基于概率的众包定价策略

摘　要

手机内置传感器和移动网络的普及和发展催生了越来越多的众包平台。平台将任务发布于大众，为完成者提供一定的奖励。过低的价格很难起到有效激励的效果，而定价过高很可能导致平台的亏损。因此，如何考虑到外界因素，为众包任务定价成为其机制设计的核心，对众包平台的运行有重大意义。本题要求我们结合所给案例和数据改良定价机制。

对于问题一，我们需要通过数据分析APP原有的定价策略。根据所给条件，我们重点考察任务所处位置和其周边会员密度对价格的影响。我们先将经纬度转化为平面坐标进行数据可视化，定性得出会员越稀疏、任务越偏僻，则定价越高的结论。在定量方面，我们首先以位置为特征进行线性回归，后又对定价和人口密度的关系进行了非线性回归。我们同时发现，失败任务往往集中在项目价格过低、人口又稠密的地区。

对于问题二，我们将APP设计者定价、用户选取项目执行的过程看做两个可训练系统交互的过程，继而选择概率模型。我们对（会员，项目）对建立取决于两者位置、任务价格和会员信誉的激励函数场。从用户的角度，我们设计了与激励函数相关的可训练系统S_θ来模拟用户的行为模式，化归其为逻辑回归（Logistic Regression）的形式并用最大似然估计获得参数。从APP设计者的角度，在总预算和单一任务价值固定的假设下，希望重新分配任务预算以提高完成任务数量的期望，利用拉格朗日乘子法求出最大期望对应的定价。问题二中的模型使我们基于概率模型对于问题一中统计模型的改良，是我们的主要模型。

对于问题三，我们首先分析了K-均值（K-means）聚类和基于密度聚类（Density-Based Spatial Clustering of Applications with Noise）的可行性和存在的问题。继而运用问题二拟合的激励场参数提出了一种基于贪心的启发式迭代算法，并证明了问题二所提出的概率模型在一定参数环境下给项目聚类提供了一种良性、稳定的方案。

对于问题四，我们首先对新的任务集合可视化后与附件一数据对比。为了评估不同定价策略的影响，我们对比第一问拟合的题目策略，问题二中不包含聚类的新策略，问题三包含聚类的新策略表现。我们发现，附件三中的项目整体完成可能性偏低，这是直观的图形观察、问题一的回归模型和问题二的概率模型共同的结论。

在理论推导之外，对于四个问题，我们都使用局部化或者理想化进行了近似的定性分析，以确保整个模型体系与直观经验的吻合性。最后，我们证明了我们的模型是一般化的最大期望算法[1]（Expectation Maximization Algorithm， EM Algorithm）的特例，继而将我们的主要模型归类于一个更具普适的性的概率模型框架之中。

关键词：　群智感知　定价机制　回归分析　聚类分析　EM算法

1 问题重述

1.1 问题背景

近年来，智能手机和移动网络走进了每个人的生活。因为手机上装载各种不同的传感器，人们可以轻易收集和存储身边的信息，从图像到空气质量等[2]。伴随着硬件的发展，基于手机的新应用也层出不穷，基于互联网的移动众包就是其中之一。

众包（crowdsourcing），即公司或机构把过去由员工执行的工作任务，以自由自愿的形式外包给非特定的大众网络[3]。在信息收集领域，这种方式相比于传统方式大大节约了调查成本，有效地保证了调查数据真实性，缩短了调查的周期，促进了社会资源的合理利用。

然而，众包的自愿性也带来了更多的不确定性，因此如何在有限的条件下激励大众更多地参与众包在一直以来是一个热门的问题，激励不足或不合理的激励机制常常引起无法招募到足够的工人或工人大量欺诈、反悔。在激励方式上，尽管游戏化等非物质激励方式在某些方面展现出其独特的优势[4]，现金奖励依然是主流的激励方式。本文我们只考虑经典的现金激励金额对众包的影响。同时，因为现实条件的复杂性，我们往往还需要考虑工人和环境的异质性。在本题中，我们考虑这样一款众包APP：会员在不同地点为平台完成拍照任务后，平台提供了一定数额的现金报酬。我们将分析其现有的定价机制并进行优化，

1.2 问题描述

“拍照赚钱”是移动互联网下的一种自助式服务模式，题目中的APP即为这类平台：大众为平台拍取货架信息，得到平台提供的报酬。因为价格因素直接影响用户完成任务的积极性，因而任务定价成为平台有效运行的核心要素。基于题目中提供的某地区内用户的位置信誉信息，一组该地区历史任务定价和完成情况，我们将对众包过程进行建模，分析不同人物奖金的变化对平台收益的影响。

问题一要求我们结合附件一中的一组已结束的任务标价及完成情况，和附件二会员的地理位置，推测平台的定价策略。同时，我们可以针对失败的任务分析定价策略的不合理性。

问题二要求我们对这次这组任务设计新的定价策略，与原方案比较。我们将根据问题一分析的原方案，对找到的定价问题进行改善，将新方案的预期结果与原结果比较；我们也将针对分析的效果评估模型寻找最优定价方案。

问题三要求我们研究将距离近的任务打包发布的效果，并为打包后的任务制定定价方案。我们将利用之前得到的效果评估方法寻找可以优化结果的任务合并和定价方法。

问题四要求我们将新定价策略应用于附件三另一个数据集并评估其表现。我们将把打包与不打包的新定价方案和问题一求得的原策略对比，研究它们的收益。

之前任务的定价和完成情况和一组新的任务坐标。问题一我们将根据之前的任务研究定价规律，分析部分任务未完成的规律；问题二我们设计新的定价方案，与原方案对比收益；问题三我们考虑将部分任务打包发布对结果的影响；问题四我们会把新的方案应用于

新的任务，与原方案进行对比。

2 模型假设

1. 假设每个任务对平台价值相同。一方面，题目除了位置信息外没有更多关于任务本身的信息，平台可以将任务划分为相近价值，因而我们可以假设它们价值相同；另一方面，这样平台的总收入和完成任务数成正比，便于之后的优化。

2. 假设所有会员信息对等，知道所有任务的地点和信息。在之后的过程中，我们考虑会员可以对所有任务的信息综合决策。

3. 假设开始每个任务时会员的起点都处于附件2的位置（考虑任务打包时“每个任务”改为“每包任务”）。因而，在非打包任务中，对于一个多任务的用户，我们不用考虑其路径选择，只需考虑他的起点到每个任务坐标的距离之和。

4. 一个会员对任务的倾向性与其他人独立，会员对多个任务的倾向性相互独立（倾向性直接影响完成任务的概率）。因此，在我们计算某会员完成一项任务的概率时，不用考虑其他行为的影响。

5. 忽略除任务价格、位置以外的外界因素对会员做任务的影响。在这个假设下，我们只要考虑题目给出的任务个人信息。

3 符号说明

表1列出了本文需要的符号。

表 1: 符号说明

符号	符号描述
$P(X,Y)$	P点纬度经度分别是X,Y
T	任务集合 $T=\{t_1,\ldots,t_m\}$
W	用户集合 $W=\{w_1,\ldots,w_n\}$
v_{t_k}	任务t_k标价
z_{t_k}	任务t_k完成度（1-完成, 0-未完成）
$\boldsymbol{v}$	任务价格向量$(v_{t_1},\ldots,v_{t_m})^{\mathrm{T}}$
$\boldsymbol{z}$	任务完成向量$(z_{t_1},\ldots,z_{t_m})^{\mathrm{T}}$
$d(P,Q)$	平面上点P,Q间欧氏距离
$c(P,r)$	点P为圆心，r为半径的圆内用户数
C	平台总预算
E	所有人完成任务总数量的期望值

4 问题分析

4.1 问题一

4.1.1 定价规律的定性分析

本题的第一问要求找出附录一中项目的定价规律，并分析任务未完成的原因。因为本题有很明确的现实意义，所以我们可以先将附录一和附录二中给出的项目位置和用户位置在同一张图1中展示，在这一过程中我们直接忽略了用户数据集上的离群点：

