用评定卡 1、2、3，设置各自合适的阈值使银行最终收入最大。

问题三：从所给数据集的 100 张信用评定卡中任取 3 张，并设置各自阈值，使银行最终收入最大。

二、 问题分析

问题一至问题三的目标均是使银行最终收入最大，由题可知，其影响因素有通过率、坏账率、利息收入率和贷款资金，为了简化模型，我们将贷款资金和利息收入率设为定值，分别为 100 万和 8%，则影响最终收入的因素仅考虑通过率和坏账率，其值由信用评分卡和阈值的选择唯一确定。

2.1 问题一的分析

本题可以看作一个二维组合优化问题，维度一是信用评分卡的选择，维度二是阈值的确定，每一个组合确定一组通过率和坏账率，我们需要根据通过率和坏账率计算每一个组合对应的最终收入，并选择使最终收入最大，同时满足阈值与评分卡均只有一个被选中这一约束关系的组合。首先我们将收入指标作为目标函数，将信用评分卡和阈值作为变量，并将阈值和评分卡的约束条件转换为数学公式表达，构建二次函数模型，之后引入 0-1 变量将该模型转换为 QUBO 模型，使用模拟退火算法求解，得到的结果即为阈值和信用评分卡的最优组合。

2.2 问题二的分析

对于问题二，变量维度比问题一增加了，我们需要在信用评分卡 1、信用评分卡 2、信用评分卡 3 中分别选择阈值进行组合，因此维度一到维度三分别是信用卡 1 阈值的确定、信用卡 2 阈值的确定和信用卡 3 阈值的确定，计算每一种阈值组合下的

总通过率和总坏账率，根据总的通过率和坏账率，计算每一种阈值组合的最终收入，同时根据“每一信用评分卡有且只有一个阈值”这一个约束条件，构建合理的函数模型，此外，我们还须合理降维，即按照原函数的逻辑，构建新变量下的二次函数模型，再转换为 QUBO 模型，使用模拟退火算法求解，得到的结果即为使最终收入最大的三种信用评分卡的阈值的最优组合。

2.3 问题三的分析

问题三在问题二的基础上，还增加了对三张信用评分卡的选择，我们需从 100 张信用评分卡选取 3 张信用评分卡，同时需要分别确定它们的阈值进行组合，计算每一种组合下的总通过率和总坏账率，根据总通过率和坏账率，计算每一种组合的最终收入，同时根据“只选三张信用卡”、“每一信用评分卡有且只有一个阈值”这两个约束条件，构建合理的函数模型，然后同样进行变量转换，使模型维度降低至二维，再转化为 QUBO 模型，使用模拟退火算法进行求解，找出使最终收入最大的组合。

三、 模型假设

1. 假设贷款资金为 1000000 元，银行贷款利息收入率为 8%
2. 100 张信用评分卡之间相互独立，互不影响
3.每种信用评分卡的阈值之间相互独立，互不影响
4.题目给出的数据集中的数据都真实可靠，都为有效值

四、 符号说明

符号	说明	单位
W	最终收入	
p_{ij}	第 i 张评分卡第 j 个阈值对应的通过率	
b_{ij}	第 i 张评分卡第 j 个阈值对应的坏账率	
r	利息收入率	
L	贷款资金	
H_{obj}	目标函数 Hamilton 量	
H_{c1}	约束条件 1 的 Hamilton 量	
H	总体 Hamilton 量	
x_i	信用评分卡的选择	
y_j	阈值的选择	
z_k	给定三张信用评分卡对阈值组合枚举的选择	
s_g	三张信用评分卡组合枚举的选择	
Q	QUBO 矩阵	
D	一维变量矩阵	

五、 问题一模型的建立与求解

5.1 模型的建立

5.1.1 确立目标函数

根据题目可知，一个信用评分卡和一个阈值确定一个通过率 p 和一个坏账率 b，通过率和坏账率与最终收入 W 的关系如下，

$$W = L(rp - rbp - pb) \tag{1}$$

为简化模型，我们设贷款资金 L=1 万、利息收入率 r=8%，即

$$h = rp - rbp - pb \tag{2}$$

为便于找出所有信用评分卡和阈值的组合中，h 最大值对应的信用评分卡和阈值，本问题中我们引入 0-1 变量，用x_i表示是否选择第 i 张信用评分卡，y_i表示是否选择第 j 个阈值，

$$x_i = \begin{cases} 0, & \text{不选第 i 张信用评分卡} \\ 1, & \text{选择第 i 张信用评分卡} \end{cases} \quad i \in [1,100], \tag{3}$$

$$y_j = \begin{cases} 0, & \text{不选第 j 个阈值} \\ 1, & \text{选择第 j 个阈值} \end{cases} \quad j \in [1,10], \tag{4}$$

引入 0-1 变量后的目标函数H_{obj}如下，

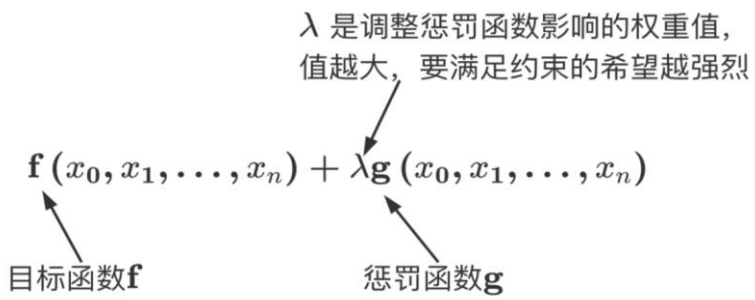
$$H_{obj} = \sum_{i=1}^{100}\sum_{j=1}^{10} rp_{ij}x_i y_j - (r+1)b_{ij}p_{ij}x_i y_j \tag{5}$$

5.1.2 构建约束条件及其对应的哈密顿算符[1]

问题一要求在 100 个信用评分卡中找出 1 张及其对应阈值，即约束条件为：

$$\sum_{i=1}^{100}\sum_{j=1}^{10} x_i y_j = 1 \tag{6}$$

正常有约束的优化问题[2]会变换成下面的形式，



其中惩罚函数[4]g()就是从约束条件获得的，本题构建惩罚函数的具体方法为：把约束条件转化为对应的哈密顿算符(Hamiltonian)，即构造一个关于x_i和y_j的二次多项式，让它在约束条件下取得最小值 0，此二次多项式即为构建的哈密顿算符H_{c1}。

本问题中构造的哈密顿算符如下：

$$H_{c1} = \left(\sum_{i=1}^{100} x_i - 1\right)^2 + \left(\sum_{j=1}^{10} y_j - 1\right)^2 \tag{7}$$

5.1.3 转换为 QUBO 格式

由于 QUBO 模型默认是求函数的最小值，因此我们对$-H_{obj}$取负号，则带约束条件的总体 Hamilton 量可表示为：

$$H = -H_{obj} + \lambda H_{c1} \tag{8}$$

代入H_{obj}和H_{c1}可得：

$$H = -\sum_{i=1}^{100}\sum_{j=1}^{10} r p_{ij} x_i y_j + (r+1) b_{ij} p_{ij} x_i y_j + \lambda[(\sum_{i=1}^{100} x_i - 1)^2 + \left(\sum_{j=1}^{10} y_j - 1\right)^2] \tag{9}$$

其中λ为调整惩罚函数影响的权重值，值越大，满足约束的希望越强烈，本模型中取λ =10。接着取一维变量矩阵 D 为

$$D = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_{100} \\ y_1 \\ y_2 \\ \cdots \\ y_{10} \end{bmatrix}$$

将二次多项式转为矩阵形式：

$$H = D^T Q D \tag{10}$$

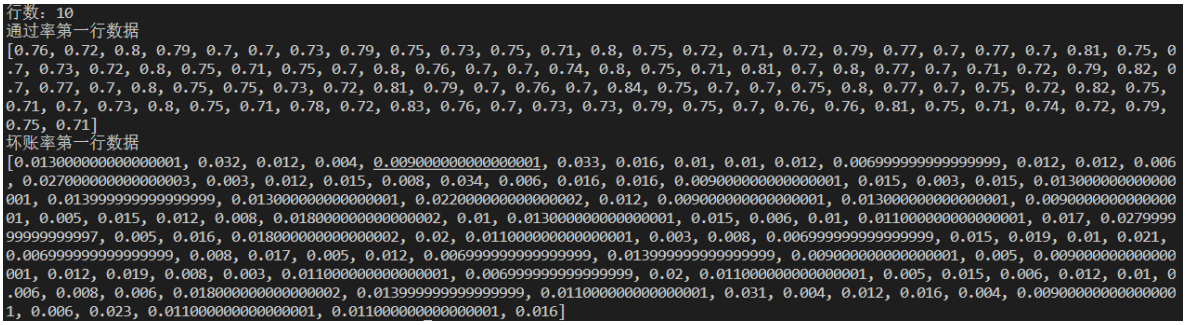
最后利用模拟退火算法以及 python 工具，编写代码进行 QUBO 模型的求解，求得使 H 值最小时对应的D矩阵[1]，从而确定被选中的信用评分卡和阈值。

5.2 模型的求解

5.2.1 数据预处理

首先将附件中的数据导入至 data 矩阵中，构建大小为 10*100 的 PASS_Aarray 和 LOSS_Array 矩阵，分别存储 100 张信用评分卡的通过率和坏账率。

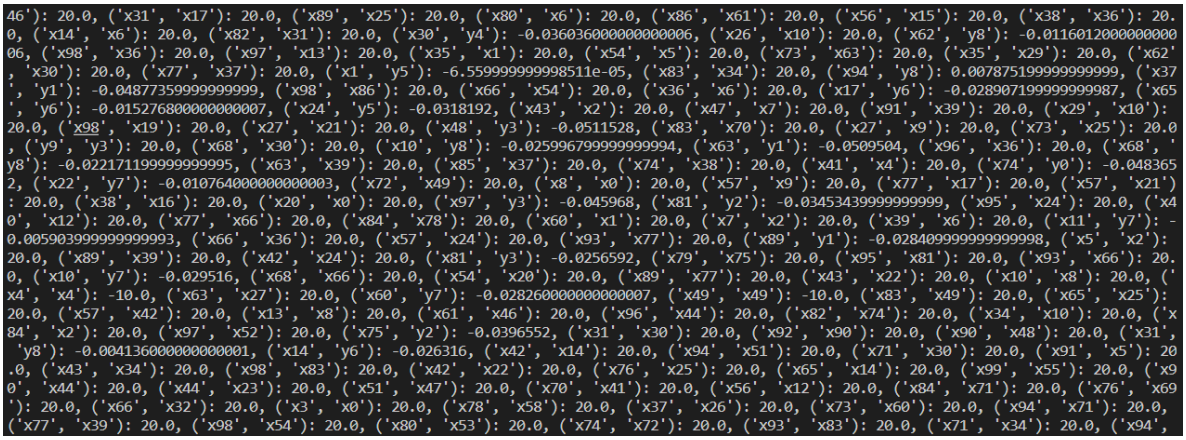
PASS_Aarray 和 LOSS_Array 都是 10*100 的二维矩阵，对应每一项的通过率与坏账率。部分数据如下图所示：



图一 通过率和坏账率矩阵的构建

5.2.2 目标函数矩阵化

根据公式（8）编写程序，利用 model=H.compile()构建汇编模型 model，再根据 qubo, offset = model.to_qubo()求得 QUBO 矩阵如图：

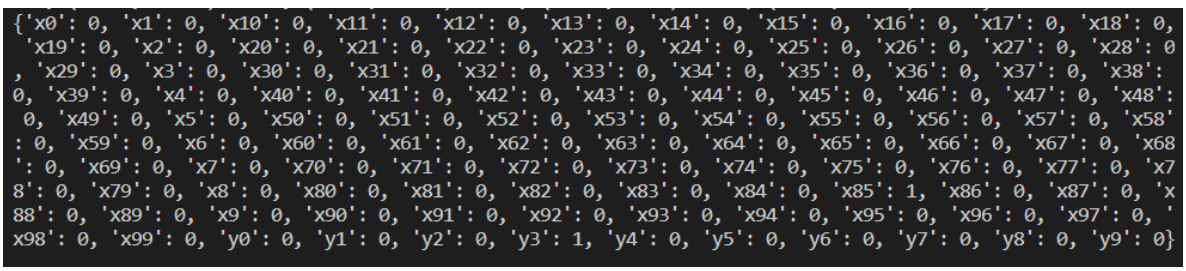


图二 问题一的 QUBO 矩阵

此处 Q 为 110*100 的矩阵，比如：('x66', 'x8'): 20.0，代表，第 66 行，第 8 列中的数据为 20.0。

5.2.3 变量矩阵 D 的求解

最后利用 sampler = neal.SimulatedAnnealingSampler()和 raw_solution = sampler.sample_qubo(qubo)求得对应的 D 矩阵，观察 D 矩阵，找到等于 1 的x_i和y_j，其下标即分别为选择的信用评分卡和阈值。