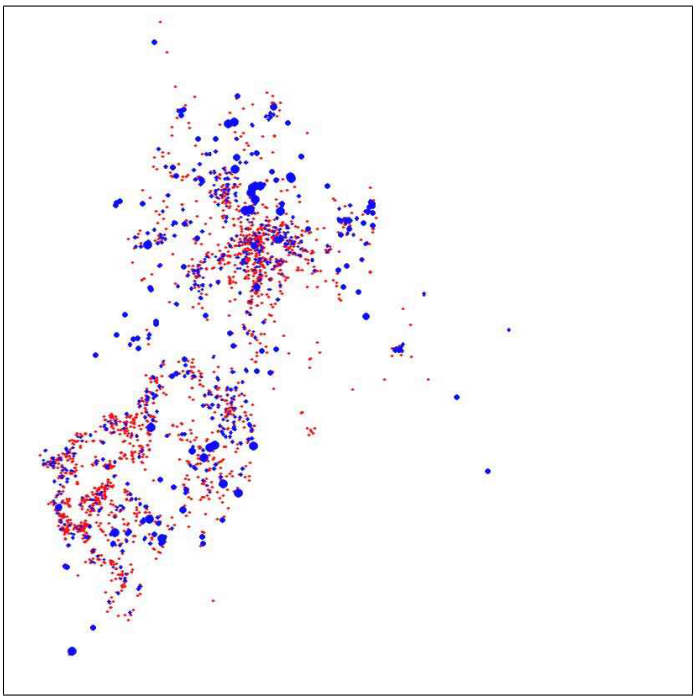

图 1: 项目和用户的地理分布，蓝点代表项目（体积越大，标价越高），红点代表用户

从图1中可以观察到一个直观的结论：项目的标价在用户比较稀疏的场合增加，在用户比较密集的场合减少，这一现象的具体推理在下一小节给出。

我们对问题一作这样的假设：APP的设计者预先只有关于用户密度在空间中分布的信息，求一个以单个项目的属性（经度，纬度）为参数的函数，返回其估价。因为已知用户分布聚类边缘的项目标价高于靠近聚类中心的标价，所以我们不妨进一步认为标价是关于某种距离度量的函数。采用二维空间中最普遍的欧氏距离，我们得到如下回归方程：

$$v_t = \alpha_1 x_t + \alpha_2 x_t^2 + \alpha_3 y_t + \alpha_4 y_t^2 + \alpha_5$$

该方程可以认为是对于一个项目坐标 (x, y) 到一系列用户聚类中心的距离之加权和，记回归参数为列向量$\boldsymbol{\alpha}$，另有$\boldsymbol{x} = (x, x^2, y, y^2, 1)^{\mathrm{T}}$，则$v = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\alpha}$，以$\boldsymbol{P}$记所有项目在附录一中的定价构成的列向量，记$\boldsymbol{X}$将所有任务的$\boldsymbol{x}$按列组成的矩阵，得到以矩阵形式表达的回归方程：

$$\boldsymbol{P} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha} \tag{1}$$

该回归方程的解析解由下式给出：

$$\boldsymbol{\alpha} = (\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}})^{-1}\boldsymbol{X}\boldsymbol{P}$$

然而，采用最小二乘法求解的线性回归算法并不是这道题目的最佳解，因为最小二乘法蕴含了输出结果上的加性高斯噪声，但是本题中的定价都是分布在$[65.0, 85.0]$，以0.5为步长的离散的数值，所以理应存在一个更直接地利用聚类性质的估价方法。

根据早先的定性观察，项目的定价和其周围的用户密度负相关，所以我们可以以固定半径r统计各种价格的项目周边的用户密度。在给定一个新项目时，我们先从其地理信息求出其周边用户密度，继而将该密度值和各价位对应的用户密度比对，取最接近的项目估价为新项目估价。整个过程的流程图如图2，其中蓝色单元代表给定的参数，绿色单元代表可训练的模块，箭头的方向代表信息的流动。

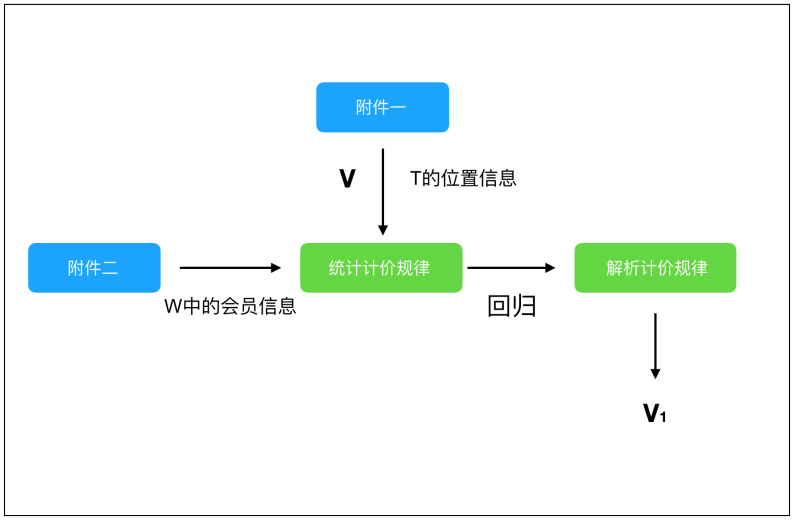

图 2: 问题一模型流程图

4.1.2 任务失败原因的分析

针对问题一的第二方面，即项目失败的原因，我们从两个角度进行定性解读：

第一，对于不同定价的项目被成功执行率进行统计，结果如下：

表 2: 附件一不同定价的成功率

任务标价	$[65, 70)$	$[70, 75)$	$[75, 80)$	$[80, 85]$
完成率	54.1%	74.4%	75.5%	82.5%

可以看出，高价位项目的被执行率普遍地高于低价位项目，这就意味着对于APP设计者的定价方案而言，其低价位定价策略相对过于保守，使得用户没有兴趣执行。

第二，观察成功/失败项目的地理分布图3：

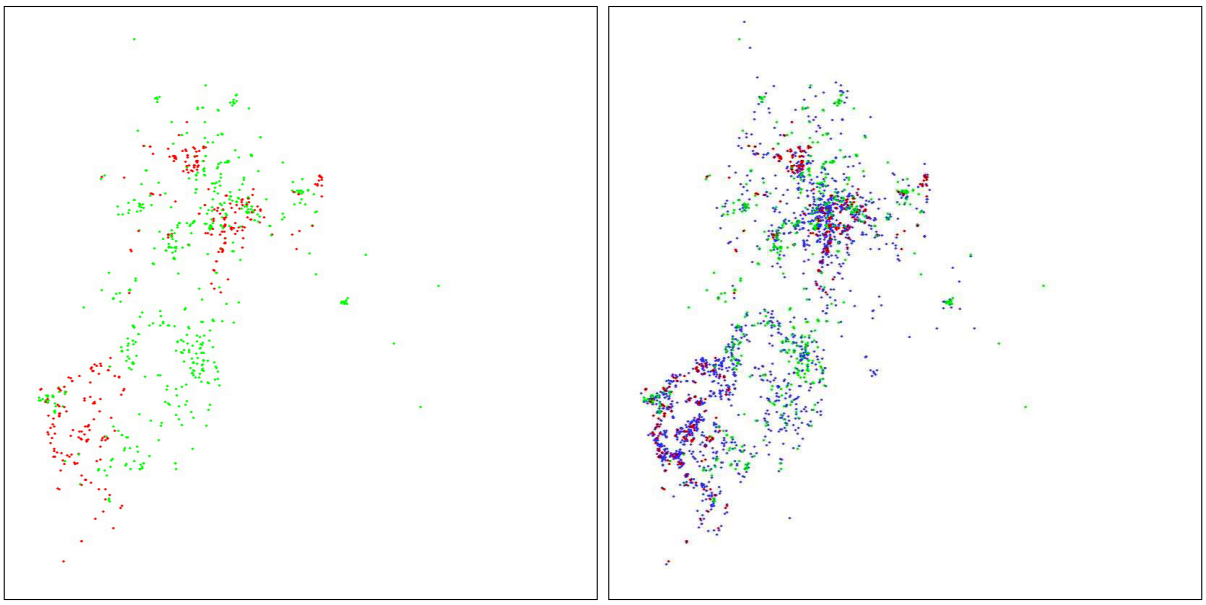

图 3: 成功项目(绿色)，失败项目(红色)和用户(紫色)的地理分布

从上两图中可以清楚地看到：在用户的密度密集程度极高的地区（如地图左下，上方中部），项目完成率反而很低。这个结果与第一点相吻合：平台因为用户的稠密而过低估计价格，从而导致任务对所有人都缺乏足够的吸引力。

4.1.3 估价与用户密度关系的推导

任务的估价随着周围用户数量的增加而减少，这是我们从图1中观察到的定性结论，这一结论可以这样解释：

考虑一个启发式定价的策略，其中一个项目的原始定价为0，估价制定者观察项目是否被执行，如果没有被执行，其价格会得以增加，假设项目的估价随着时间线性增加，则项目被执行时的最终价格可以被认为是一个可以使得周围用户接受的价钱。

其次考虑用户的行为方式，假设任务附近的用户很多，竞争激烈，故他们对于价格的敏感性不高，设前来领取任务的间隔时间服从指数分布，因为指数分布是和时间无关的分布，具有无记忆性，比较适合描述用户的行为动态。为简单起见，假设项目附近可以选择认领项目的用户数量为n，指数分布的均值为$\frac{1}{\lambda}$，则该项目直到最终被认领所花费的时间是n个同均值指数分布随机变量的最小值，也是一个指数分布，其期望为：

$$\mathbb{E}(v) = \frac{1}{n\lambda}$$

所以项目的估价和其周围用户数量呈负相关。

4.2 问题二

4.2.1 建立概率模型

问题二要求利用附件一和附件二中的信息，对于附件一中的任务进行重新定价。根据假设，我们的目标是：在所有项目的总价格不变的情况下，提高收益。而由于所有任务价值相等，我们要优化任务完成率的期望，即要使得被完成的项目的数量尽可能多。因为项目被完成的情况和人选择项目完成的情况都有随机性，我们建立概率模型，目标取一个$\boldsymbol{v}$来最大化$\boldsymbol{z}$的元素和，为了完成这一计算过程，需要估计概率分布：

$$p(\boldsymbol{z}|\boldsymbol{v})$$

但这个分布并不容易直接近似求解。为了拆分整个系统的复杂性，我们引入隐变量θ表示用户原参数组成的列向量，以便对于系统中用户的行为方式进行模拟，换言之，虽然$p(\boldsymbol{z}|\boldsymbol{v})$难以直接求解，但是下式是相对易求的：

$$p(\boldsymbol{z}|\theta,\boldsymbol{v})$$

对于θ的求值，可从贝叶斯公式得到：

$$p(\theta|\boldsymbol{z},\boldsymbol{v})=\frac{p(\theta,\boldsymbol{z},\boldsymbol{v})}{p(\boldsymbol{z},\boldsymbol{v})}\propto p(\theta,\boldsymbol{z},\boldsymbol{v})=p(\boldsymbol{z}|\theta,\boldsymbol{v})p(\theta)p(\boldsymbol{v})\propto p(\boldsymbol{z}|\theta,\boldsymbol{v})$$

在对于$p(\theta)$没有先验知识的情况下，这是对于θ的最大似然估计θ_{ML}。我们把求解$p(\boldsymbol{z}|\theta,\boldsymbol{v})$的过程抽象为一个系统$S_\theta$，则整体思路如下：

1. 设计系统S_θ，其中θ为可训练参数，它是一个函数：

$$S_\theta(\boldsymbol{v})=\boldsymbol{z} \tag{2}$$

2. 根据附件一给出的$\boldsymbol{v}$和$\boldsymbol{z}$，最大似然估计模型的参数θ_{ML}。

3. 根据训练的S_θ，尝试优化目标E。

问题二模型的整体流程图如图4所示：

4.2.2 评价系统S_θ的设计

评价系统S_θ的设计旨在模拟用户的行为模式，根据假设，我们认为用户的行为服从如下的基本模式，用户执行一个项目的积极性：

- 和该项目的奖金v_t呈正相关；
- 和该用户到该项目位置的距离$d(w,t)$以某种方式呈负相关；

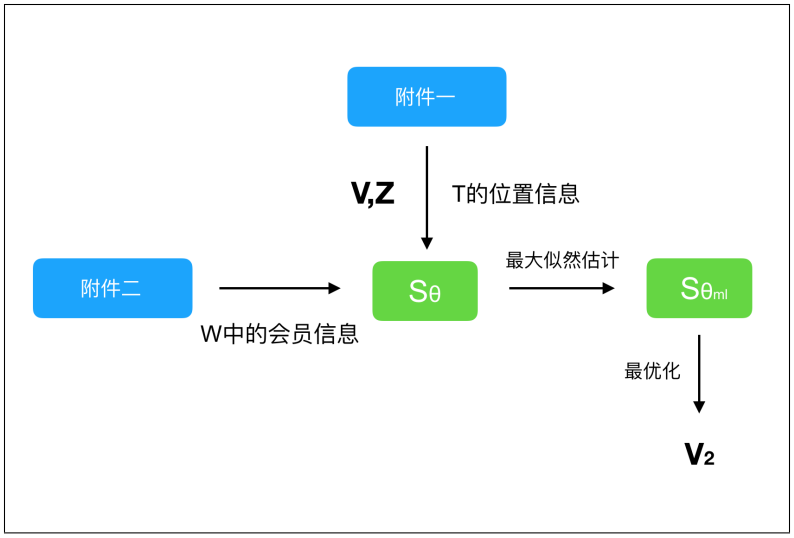

图 4: 问题一模型流程图

- 和该用户自身的信誉度正相关，也即选择额度越高、选择时间越早，越容易执行项目。

上述两个和用户有关的关联性可以合并为因子：

$$f(w,t) = \frac{c_w}{d(w,t)^2} \tag{3}$$

其中c_w和用户w的信誉度正相关且为正。注意到此式只用到任务的位置信息，我们可以考虑只和与用户w的距离r, c_w有关的标量场，我们称之为w的激励场，满足相加性，此时用户相当于场的发生源，多个用户同时存在时，一点上激励场即为所有人单独形成场强的简单叠加，如图5最右侧所示。通过引入新度量$f(w,t)$和其平方项，平方根项为系统引入更多非线性性，假设其他参数关系都是线性的，下式度量的是项目t收到用户w产生的激励场的影响：

$$A_{w,t} = \alpha \cdot v_t + \beta f(w,t) + \gamma f(w,t)^{\frac{1}{2}} + \delta f(w,t)^{\frac{1}{4}} + \epsilon$$

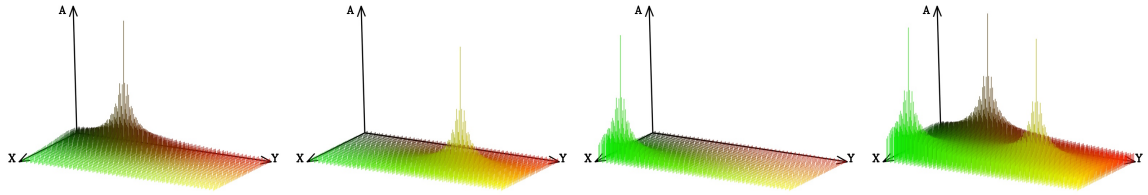

图 5: 前3张为3位用户单独存在时形成的激励场，第4张为3个人同时存在时场的叠加

对于一个任务t而言，其被执行的概率$p(t)$正相关于所有会员执行它的积极性之和$\sum_w A_{w,t}$. 我们使用Sigmoid函数进行归一变换，将$(-\infty, +\infty)$映射到$(0,1)$:

$$p(t) = \sigma(\sum_{w \in W} A_{w,t}) = \frac{1}{1 + \exp\left\{-\sum_{w \in W} A_{w,t}\right\}}$$