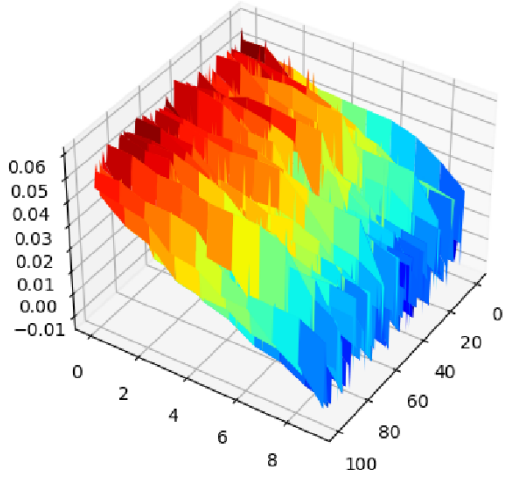


图三 问题一的最优 D 矩阵结果

从以上输出得知，QUBO 模型选中的评分卡为 x85 号，即评分卡 86.选中的阈值为 y3，即阈值 4。故此 QUBO 模型的解为评分卡 86，阈值 4 。（注意：qubo 算法由于其随机性，每次获得的解可能不同，或为局部最优解，这和建模方法和约束强度有关。对于我们的模型，约束强度为 10）

5.2.4 结果分析

由于数量级比较小，我们可以使用遍历的方法得到所有解，即所有评分卡和阈值对应的收益率，输出如下：



图四 每一张评分卡和其设定阈值对应的收益分布图

从图中可以看出，由于数据的离散性，我们很有可能进入局部最优解。整体上看，收益率高的分布都在于阈值比较低的位置，而评分卡的类型对整体影响不是很大。

六、 问题二模型的建立与求解

6.1 模型的建立

6.1.1 初步确立目标函数

要对信用评分卡 0、信用评分卡 1、信用评分卡 2 分别确定阈值进行组合，并找出使最终收入最大的最优组合，同样引入 0-1 变量：

$$y_j^1 = \begin{cases} 0, & \text{不选第一张信用评分卡的第 j 个阈值} \\ 1, & \text{选择第一张信用评分卡的第 j 个阈值} \end{cases} \quad j \in [1,10] \tag{11}$$

$$y_j^2 = \begin{cases} 0, & \text{不选第二张信用评分卡的第 j 个阈值} \\ 1, & \text{选择第二张信用评分卡的第 j 个阈值} \end{cases} \quad j \in [1,10] \tag{12}$$

$$y_j^3 = \begin{cases} 0, & \text{不选第一张信用评分卡的第 j 个阈值} \\ 1, & \text{选择第一张信用评分卡的第 j 个阈值} \end{cases} \quad j \in [1,10] \tag{13}$$

不同的组合对应的总坏账率为：

$$\mathrm{b}_{总} = \frac{1}{3} * \left(\sum_{j=1}^{10} b_{1j} y_j^1 + \sum_{j=1}^{10} b_{2j} y_j^2 + \sum_{j=1}^{10} b_{3j} y_j^3 \right) \tag{14}$$

总通过率为：

$$p_{总}=\sum_{j=1}^{10}p_{1j}y_j^1*\sum_{j=1}^{10}p_{2j}y_j^2*\sum_{j=1}^{10}p_{3j}y_j^3 \tag{15}$$

建立目标函数H_{obj}：

$$H_{obj}=rp_{总}-(r+1)b_{总}p_{总} \tag{16}$$

将$p_{总}$和$b_{总}$代入，可得：

$$\begin{aligned}H_{obj}=r\sum_{j=1}^{10}p_{1j}y_j^1*\sum_{j=1}^{10}p_{2j}y_j^2*\sum_{j=1}^{10}p_{3j}y_j^3-\frac{(r+1)1}{3}\\ *\left(\sum_{j=1}^{10}b_{1j}y_j^1\right.\\ \left.+\sum_{j=1}^{10}b_{2j}y_j^2+\sum_{j=1}^{10}b_{3j}y_j^3\right)\left(\sum_{j=1}^{10}p_{1j}y_j^1*\sum_{j=1}^{10}p_{2j}y_j^2*\sum_{j=1}^{10}p_{3j}y_j^3\right)\end{aligned} \tag{17}$$

6.1.2 初步构建约束条件及其对应的惩罚函数

约束条件为每种信用评定卡的阈值有且只能选一个，即：

$$\sum_{j=1}^{10}y_j^1*\sum_{j=1}^{10}y_j^2*\sum_{j=1}^{10}y_j^3=1 \tag{18}$$

和 5.1.2 的方法一样，将约束条件转换为哈密顿算符[6]如下：

$$H_{c1}=\left(\sum_{i=1}^{10}y_j^1-1\right)^2+\left(\sum_{j=1}^{10}y_j^2-1\right)^2+\left(\sum_{j=1}^{10}y_j^3-1\right)^2 \tag{19}$$

6.1.3 重新选取二维变量对目标函数进行降维

根据式（8）、式（17）和式（19），得到带约束条件的总体H：

$$\begin{aligned}H=-r\sum_{j=1}^{10}p_{1j}y_j^1*\sum_{j=1}^{10}p_{2j}y_j^2*\sum_{j=1}^{10}p_{3j}y_j^3+(r+1)1/3*(\sum_{j=1}^{10}b_{1j}y_j^1\\ +\sum_{j=1}^{10}b_{2j}y_j^2+\sum_{j=1}^{10}b_{3j}y_j^3)(\sum_{j=1}^{10}p_{1j}y_j^1*\sum_{j=1}^{10}p_{2j}y_j^2*\sum_{j=1}^{10}p_{3j}y_j^3)\\ +\lambda[(\sum_{i=1}^{10}y_j^1-1)^2+\left(\sum_{j=1}^{10}y_j^2-1\right)^2+\left(\sum_{j=1}^{10}y_j^3-1\right)^2]\end{aligned} \tag{20}$$

我们发现初步构建的总体 H 是一个四次函数,而 QUBO 模型需要我们将问题转化

为决策变量为二值变量的形式，同时通过优化目标函数的二次函数形式进行求解。为将其转换为 QUBO 模型，需要重新选取变量，以降低维度。

我们通过对三张信用评分卡的阈值组合进行枚举，将阈值和信用评分卡构成的矩阵转换为阈值枚举和信用评分卡的矩阵，引入枚举组合的 0-1 变量：

$$z_k = \begin{cases} 0, & \text{不选第 k 个枚举组合} \\ 1, & \text{选择第 k 个枚举组合} \end{cases} \quad k \in [1,1000]， \tag{21}$$

其中通过z_k的下标 k-1 可推断出三个评分卡阈值的设置，如 k=35 时，k-1=34，将其扩展为三位数即 034，百位数 0 代表信用评分卡 1 设置的阈值为 0，十位数 3 代表信用评分卡 2 设置的阈值为 3，个位数 4 代表信用评分卡 3 设置的阈值为 4，以此类推。

此时总坏账率$\mathrm{b}_{总}$可表示为

$$\mathrm{b}_{总} = \frac{1}{3} * \left(\sum_{k=1}^{1000} z_k \left(b_{1k} + b_{2k} + b_{3k} \right) \right) \tag{22}$$