这是一个概率模型，通过期望的线性可以求被执行任务总数的期望，也即我们试图最大化的目标，根据假设1以及符号约定$E = \mathbb{E}(Tasks)$，那么：

$$E = \sum_{t \in T} p(t) \tag{4}$$

故基于这一思路对问题二的求解即在满足以下约束条件时：

$$\sum_{t \in T} v_t = C,\ v_t \geq 0, \forall t \in T$$

求$\boldsymbol{v} = (v_1, \ldots, v_m)^{\mathrm{T}}$，使得下式最大化：

$$\sum_{t \in T} \sigma \left\{ \sum_{w \in W} A_{w,t} \right\}$$

该最优化解析解可以通过引入拉格朗日乘子直接求解。

4.2.3 评价系统S_θ的参数估计

我们通过附录一中的数据对参数α，β和γ进行估计。

在附录一给定$\boldsymbol{v}$和$\boldsymbol{z}$的情况下，我们试图寻找到一组值$\theta = \{\alpha, \beta, \gamma\}$，记$p(t, \theta)$为函数$S_\theta$对于项目$t$估计的执行概率，进行最大似然估计：

$$p(\theta|\mathbf{z}, \mathbf{v}) \propto \mathbf{p}(\mathbf{z}|\theta, \mathbf{v}) = \prod_{\mathbf{t} \in \mathbf{T}} \mathbf{p}(\mathbf{t}, \theta)^{\mathbf{z_t}} (\mathbf{1} - \mathbf{p}(\mathbf{t}, \theta))^{\mathbf{1} - \mathbf{z_t}} \tag{5}$$

也即最大化其对数(忽略常数项)：

$$H(\theta) = \sum_{t \in T} z_t \ln p(t, \theta) + (1 - z_t) \ln \{1 - p(t, \theta)\} \tag{6}$$

不难发现，对于θ进行优化的问题正是一个逻辑回归（Logistic Regression），而其输出也恰恰是二类分类的概率，所以可以直接套用分类理论中的参数调节方法。比较常用的寻参方法有梯度下降与其变种IRLS(Iterative Reweighted Least Squares)，在数值模拟的情况下也可以尝试遍历参数空间来快速寻求一个近似解。

4.2.4 $S_{\theta_{ML}}$最优化

引入拉格朗日乘子，对于任一个项目t，我们试图求其估价v_t的平衡条件，即(注意到与v_t无关的量已经被写进一个常数)：

$$\frac{\partial}{\partial v_t} \left\{ \sum_{t \in T} \sigma(\alpha \cdot v_t + C_t) + \lambda \left\{ \sum_t v_t - C \right\} \right\} = 0 \tag{7}$$

利用σ函数的性质：

$$\frac{\partial}{\partial\sigma}\sigma = \sigma(1-\sigma)$$

可将式(7)化为：

$$\alpha\sigma(\alpha \cdot v_t + C_t)(1-\sigma(\alpha \cdot v_t + C_t)) + \lambda = 0 \tag{8}$$

所以σ函数的值等于(对式(8)使用均值不等式易知其有两个处于[0,1]解)：

$$\frac{1 \pm \sqrt{1+4\lambda/\alpha}}{2} \tag{9}$$

故：

$$v_t = \frac{1}{\alpha}\left\{\sigma^{-1}(\frac{1 \pm \sqrt{1+4\lambda/\alpha}}{2}) - C_t\right\} \tag{10}$$

到目前为止v_t是关于λ的函数，通过计算：

$$\sum_{t\in T} v_t(\lambda) = C \tag{11}$$

可以从数值上解出λ。

4.2.5 结果讨论

本节就上述求解所得的形式进一步讨论。

因为：

$$\sigma^{-1}(x) = -\ln(\frac{1}{x} - 1)$$

所以：

$$\sigma^{-1}(\frac{1+x}{2}) + \sigma^{-1}(\frac{1-x}{2}) = -\ln(\frac{2}{1+x} - 1) - \ln(\frac{2}{1-x} - 1) = 0$$

当所有项目的估价都根据式(10)取到平衡条件时，它们被选取的概率只有两种可能值，对应式(9)。

换言之，最优化策略使得所有项目被选取的概率趋近两个匀质的类别，每一个类别中所有项目被完成的概率是相同的，记两个类别中项目被选取的概率为以较高的概率p_1和较低的概率p_2，当两个类别中确实都有项目存在时，我们有$p_1 + p_2 = 1$（这一性质隐含了最大化期望的分配方法是取$A = 0$）。

假设选择 $(T - A)$ 个项目 $(A \leq 0.5T)$ 使其被选取概率为：

$$p_1 = \frac{1 + \sqrt{1+4\lambda/\alpha}}{2}$$

另外A个项目被选取概率为：

$$p_2 = \frac{1 - \sqrt{1 + 4\lambda/\alpha}}{2}$$

记：

$$m = \sigma^{-1}(\frac{1 + \sqrt{4\lambda/\alpha}}{2})$$

我们首先展开式(11)：

$$(T - A)m - Am = C + \frac{1}{\alpha}\sum_t C_t$$

$$m = \frac{C + \sum_t C_t/\alpha}{T - 2A} \tag{12}$$

由此可见，刻意选取部分项目使得其被选取概率变低能够使得其他项目被选取的概率变高，这也和我们的直觉相同。

选定A时，被完成项目数的期望为：

$$(T - 2A)\sigma(\frac{C'}{T - 2A}) + A$$

此时的最优解是取$A = 0$，此时所有项目被选取的概率均为：

$$\sigma(m) = \sigma(\frac{C + \frac{1}{\alpha}\sum_t C_t}{T}) \tag{13}$$

因为$Sigmoid$函数是单调增的，可以定性地认识式(13)的推论：

- 项目被选取的概率与C，即APP公司对于所有项目的总预算呈正相关，即APP公司投入的总资金越多，项目被执行的成功率越高；
- 项目被选取的概率与$\sum_t C_t$，也即项目导致的路程流畅性呈正相关(C_t是一系列与距离负相关的项之和)，即项目被设置的位置越利于用户接近，项目被执行的成功率越高。
- 项目被选取的概率与T，也即项目的总数呈负相关，可以理解为在APP公司的总投入一定时，项目总数增加将导致平局每个项目的赏金变低，继而降低成功率。

4.3 问题三

4.3.1 定性分析

问题三要求根据位置等信息，将某些位置相近任务联合在一起打包发布，从直观上而言，将地理上毗邻的项目聚类合并，并交给某一个用户完成，可以减少大量客户往来自己所在地和项目所在地的损耗。我们首先尝试分析两种最经典的聚类算法：K-均值聚类[5]（K-means Clustering）和基于密度聚类[6]（ Density-Based Spatial Clustering of Applications with Noise（DBSCAN））。

K-均值聚类可以将项目根据地理位置划分为K个聚类，但是它并不适用于本问题。K平均本身是混合高斯分布（Mixture of Gaussian）的最大似然估计算法之近似，所以其最适用的场合是所有原始数据分布服从一系列不同高斯分布的情况，但是观察图1可知本系列问题中的项目地理位置分布并没有明显的正态分布样式。

基于密度聚类是一种基于当前位置数据项密度的启发式算法，其聚类思想如下：

首先选择$r > 0, N \in \mathbb{N}$，如果两个点A, B的距离$d(A, B) < r$，我们定义 A, B 是直接相连的。如果至少有N个点与A直接相连，那么定义A是一个核心点。如果存在一条路径$A = p_1, p_2, \ldots, p_n = B$ 使得对于 $i \in [n-1]$， p_i, p_{i+1}是直接相连的，则我们定义A, B 是相连的。定义关系 $A \sim B$ 当且仅当A与B是相连的，这是一个等价关系，诱导的等价类构成了点集的一个划分。基于密度聚类要求我们将每个包含核心点的等价类聚为一类。