总通过率$p_{总}$可表示为：

$$p_{总} = \sum_{k=1}^{1000} p_{1k} p_{2k} p_{3k} z_k \tag{23}$$

将$\mathrm{b}_{总}$和$p_{总}$代入（16）式，可得目标函数：

$$H_{obj} = \sum_{k=1}^{1000} z_k \ （r p_{1k} p_{2k} p_{3k} - （r+1）1/3 * (b_{1k} + b_{2k} + b_{3k}) p_{1k} p_{2k} p_{3k}) \tag{24}$$

6.1.4 新一维变量下构建约束条件及其对应的哈密顿算符

在新的二维变量下，只能选择一个阈值组合，即约束条件为：

$$\sum_{k=1}^{1000} z_k = 1 \tag{25}$$

将约束条件转换为哈密顿算符如下：

$$H_{c1} = \left(\sum_{k=1}^{1000} z_k - 1 \right)^2 \tag{26}$$

6.1.5 转换为 QUBO 格式

根据式（8）、式（24）和式（26），就得到了 QUBO 形式的总体 H：

$$H = -\sum_{k=1}^{1000} z_k \quad (rp_{1k}p_{2k}p_{3k} - \quad (r+1)\,1/3 * (b_{1k} + b_{2k} + b_{3k})p_{1k}p_{2k}p_{3k}) + \lambda\left(\sum_{k=1}^{1000} z_k - 1\right)^2 \tag{27}$$

本模型中λ取 10，接着取一维变量矩阵 D 为

$$D = \begin{bmatrix} z_1 \\ z_2 \\ \cdots \\ z_{1000} \end{bmatrix}$$

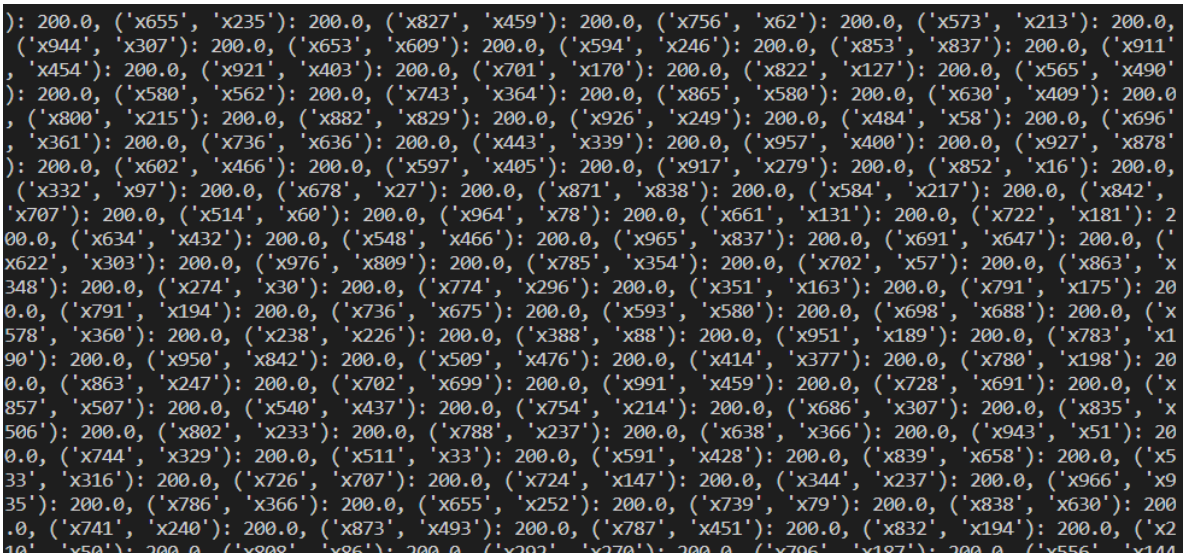
将函数转为矩阵形式：

$$H = D^T Q D \tag{28}$$

最后利用模拟退火算法以及 python 工具，编写代码进行 QUBO 模型的求解，求得使 H 值最小时对应的D矩阵，从而确定选中的枚举组合，根据枚举组合的下标确定三个信用评分卡阈值的设置。

6.2 模型的求解

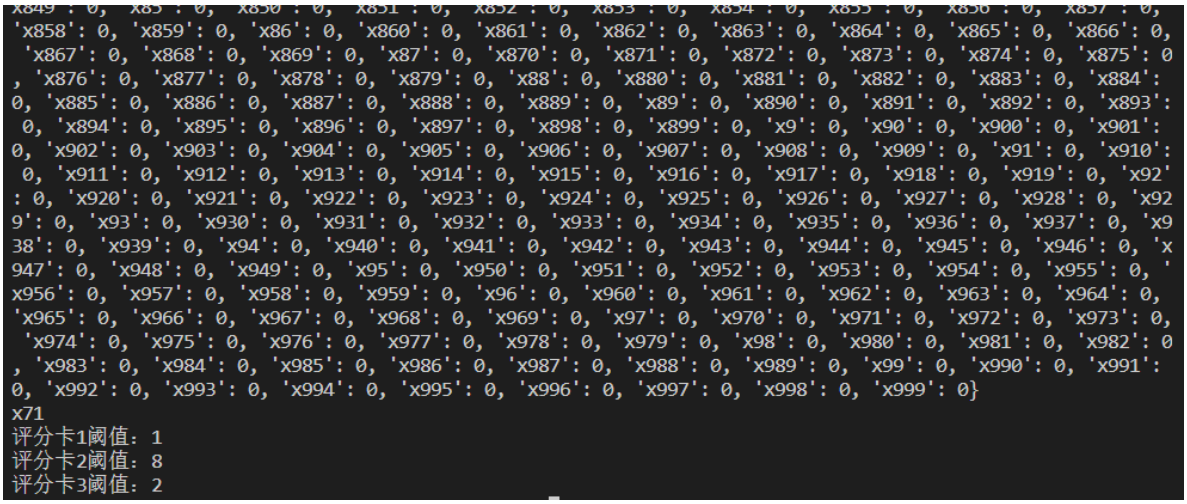
6.2.1 根据模型生成对应的 QUBO 矩阵，过程类似于问题一：



图五 问题二得到的 QUBO 矩阵

此处 Q 为 1000*1000 的矩阵，比如：('x655', 'x235'): 200.0，代表，第 655 行，第 235 列中的数据为 200.0。

6.2.2 根据 QUBO 矩阵求解：



图六 问题二的最优 D 矩阵结果

这里做了输出处理，可以很清晰的看到，评分卡 1 阈值：1，评分卡 2 阈值：8，评分卡 3 阈值：2

七、 问题三模型的建立与求解

7.1 模型的建立

7.1.1 确立目标函数

首先通过遍历算法枚举出所有三张信用评分卡的组合，同样对组合引入 0-1 变量：

$$s_g=\begin{cases}0, & \text{不选第 g 个信用评分卡组合}\\ 1, & \text{选第 g 个信用评分卡组合}\end{cases}\quad g\in[0,999999] \tag{29}$$

g 的 6 位数分别表示三个信用评分卡的选择，分别记为信用评分卡 l、信用评分卡 m、信用评分卡 n，其与 g 的关系如下：

$$\mathrm{g}=10000*\mathrm{l}+100*\mathrm{m}+\mathrm{n} \tag{30}$$

如 g=110722 时，前两高位 11 表示选中信用评分卡 12，中间两位 07 表示选中信用评分卡 8，最低两位 22 表示选中信用评分卡 23，以此类推。

对每一种信用评分卡的组合分别确定阈值的组合，结合问题二中引入的阈值枚举组合选择z_k，此时总坏账率$\mathrm{b}_{总}$可表示为

$$\mathrm{b}_{总}=1/3*(\sum_{g=0}^{999999}s_g(\sum_{k=1}^{1000}z_k\left(b_{(l+1)k}+b_{(m+1)k}+b_{(n+1)k}\right))) \tag{31}$$

总通过率$p_{总}$可表示为：

$$p_{总} = \sum_{g=0}^{999999} s_g \left(\sum_{k=1}^{1000} p_{(l+1)k} p_{(m+1)k} p_{(n+1)k} z_k \right) \tag{32}$$

将$b_{总}$和$p_{总}$代入（16）式，可得目标函数：

$$H_{obj} = \sum_{g=0}^{999999} s_g \sum_{k=1}^{1000} z_k \left(r p_{(l+1)k} p_{(m+1)k} p_{(n+1)k} - (r+1) * \frac{1}{3} \right. \\ \left. * \left(b_{(l+1)k} + b_{(m+1)k} + b_{(n+1)k} \right) * p_{(l+1)k} p_{(m+1)k} p_{(n+1)k} \right) \tag{33}$$

7.1.2 构建约束条件及其对应的哈密顿算符

信用评分卡组合和阈值组合均有且只有一个，即约束条件为：

$$\sum_{g=0}^{999999} s_g \sum_{k=1}^{1000} z_k = 1 \tag{34}$$

构造的哈密顿算符如下：

$$H_{c1} = \left(\sum_{g=0}^{999999} s_g - 1 \right)^2 + \left(\sum_{k=1}^{1000} z_k - 1 \right)^2 \tag{35}$$

7.1.3 转换为 QUBO 格式

根据式（8）、式（33）和式（34），就得到了 QUBO 形式的总体 H：

$$H = - \sum_{g=0}^{999999} s_g \sum_{k=1}^{1000} z_k \left(r p_{(l+1)k} p_{(m+1)k} p_{(n+1)k} - (r+1) * \frac{1}{3} \right. \\ \left. * \left(b_{(l+1)k} + b_{(m+1)k} + b_{(n+1)k} \right) * p_{(l+1)k} p_{(m+1)k} p_{(n+1)k} \right) \\ + \lambda \left[\left(\sum_{g=0}^{999999} s_g - 1 \right)^2 + \left(\sum_{k=1}^{1000} z_k - 1 \right)^2 \right] \tag{36}$$

本模型中λ取 10，接着取一维变量矩阵 D 为

$$D = \begin{bmatrix} s_0 \\ s_1 \\ \cdots \\ s_{999999} \\ z_1 \\ z_2 \\ \cdots \\ z_{1000} \end{bmatrix}$$

将函数转为矩阵形式：

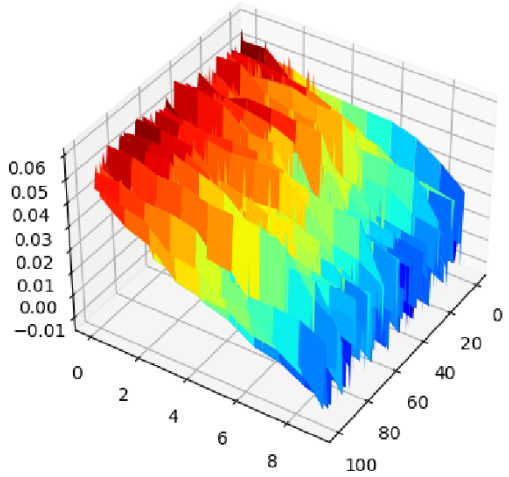
$$H = D^T Q D \tag{37}$$

最后利用模拟退火算法以及 python 工具，编写代码进行 QUBO 模型的求解，求得使 H 值最小时对应的D矩阵，从而确定选中的枚举组合，根据枚举组合的下标确定三个信用评分卡的选择及其阈值的设置。

7. 2 模型的求解

问题三的数量级是问题二的指数级，求解过程需要耗费很大的算力和时间，这里似乎出现一种以空间换时间的现象。以上论述已提供算法的思路，这里将仅提供求解的源码，其中含有普通遍历方法和 QUBO 模型转化法。读者兴趣可以自己运行。源码见附录（question_three_nomal.py/ question_three_qubo.py）

根据图：



我们可以做出一个合理的假设，即任何评分卡，都在 0-2 阈值之间取得较优解，如此，我们可以使用遍历方法（question_three_nomal.py），得到一组较优解，如下：

```
最大收益：0.043191913600000006
最优评分卡1:48
阈值::2
最优评分卡2:32
阈值::2
最优评分卡3:7
阈值::1
```

八、 信用评分卡 QUBO 模型的分析与检验

8.1 QUBO 解不同的原因分析

本文采取了模拟退火算法求解 QUBO 模型的最优解，由于算法的随机性，会导致每次运行的结果都可能不同，此外，QUBO 问题本身也可能具有多个局部最优解，这也是导致每次求解结果不同的原因之一。

为了获得稳定的结果，可以尝试多次运行 QUBO 求解器，并对每次求解结果进行统计分析，以确定最可能的最优解。另外，还可以使用一些确定性求解方法来解决 QUBO 问题，这些方法可以保证每次求解结果的一致性，但通常会牺牲一些求解速度,当然，这可能需要一些探索性的工作，在本文中不做讨论。

8.2 QUBO 建模中的难点分析

在使用 QUBO 方法建模时，我们遇到的一些的难点包括：

问题复杂度：QUBO 方法通常适用于 NP 难问题，这些问题的复杂度非常高，因此建模本身可能会很困难。为了解决这些问题，可能需要花费大量的时间来设计适当的变量和约束条件。

变量数目：QUBO 方法的求解时间和空间复杂度随着变量数目的增加而呈指数级增长。因此，在建模时需要考虑变量数目的限制，并尽可能减少变量数目。我们的模型，在本题中第一问变量数仅为 110 个，第二问中变量数为 1000 个，第三问中变量数达到了惊人的 1000×1000000 个，这可能是因为我们的模型建立的不是很好。

目标函数形式：QUBO 方法要求目标函数必须是一个二次型，这可能会限制建模的自由度。有时，需要使用一些技巧将非二次型的目标函数转化为二次型。我们在求解第二问时，发现出现了三次变量，这是显然无法使用 qubo 方法求解的，我们在想如何转化为二次型。对于 qubo 模型，一次型也是可以看作二次型，这样，我们通过把问题展开，赋予更多的二元变量，即可到达降低次数的目的，但是显然，这样会增加变量数，从而增加运算时间，我们猜测，如果能将问题直接建模为二次型，就不要先降为一次型而后看作二次型。