4.3.2 执行聚类的原因分析

项目聚类的起因是直观的，不过沿用问题二中的概率模型，我们可以证明：将密集程度高的项目聚为一类并视为一个任务确实有利于任务众包。

首先，对于已经被聚类的项目形成的项目集合T^*，对于用户w对T^*的完成激励进行量化：沿用问题二中的假设，式(11)中需要重新度量的是该用户执行任务集合T^*所必须的路程$d(w, T^*)$，它应该是T^*中所有点和代表w的点构成的点集的哈密尔顿路径：

$$d(w, T^*) = H(T^* \bigcup w)$$

类比于式(3)定义：

$$f(w, T^*) = \frac{c_w}{d(w, T^*)^2}$$

我们有：

$$A_{w,T^*} = \sum_{t \in T^*} (\alpha \cdot v_t) + \beta \cdot f(w, T^*) + \gamma \cdot f(w, T^*)^{1/2} + \delta \cdot f(w, T^*)^{1/4} + \epsilon$$

$$\sum_w A_{w,T^*} = \sum_{t \in T^*} \alpha \cdot v_t + C_{T^*}$$

参数$\theta = (\alpha, \beta, \gamma, \delta, \epsilon)^{\mathrm{T}}$使用5.2.1中所求的结果。

在判断一个聚类T^*是否比其中元素分开更合理时，我们分别求两种状况下它们进入S_θ后给APP设计者带来的项目执行期望增量：

$$\Delta\mathbb{E}_{cluster} = |T^*| \cdot p(T^*) + 0 \cdot (1 - p(T^*)) = |T^*|\sigma(\sum_{w} A_{w,T^*})$$

$$\Delta\mathbb{E}_{separate} = \sum_{t \in T^*} p(t)$$

如果利用在问题二的分析中所解得的平衡条件，即对于任意的t，$p(t)$为常量，取$t' \in T^*$为用户执行项目聚类T^*时第一个执行的子项目，则上式化为：

$$\Delta\mathbb{E}_{separate} = \sum_{t \in T^*} p(t) = |T^*|\sigma(\alpha \cdot v_{t'} + C_{t'})$$

在我们的概率模型中，聚类T^*比分散更优等价于$\Delta\mathbb{E}_{cluster} > \Delta\mathbb{E}_{separate}$，即：

$$\alpha \cdot \sum_{t \in T^*} v_t + C_{T^*} > \alpha \cdot v_{t'} + C_{t'} \tag{14}$$

对上式的定性分析如下：

- 上式左侧的第一项大于右侧的第一项；
- 上式左侧的第二项小于右侧的第二项，因为这一项代表了用户进行这个项目/项目聚类的总行程导致的激励，而执行项目聚类势必先执行t'后进行额外位移，导致更大的惩罚，更小的奖励。

通过一个简单的实例可以阐释这一聚类合理性判断的进行，如图6所示：

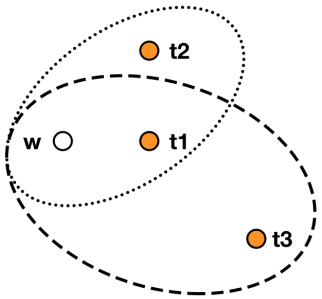

图 6: 单用户多任务聚类示例

整个环境中只有一个用户w，有三个任务t_1, t_2, t_3，此时的用户激励场只有一个发射源，故C_t的计算是直接的，再假设三个项目的定价均为pr，从图中可得到关系(注意到一般情况下只有$T_1 \in T_2$容易判断$C_{T_1} > C_{T_2}$)：

$$C_{t_1} > C_{t_2} > C_{t_3} > C_{t_1,t_2} > C_{t_1,t_3} > C_{t_2,t_3} > C_{t_1,t_2,t_3}$$

则聚类t_1与t_2合理等价于：

$$\alpha \cdot v > C_{t_1} - C_{t_1,t_2} \tag{15}$$

聚类t_1与t_3合理等价于：

$$\alpha \cdot v > C_{t_1} - C_{t_1,t_3} \tag{16}$$

由于$C_{t_1,t_2} > C_{t_1,t_3}$，所以式(15)比式(16)更容易成立，换言之，聚类t_1和t_2比聚类t_1和t_3更合理。

4.3.3 聚类算法

注意到计算$A_{w,T'}$中存在寻找哈密尔顿路，所以判断哪些集合是可打包的是NP完全问题。我们使用启发式方法来求解打包方案，从而得到使总预算减小的局部最优解。记当前的任务集合为T(被打包的任务集合当做一个任务$t \in T$，所以可以形成元素数量大于等于二的聚类)，且所有项目都有着相同的被选取概率C'，我们每次找t_i, t_j，使得合并t_i, t_j后$\Delta\mathbb{E}_{t_i,t_j} = \Delta\mathbb{E}_{cluster}(t_i, t_j) - \Delta\mathbb{E}_{separate}(t_i, t_j)$最大，即寻找：

$$argmax_{i,j}\{\alpha \sum_{t_j} v_{t_j} + C_{t_i,t_j} - C_{t_i}\} \tag{17}$$

若$\Delta\mathbb{E}_{max} = \mathbb{E}_{t_i,t_j} > 0$，说明将$t_i, t_j$打包是当前局面下可以最大程度提高$\mathbb{E}$的措施，因此将$t_i, t_j$合并成为新的$t_k$，从而得到新的任务集合$T$，一直迭代直到$\mathbb{E}_{max_{i,j}} \leq 0$为止。伪代码如1：

Algorithm 1 众包任务打包

```
Find argmax_{i,j} ΔE_{t_i,t_j}
    //枚举i, j，找到上文中所说的t_i, t_j
while ΔE_{t_i,t_j} > 0 do
    //若将t_i, t_j打包使得期望完成任务数增加则进行迭代，否则退出
    Union t_i, t_j
    //将任务t_i, t_j打包成为一个任务
    Calculate C'
    //计算打包后最佳定价下的C
    Find argmax_{i,j} ΔE_{t_i,t_j}
    //在数据更新之后再次寻找t_i, t_j
end while
return T;
    //输出最终的打包方案
```

4.3.4 计算$d(w,T)$

对于任意一个集合$T' \in T$，若要计算$A_{w,T'}$，即要求出点集$\{w,T'\}$的最短哈密尔顿路径，但由于这是一个NP完全问题，所以我们使用近似方法求出近似最短哈密尔顿路径的长度来代替$A_{w,T'}$中所要用的路径长度：

- 最小生成树使用点集$\{w,T'\}$的最小生成树，从w点沿着最小生成树进行深度优先遍历(每条边走两次)，这是最短哈密尔顿路径的两倍近似[7]。
- 链近似将打包后的点集都当成链，每次合并时将两条链首尾相连形成新的链。

4.4 问题四的分析

4.4.1 定性分析

问题四要求对于附件三中的项目作出定价方案。首先，我们在图7上标记出附件三中的项目位置：

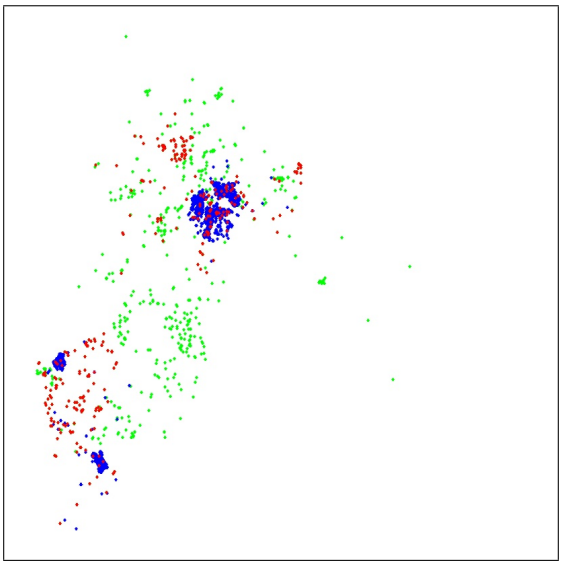

图 7: 附件一和附件三中项目的地理位置分布，绿色表示附件一中成功执行的项目，红色表示附件一中未成功执行的项目，蓝色表示附件三中的项目

可以定性地发现，附件三中给出的项目位置大都位于附件一中给出的未被完成项目附近。因此，我们可能期望附件三中项目的平均完成率会比附件一中给出的项目要更低。

4.4.2 求解方案

从问题一和问题二的求解中，我们已经得到了如下的几个模块：

- 问题一中的计价规律，即给定项目t_i地理位置的信息，使用统计方法计算v_i；

- 问题二中的评价系统S_θ，即给定$\mathbf{v}$，求解$\mathbf{z}$；
- 问题二中对于评价系统S_θ的最优化，即给定θ，C，T，求解$\mathbf{z}$。

因为我们最终希望求得附件三中项目的完成数量期望，所以可以基于上述模块设计两种独立的求解定价方案，其流程分别如图10的左图和右图所示：

在方案一中，我们使用问题一所得的计价规律来直接求解$\mathbf{v}$，并用S_θ求$\mathbf{z}$。

在方案二中，我们使用问题二所得的计价模型，代入附件三项目的整体参数C和T来求解$\mathbf{v}$，并用S_θ来求$\mathbf{z}$。

计价方案$\mathbf{v}$的最终效果评判标准仍旧来自式(2)。

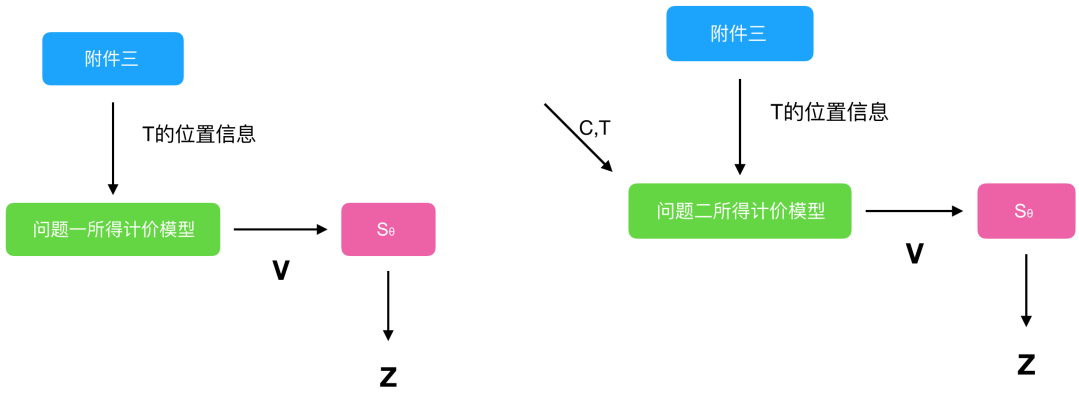

图 8: 问题四的两种定价——评估方案

5 实验模拟和解答分析

5.1 问题一的求解和分析

首先，用TensorFlow[8]对式(1)线性回归直接求解，以标准误差(Root mean squared error, Re)度量回归的效果：

$$Re_1 = 4.47$$

其次，使用各价格周边顾客密度的统计方法，分别以半径r为0.5，0.25和0.1，以项目位置为中心，统计该区域内的顾客数量，在图7中画出区域内顾客数量和平台估价的关系，其中$c(r)$代表某种标价的项目其半径为r的圆周内存在的用户数量的平均数：

为了对一个给定地理位置的项目求估价，我们取上述三个试验中最适合拟合的情况($r = 0.1$)拟合圆内用户数量和估计价格的关系：因为平台定价$pr \in [65, 85]$, 分度为0.5, 故取横轴v为标准化后的价格: $x = 2(v - 65) + 1$，纵轴y表示以任务为圆心，半径$r = 0.1$的圆内人数，我们记作$c(r)$. 对$c(r), x$进行对数和指数回归（图10）：

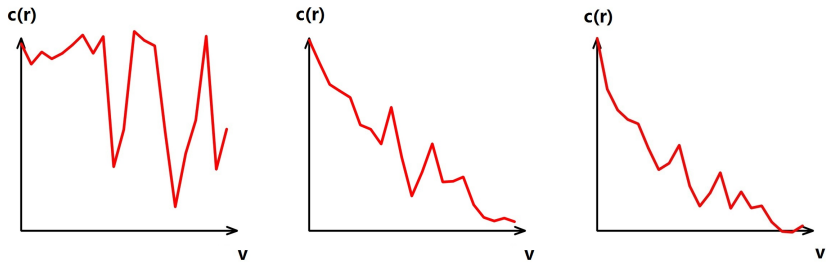

图 9: 用户数量关系 - 平台估价，r = 0.5，r = 0.25，r = 0.1

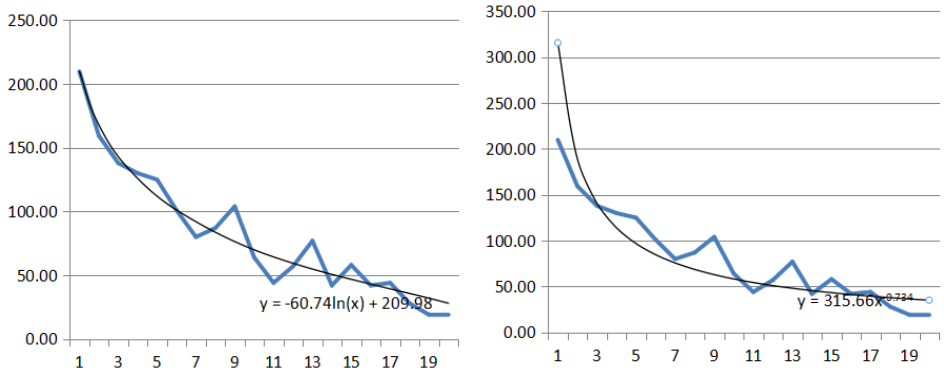

图 10: r = 0.1时任务周围人口数 - 估计任务价格的拟合

我们得到：

$$c = -60.74 \ln x + 209.98 = -60.74 \ln[2(v - 65) + 1] + 209.98 \tag{18}$$

给定新项目时，首先以$r = 0.1$求周围用户数量，再利用式(18)求得对应价格，此算法给出的二乘误差总和为：

$$Re_2 = 3.92$$

和线性回归方法相比，标准误差减小了12.3%。也可以注意到该关系的幂拟合中，价格和用户数量的-1.14次幂正相关，这一点也佐证了在5.1.2节中推测的反比例关系。

但是，注意到上述拟合函数会导致求出的估价仍然是一个连续量，通过取顶和底运算将其化归到以0.5为步长的离散值后，给出标准误差缩小为：

$$Re_3 = 3.64$$

相对于线性回归，优化幅度达到了18.8%.