约束条件：在 QUBO 模型中，约束条件通常是通过惩罚项的方式添加到目标函数中的。但是，这可能会导致一些问题的求解变得更加困难，因为它们引入了一些复杂性和噪声。在建模时，需要仔细设计约束条件，以确保它们不会过度影响求解的结果。

求解器的选择：QUBO 方法的求解器有很多，每个求解器都有其独特的优缺点。在选择求解器时，需要考虑问题的性质、求解器的性能和可用资源等因素，并进行适当的权衡。我们使用的求解器为经典计算机，显然，经典计算机跑这个模型，特别是第三问的时候是非常吃力的，这说明经典计算机在求解 qubo 模型是不讨好的，用一般方法可能也会比 qubo 方法更快，所以，qubo 方法只钟情于量子计算机。

8.3 利用遍历算法对 QUBO 模型结果进行检验

1、我们分别编写三个问题的遍历算法，得到最优解分别为：

问题一：选取信用评分卡 48，设置阈值为 0，此时对应的最大收益指标为 0.061172

问题二：信用评分卡 1、信用评分卡 2、信用评分卡 3 的阈值均设为 3，此时对应的最大收益指标为 0.037313191520000004

问题三：评分卡组合：［8，33，49］阈值组合：［2，6，3］，此时对应的最大收益指标为 0.0438809712

2、而 QUBO 模型得到最优解分别为：

问题一：选取信用评分卡 48，设置阈值为 1

问题二：信用评分卡 1、信用评分卡 2、信用评分卡 3 的阈值分别 1、9、10

问题三：计算量过于庞大导致没能在给定时间内运行出结果。

3、通过比较我们不难发现，QUBO 模型得到的解与遍历算法的解还是存在一定差别，其可能原因是在不同的信用评分卡和阈值的组合下，其对应的通过率和坏账率求得的 H 值可能是相同的。此外，约束条件的权重λ的大小也会影响带约束的 QUBO 模型的准确性。

九、 信用评分卡 QUBO 模型的评价、改进与展望

9.1 模型的优点

1. 该模型转化为 QUBO 形式后，可以用量子计算机或量子退火器求解，在处理计算量庞大的组合优化类问题时，能大大节约时间，降低成本
2. 该模型能够处理大量的复杂组合问题，具有很强的可拓展性
3. 该模型的基本原理简单，只需调用库来求解 QUBO 矩阵即可

9.2 模型的缺点

1. QUBO 模型求解出来的解可能只是局部最优解，并且解并不固定，导致最终得到的结果存在误差
2. QUBO 模型的求解器有很多，我们采用的求解器只是普通计算机，因此在求解第三问时会很吃力
3. 由于实际问题中银行的利率会波动以及存在很多复杂的情况，所以我们模型求解的结果也会存在偏差

9.3 模型的改进

1. 可以选择更高效的求解器来加快求解速度，如量子计算机
2. 对模型的约束条件可以进一步优化以简化模型
3. 模型中的惩罚系数λ的选择可能不是最优，可以找出最优的λ来使模型的精度更高

9.4 QUBO 模型的展望

QUBO 模型作为一种重要的优化模型，在近年来已经得到了广泛的研究和应用。以下是我对 QUBO 模型未来的猜测和展望：

更高效的求解器：随着硬件技术的不断发展，我们可以期待更加高效的 QUBO 求解器的出现。比如，随着量子计算技术的不断发展，我们可以期待更加强大的量子求解器的出现，从而实现更高效的 QUBO 求解。

更广泛的应用领域：QUBO 模型已经在许多领域得到了应用，例如图像处理、物流管理、供应链优化等等。未来，我们可以期待 QUBO 模型在更多领域中得到应用，如金融、医疗、环境等领域。

更复杂的问题建模：QUBO 模型已经可以应用于解决很多 NP 难问题，但是对于某些更复杂的问题，例如约束满足问题(CSP)和布尔可满足性问题(SAT)，QUBO 模型的建模可能会变得更加困难。未来，我们可以期待更加智能的算法和方法的出现，来解决更复杂的问题建模。

更多的应用场景：随着量子计算技术的不断发展，我们可以期待 QUBO 模型在量子计算领域的应用得到更多的探索。例如，QUBO 模型可以用于解决量子化学问题，优化量子门序列等等。

总之，QUBO 模型是一个非常重要的优化模型，未来它将在各个领域得到更广泛的应用，并且有望通过更加高效的求解器和更智能的算法来解决更加复杂的问题。

十、 参考文献

[1] gang_akarui, (6 条消息) 量子退火算法入门（1） : QUBO 是什么？ _qubo 算法 _gang_akarui 的博客-CSDN 博客,2023

[2] gang_akarui, (6 条消息) 量子退火算法入门（2）：有约束优化问题的 QUBO 怎么求？ _qubo 算法_gang_akarui 的博客-CSDN 博客,2023

[3] Christos Papalitsas , Theodore Andronikos , Konstantinos Giannakis , Georgia Theocharopoulou and Sofia Fanarioti. A QUBO Model for the Traveling Salesman Problem with Time Windows.2019.

[4] GLOVER, FRED, KOCHENBERGER, GARY, DU, YU. Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models[J]. 4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies,2019,17(4):335-371.

[5] Thomas Morstyn , Senior Member, IEEE. Annealing-Based Quantum Computing for Combinatorial Optimal Power Flow. 2022

[6] gang_akarui, (6 条消息) 量子退火算法入门（4）：旅行商问题的 QUBO 建模「上篇」 _qubo 模型_gang_akarui 的博客-CSDN 博客，2023

附录

附录 1

介绍：支撑材料的文件列表

data_100.csv ：附件中的数据文件：
file_read.py：通过率和坏账率矩阵的读取
question_one_normal.py：问题一的遍历算法求解
question_one_qubo.py：问题一的 QUBO 模型模拟退火算法求解
question_two_normal.py：问题二的遍历算法求解
question_two_qubo.py：问题二的 QUBO 模型模拟退火算法求解
question_three_nomal.py：问题三的遍历算法求解
question_three_qubo.py：问题三的 QUBO 模型模拟退火算法求解

附录 2

介绍：file_read.py：通过率和坏账率矩阵的读取

```
    # 读取数据
import pandas as pd
import numpy as np
data = pd.read_csv("data_100.csv")
data = np.array(data)
list_pass=[]
list_loss=[]
PASS_Aarray=[]
LOSS_Array=[]
for j in range(10):
    for i in range(100):
        list_pass.append(data[j][2*i])
        list_loss.append(data[j][2*i+1])
    PASS_Aarray.append(list_pass)
    LOSS_Array.append(list_loss)
    list_pass=[]
    list_loss=[]
# 以下内容为测试
print(len(PASS_Aarray[0]))
print(len(PASS_Aarray))
print(PASS_Aarray)
print("/n")
print(LOSS_Array)
```