5.2 问题二的求解与分析

5.2.1 S_θ参数估计

S_θ的参数估计是一个Logistic回归上求解线性参数的问题，输入数据向量(对于不对会员求和的项，没有引入乘数的必要)：

$$\mathbf{x} = (v_t, \sum_w f(w,t)^2, \sum_w f(w,t)^{\frac{1}{2}}, \sum_w f(w,t)^{\frac{1}{4}}, \epsilon)^{\mathrm{T}}$$

我们使用梯度下降法来迭代地求解。如同式(6)，记其相反数为关于参数θ的损失函数并最小化：

$$E(\theta) = -\ln H(\theta)$$

则：

$$\nabla E(\theta) = \sum_t (\sigma(\theta^{\mathrm{T}}\boldsymbol{x}) - fi_t)\boldsymbol{x}$$

该损失函数也被称为相互熵(cross entrophy)。在经过预处理提取出特征向量集合$\{\boldsymbol{x}\}$和目标向量集合(使用热洞编码)后，使用Tensorflow直接进行梯度下降，我们使用了一般的梯度下降算子，以前50的任务为训练集，以0.001为学习速率进行了1000回的迭代，最终收敛至θ_{ML}：

$$\begin{cases} \alpha &= 0.01568 \\ \beta &= -0.0049 \\ \gamma &= 0.02914 \\ \delta &= 0.00312 \\ \epsilon &= 0.00171 \end{cases} \tag{19}$$

在似然估计参数以后，我们以一种启发式的方法展示这一解的合理性，设置阈值0.53，概率在此以上的项目设置为被执行，概率在此以下的项目设置为未被执行，则得到以下结果（图11）：

$S_{\theta_{ML}}$的判断准确率为64%，不过在总体的染色趋势上与原始数据十分相似。图11的最右侧一张图片说明发生误判的项目也多处于被完成项目和未被完成项目的交界处，这说明我们的近似没有严重的系统性误差。

5.2.2 新计价方案与原方案的对比

使用6.2.1节中估计的参数，代入附录一中提取的数据，其中我们对于定价进行了预处理，即减去最小定价65.0并除以0.5，使得所有的标准定价变为整数。

用式(12)中进行计算，代入：

$$\begin{cases} C &= 6865 \\ T &= 835 \end{cases} \tag{20}$$

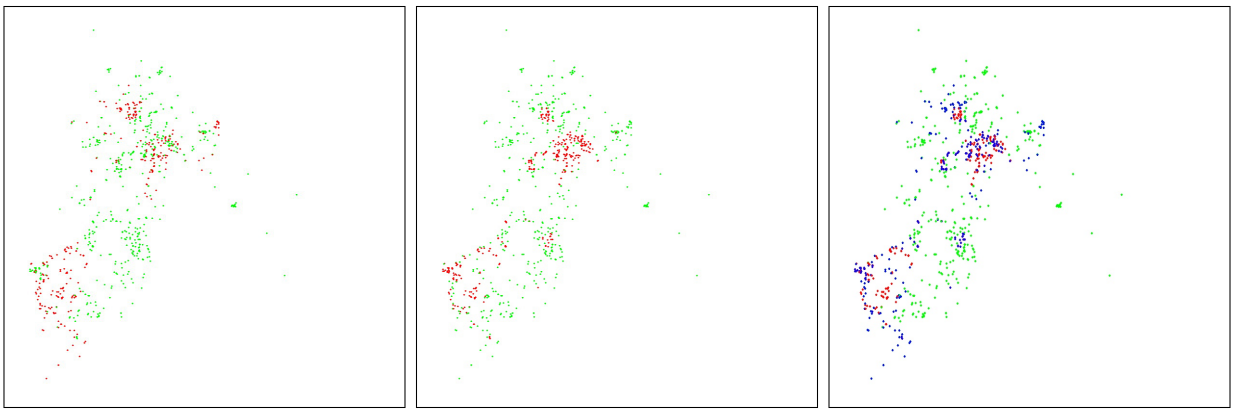

图 11: 前两张图依次为：附录一中的成败情况、S_θ拟合的成败情况。绿色代表成功，红色代表未成功。第三张图将S_θ误判点标记为蓝色

解得：

$$p(t) = 0.845201$$

此时期望的完成项目数量为$835 \cdot p(t) \approx 709.75$。

为做对比，我们还统计了附录一的原始数据中被成功完成的项目数量，以及以附录一中的$\mathbf{v}$为输入，S_θ输出的完成期望数量：

	附录一原始数据	S_θ对附录一定价方案的评价	S_θ对最优定价方案的评价
完成数量	522	499	710
完成率	62.5%	59.8%	85.0%

表 3: 完成数量期望

可见此时在项目总支出一定时，项目完成率有所提升。

因为总预算相同，所以模型求解v_t的均值和附录一中的估价均值相同，但是我们的模型得出的所有估价均处于区间$[67.8, 72.6]$之间，即拥有一个更小的震荡。这一现象符合直观预期，因为修订定价策略的最基本启发式思路就是将未被完成的项目之估价提升，将被完成的项目之估价降低。又因为未完成的项目大都定价较低，完成的项目大都定价较高，所以整个调整过程是一个将所有估价向平均值拉动的过程。如果用户的数量足够稠密，而且如果其对于项目距离的敏感性相对较低，则可以认为由这样一个用户群体构成的激励场为一个均匀场，此时项目估价的安排也应该大都相同。激励场的平均效应这可以从的图12中观察到：

5.3 问题三的模拟和对比

5.3.1 参数选取

在第一次模拟中，我们完全使用算法1，并沿用了问题二中所得的$\theta_{ML} = (\alpha, \beta, \gamma, \delta, \epsilon)^{\mathrm{T}}$，

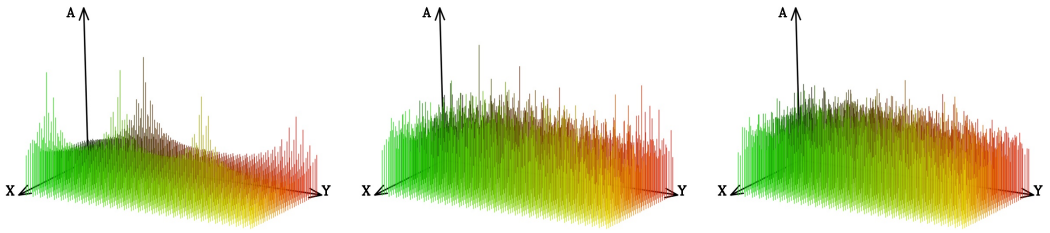

图 12: 从左到右依次是同一区域中有随机分布的5位、20位、100位用户时生成的激励场

因为该启发式算法在迭代的每一步中对于每一个点对计算聚类造成的项目完成数增量期望，所以时间复杂度为$O(|T|^2|W|)$，运行一次的时间很长。下图体现了一次启发式聚类的结果（图13）：

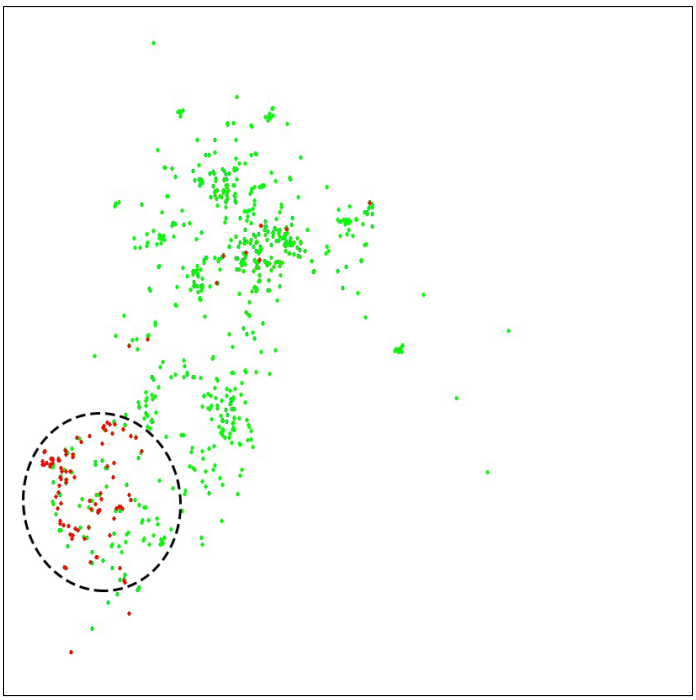

图 13: 使用问题二所求参数与启发式打包算法的聚类结果，虚线圈内的红色点代表一个聚类

可以看出，虽然聚类中的元素有地理上相邻的倾向，但是并不是所有地理上毗邻的点都被包含进聚类中，同时也存在一些虽然地理上偏远但是被划分入聚类的异常点。

我们认为这是因为问题二根据附录一数据训练出的参数并不一定描述了一个适合聚类的环境。如果存在参数使得式(17)中的$C_{t_i} - C_{t_i,t_j}$也即一次聚类从行程的角度带来的额外负荷很小，那么这组参数就描述了一个很倾向于聚类的激励场也即会员群体。