附录 3

介绍：question_one_normal.py：问题一的遍历算法求解

```
    import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = pd.read_csv("data_100.csv")
data = np.array(data)
list_pass=[]
list_loss=[]
PASS_Aarray=[]
LOSS_Array=[]
for j in range(10):
    for i in range(100):
        list_pass.append(data[j][2*i])
        list_loss.append(data[j][2*i+1])
    PASS_Aarray.append(list_pass)
    LOSS_Array.append(list_loss)
    list_pass=[]
    list_loss=[]

Card_id = np.arange(0,100,1)
Threshold = np.arange(0,10,1)

#假设贷款资金为1
Income = 0
Rate = 0.08 #收益率为8%
MAX_income =  0
Income_Height =
np.zeros((len(Threshold),len(Card_id)),dtype=np.float32)
for j in range(10):
    for i in range(100):
        Income = PASS_Aarray[j][i]*(1-LOSS_Array[j][i])*Rate -
PASS_Aarray[j][i]*LOSS_Array[j][i]
        Income_Height[j][i]=Income
        if(Income>MAX_income):
            MAX_income = Income
            MAX_i = i
            MAX_j = j


# print(Income_Height)

X,Y = np.meshgrid(Card_id,Threshold)
fig = plt.figure()
```

```
ax = fig.add_subplot(projection='3d')
surf = ax.plot_surface(X,Y,Income_Height,cmap='jet')
plt.show()


print("最大收益："+str(1000000*MAX_income))
print("评分卡:"+str(MAX_i))
print("阈值："+str(MAX_j))
##输出结果
#所有从 0 开始计数
# 最大收益：0.061172
# max_i:48
# max_j:0
```

附录 4

介绍：question_one_qubo.py：问题一的 QUBO 模型模拟退火算法求解

```
    import pandas as pd
import numpy as np
# neal 是模拟退火的库
import neal
# pyqubo 可以使用 Binary 定义变量，Constarint 定义约束
from pyqubo import Binary, Constraint

def process():
    data = pd.read_csv("data_100.csv")
    data = np.array(data)
    list_pass=[]
    list_loss=[]
    PASS_Aarray=[]
    LOSS_Array=[]
    for j in range(10):
        for i in range(100):
            list_pass.append(data[j][2*i])
            list_loss.append(data[j][2*i+1])
        PASS_Aarray.append(list_pass)
```

```
        LOSS_Array.append(list_loss)
        list_pass=[]
        list_loss=[]
    #假设贷款资金为1
    # Income = 0
    Rate = 0.08 #收益率为8%
    # MAX_income =  0

    X_select=[]
    Y_select=[]
    for i in range(100):
        X_select.append(Binary('x'+str(i)))

    for j in range(10):
        Y_select.append(Binary('y'+str(j)))
    M = 10
    H = (- Rate * Double_array_pass_sum(X_select,Y_select,PASS_Aarray)

+(1+Rate)*Double_array_pass_loss_sum(X_select,Y_select,LOSS_Array,PASS_
Aarray)
         + M * Constraint(((np.sum(X_select)-1)**2+(np.sum(Y_select)-
1)**2), label='only one x and y')
    )
    model = H.compile()
    qubo, offset = model.to_qubo()
    # print(qubo)
    # print(offset)
    sampler = neal.SimulatedAnnealingSampler()
    raw_solution = sampler.sample_qubo(qubo)
    print(raw_solution.first.sample)


def Double_array_pass_sum(X_select,Y_select,pass_arr):
    arr = np.array(pass_arr)
    list=[]
    for i in range(100):
        for j in range(10):
            list.append(arr[j][i]*X_select[i]*Y_select[j])
    return np.sum(list)

def Double_array_pass_loss_sum(X_select,Y_select,loss_arr,pass_arr):
    arr = np.array(loss_arr)
    list=[]
    for i in range(100):
```

```
        for j in range(10):
            list.append(arr[j][i]*pass_arr[j][i]*X_select[i]*Y_select[j])
    return np.sum(list)


if __name__=='__main__':
    process()
```

附录 5

介绍：question_two_normal.py：问题二的遍历算法求解

```
    import pandas as pd
import numpy as np
data = pd.read_csv("data_100.csv")
data = np.array(data)
list_pass=[]
list_loss=[]
PASS_Aarray=[]
LOSS_Array=[]
for j in range(10):
    for i in range(100):
        list_pass.append(data[j][2*i])
        list_loss.append(data[j][2*i+1])
    PASS_Aarray.append(list_pass)
    LOSS_Array.append(list_loss)
    list_pass=[]
    list_loss=[]

#假设贷款资金为1
Income = 0
Rate = 0.08 #收益率为8%
MAX_income =  0

for i in range(10):
    for j in range(10):
        for k in range(10):
            pass_rate =
PASS_Aarray[0][i]*PASS_Aarray[0][j]*PASS_Aarray[0][k]
            loss_rate =
(LOSS_Array[0][i]+LOSS_Array[0][j]+LOSS_Array[0][k])/3
            Income = pass_rate*(1-loss_rate)*Rate - pass_rate*loss_rate
            print(Income)
            if(Income>MAX_income):
```

```
                MAX_income = Income
                MAX_i = i
                MAX_j = j
                MAX_k = k

print("最大收益："+str(MAX_income))
print("评分卡 0 阈值:"+str(MAX_i))
print("评分卡 1 阈值:"+str(MAX_j))
print("评分卡 2 阈值:"+str(MAX_k))

# 最大收益：0.037313191520000004
#所有从 0 开始计数
# 评分卡 0 阈值:3
# 评分卡 1 阈值:3
# 评分卡 2 阈值:3
```

附录 6

介绍：question_two_qubo.py：问题二的 QUBO 模型模拟退火算法求解

```
    import pandas as pd
import numpy as np
# neal 是模拟退火的库
import neal
# pyqubo 可以使用 Binary 定义变量，Constarint 定义约束
from pyqubo import Binary, Constraint

def process():
    data = pd.read_csv("data_100.csv")
    data = np.array(data)
    list_pass=[]
    list_loss=[]
    PASS_Aarray=[]
    LOSS_Array=[]
    for j in range(10):
        for i in range(100):
            list_pass.append(data[j][2*i])
            list_loss.append(data[j][2*i+1])
        PASS_Aarray.append(list_pass)
        LOSS_Array.append(list_loss)
        list_pass=[]
        list_loss=[]
    #假设贷款资金为 1
```

```
    # Income = 0
    Rate = 0.08 #收益率为8%
    # MAX_income =  0

    X_select=[]

    for i in range(1000):
        X_select.append(Binary('x'+str(i)))

    arr_pass = np.zeros(1000)
    arr_loss = np.zeros(1000)
    for i in range(10):
        for j in range(10):
            for k in range(10):
                n=100*i+10*j+k

pass_num=PASS_Aarray[j][0]*PASS_Aarray[j][1]*PASS_Aarray[k][2]
                arr_pass[n]=pass_num

    for i in range(10):
        for j in range(10):
            for k in range(10):
                n=100*i+10*j+k

loss_num=LOSS_Array[j][0]*LOSS_Array[j][1]*LOSS_Array[k][2]
                arr_loss[n]=loss_num

    M = 100
    H = (- Rate * Double_array_pass_sum(arr_pass,X_select)
         +(1+Rate)*Double_array_pass_loss_sum(arr_pass,arr_loss,X_select)
         + M * Constraint(((np.sum(X_select)-1)**2), label='1000 have one
right choice')
    )

    model = H.compile()
    qubo, offset = model.to_qubo()
    # print(qubo)
    # print(offset)
    sampler = neal.SimulatedAnnealingSampler()
    raw_solution = sampler.sample_qubo(qubo)
    a = raw_solution.first.sample
    print(a)
    return a
```

```
def Double_array_pass_sum(pass_arr,x_select):
    list=[]
    for i in range(1000):
        list.append(pass_arr[i]*x_select[i])
    return np.sum(list)


def Double_array_pass_loss_sum(arr_pass,arr_loss,X_select):
    list=[]

    for i in range(1000):
        list.append(arr_loss[i]*arr_pass[i]*X_select[i])

    return np.sum(list)


if __name__=='__main__':
    list = process()

    for i in range(1000):
        if(list["x"+str(i)]!=0):
            num=i
            print("x"+str(i))

    print("评分卡 1 阈值："+str(int(num/100)+1))
    print("评分卡 2 阈值："+str(int(num // 10 % 10)+1))
    print("评分卡 3 阈值："+str(int(num % 10)+1))
```

附录 7

介绍：question_three_nomal.py：问题三的遍历算法求解

```
    import pandas as pd
import numpy as np
data = pd.read_csv("data_100.csv")
data = np.array(data)
list_pass=[]
list_loss=[]
PASS_Aarray=[]
LOSS_Array=[]
for j in range(10):
    for i in range(100):
        list_pass.append(data[j][2*i])
```

```
        list_loss.append(data[j][2*i+1])
    PASS_Aarray.append(list_pass)
    LOSS_Array.append(list_loss)
    list_pass=[]
    list_loss=[]

#假设贷款资金为1
Income = 0
Rate = 0.08 #收益率为8%
MAX_income =  0
for m in range(100):
    for n in range(100):
        if(m==n):
            break
        for l in range(100):
            if(m==l or n==l):
                break
            for i in range(10):
                for j in range(10):
                    for k in range(10):
                        pass_rate =
PASS_Aarray[i][m]*PASS_Aarray[j][n]*PASS_Aarray[k][l]
                        loss_rate =
(LOSS_Array[i][m]+LOSS_Array[j][n]+LOSS_Array[k][l])/3
                        Income = pass_rate*(1-loss_rate)*Rate -
pass_rate*loss_rate
                        if(Income>MAX_income):
                            MAX_income = Income
                            MAX_i = i
                            MAX_j = j
                            MAX_k = k
                            MAX_m = m
                            MAX_n = n
                            MAX_l = l
                            print(MAX_income)

print("最大收益："+str(MAX_income))

print("最优评分卡1:"+str(MAX_m))
print("阈值::"+str(MAX_i))

print("最优评分卡2:"+str(MAX_n))
print("阈值::"+str(MAX_j))
```

```
print("最优评分卡 3:"+str(MAX_l))
print("阈值::"+str(MAX_k))
```

附录 8

介绍：question_three_qubo.py：问题三的 QUBO 模型模拟退火算法求解

```
    import pandas as pd
import numpy as np
# neal 是模拟退火的库
import neal
import itertools
# pyqubo 可以使用 Binary 定义变量，Constarint 定义约束
from pyqubo import Binary, Constraint

def process():
    data = pd.read_csv("data_100.csv")
    data = np.array(data)
    list_pass=[]
    list_loss=[]
    PASS_Aarray=[]
    LOSS_Array=[]
    for j in range(10):
        for i in range(100):
            list_pass.append(data[j][2*i])
            list_loss.append(data[j][2*i+1])
        PASS_Aarray.append(list_pass)
        LOSS_Array.append(list_loss)
        list_pass=[]
        list_loss=[]
    #假设贷款资金为 1
    # Income = 0
    Rate = 0.08 #收益率为 8%
    # MAX_income =  0


    #做优化，比如 123，132，321，213，231，312 是一种情况，就可以不必计算
    pass_arr = np.zeros([1000000,1000])
    loss_arr = np.zeros([1000000,1000])
    digits = range(100)
    combos = itertools.combinations(digits, 3)

    # 输出所有可能的三个数字的无序组合
    for combo in combos:
```

```
        for i in range(10):
            for j in range(10):
                for k in range(10):
                    n = combo[0]*10000+combo[1]*100+combo[2]
                    m = 100*i+10*j+k
                    pass_arr[n][m] =
PASS_Aarray[i][combo[0]]*PASS_Aarray[j][combo[1]]*PASS_Aarray[k][combo[
2]]
                    loss_arr[n][m] =
LOSS_Array[i][combo[0]]*LOSS_Array[j][combo[1]]*LOSS_Array[k][combo[2]]



    X_select = []
    Y_select= []
    for i in range(1000):
        Y_select.append(Binary('y'+str(i)))


    for i in range(1000000):
        X_select.append(Binary('x'+str(i)))


    M = 10
    H = (- Rate * Double_array_pass_sum(pass_arr,X_select,Y_select)

+(1+Rate)*Double_array_pass_loss_sum(pass_arr,loss_arr,X_select,Y_selec
t)
         + M * Constraint(((np.sum(X_select)-1)**2+(np.sum(Y_select)-
1)**2), label='each have one right choice')
    )

    model = H.compile()
    qubo, offset = model.to_qubo()
    # print(qubo)
    # print(offset)
    sampler = neal.SimulatedAnnealingSampler()
    raw_solution = sampler.sample_qubo(qubo)
    a = raw_solution.first.sample
    print(a)
    return a


def Double_array_pass_sum(PASS_Aarray,X_select,Y_select):
```

```
    list=[]
    for i in range(1000000):
        for j in range(1000):
            list.append(PASS_Aarray[i][j]*X_select[i]*Y_select[j])
    return np.sum(list)


def Double_array_pass_loss_sum(pass_arr,loss_arr,X_select,Y_select):
    list=[]
    for i in range(1000000):
        for j in range(1000):

list.append(loss_arr[i][j]*pass_arr[i][j]*X_select[i]*Y_select[j])

    return np.sum(list)


if __name__=='__main__':
    list = process()
```