为了定性分析一个聚类倾向较高的会员群体，我们采取如下近似：

- 使用5.2.2提到的匀强激励场的假设，继而假设所有项目的估价都为v，所有项目所在处的场强影响因子$\sum_w f(w,t)^k$均相同；

- 每次只考虑向一个聚类中纳入一个新项目，不考虑项目聚类和项目聚类之间的合并；

- 考虑会员作为执行者的疲劳，认为聚类从行程角度带来的负荷与聚类的元素数量正相关，与新元素到聚类中心的距离正相关。

聚类的判断标准依然是最大化式(2)中的：

$$\alpha \sum_{t_j} v_{t_j} - (C_{t_i} - C_{t_i,t_j})$$

令t_j表示新加入聚类t_i的单个项目，则可以粗略地将上式近似于(其中C，k_1和k_2为非负常数)：

$$C - k_1 \cdot |t_i| - k_2 \cdot d(t_i, t_j) \tag{21}$$

其中$d(t_i, t_j)$度量点t_i到点集t_j的距离，比较合适的度量有取均方和、取点到点集中最近点的距离等等，在项目均匀分布的情况下，二者分别可近似为$|t_i|$的二次幂和零次幂，换言之，当聚类增长到比较大时，式(21)随着聚类的增大线性减小。

在这种近似分析下，整个会员——项目系统的聚类机制有两个很好的性质：

- 当一开始没有任何聚类关系时，所有项目中两个距离最近的首先被聚类，只需令$|t_i| = 1$代入式(21)即可，这一方面符合我们的常识判断，一方面给整个聚类过程提供了一个确定性的开始；

- 聚类的大小增长不是无限制的，因为随着$|t_i|$的增加，它带给式(21)一个线性的减益。换言之，一个聚类增长到一定大小时，聚类机制就会抵制继续将新项目聚入其中的行为，而会转而寻找新的聚类起点。其中具体的大小和式(21)中的三个系数相关。

这两点也是一个参数适合聚类的匀质模型所必备的两个性质：即聚类优先发生在距离最近的元素上、聚类的大小增长不是无限制的。

利用上述的两点性质，我们使用了DBSCAN算法，并限制了聚类大小的阈值，在当前的项目地理分布上进行了聚类的展示（图14）：

需要说明的是，虽然可以证明在参数合理的情况下，存在求聚类的启发式算法，而且聚类也确实能提高APP设计者的收益期望，但图14中的结果是只这种聚类算法的近似方案在给定数据集上的实验性结果，附录一中蕴含的系统参数并不能保证该系统是一个很适合被聚类的系统，但是上述的两点性质能确保一个可聚类的系统仍是相对比较稳定的(不会产生爆炸式增长的聚类)。

5.4 问题四模拟结果

5.4.1 方案模拟结果

方案一具体执行方案如下：

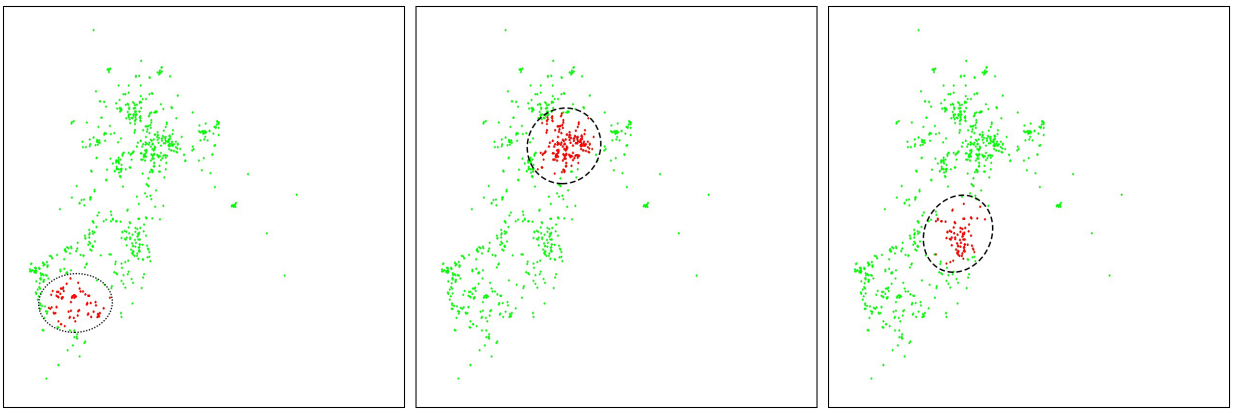

图 14: DBSCAN在附录一给定项目分布上的测试结果(未使用问题二求得的参数)

1、利用在6.1节所得的近似定量关系式(18)，对于附件三中的所有任务计算其价格向量$\mathbf{v}_1$；

2、利用6.2.1节所得的S_θ，对于$\mathbf{v}_1$求$\mathbf{z}_1$；

3、求附件三中项目被成功执行的概率：

$$\overline{p}_1 = \frac{\sum_{t \in T} z_t}{2066}$$

程序模拟得$\overline{p}_1 = 0.47932$。

方案二具体执行方案如下：

1、假设APP设计者对于附件三中项目安排的平均价格和附件一中相同，引入额外参数$T = 2066$，$C = T \cdot 8.22 = 16982$；

2、利用5.2.5节的式(14)，使用6.2.1节所得的参数θ得到最优化时项目被选取的概率$\overline{p}_2$。

也可以先利用5.2.5节的结论求得λ，继而对于附件三中的所有项目计算估价，再利用S_θ计算选取概率。

程序模拟得$\overline{p}_2 = 0.646226$。

5.4.2 结论分析

我们将附件一给出的项目完成比例、问题一模型对于附件一项目估价后计算的完成比例、问题二模型对于附件一项目估价后计算的完成比例，以及问题一和二的模型对于附件三中项目的完成比例作图15

可以发现两点结论：

- 问题二模型估计的成功执行比例高于问题一模型估计的成功比例，这是因为问题二的模型专门就成功比例进行了最优化；

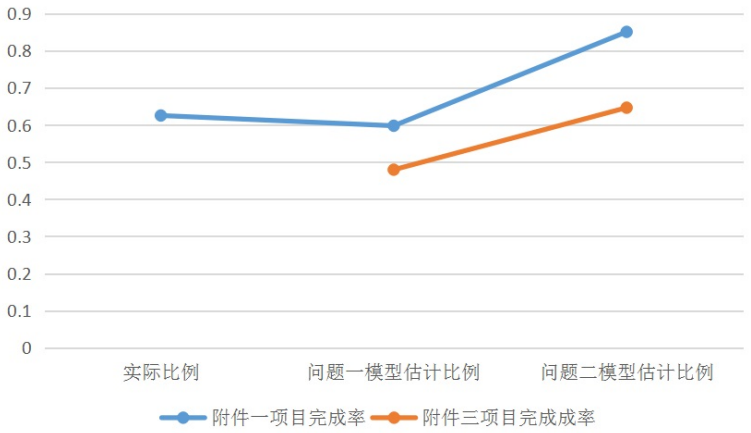

图 15: 问题一和问题二模型对应的的贝叶斯网络，蓝色和绿色分别代表已知量和未知量

- 问题一和问题二模型对于附件三中项目的完成率的估计均低于对附件一的估计，这是因为如同图7所示，附件三给出的项目位置大都集中于项目完成率偏低的区域。

6 模型总结

6.1 模型优点

在这一节中，我们将证明我们的概率模型在测试中产生较好的效果不仅仅是一个偶然，因为这个模型本身是EM算法的一种退化近似，而EM算法本身也适用于这个问题，这是由于本问题的本质是一个含有隐含元的复杂概率系统的参数推断。

考虑EM算法的一般应用场景：给定模型输出的可观测集合$\mathbf{X}$，试求模型的参数μ，这是在对于：

$$p(\mu|\mathbf{X})$$

求值，不过复杂的系统会使得这一条件概率难以直接求出，故引入中间变量，也即隐含元（Latent Variables）使得模型简化。引入合适的隐含元θ将使得以下两个条件概率容易求得：

$$p(\theta|\mathbf{X},\mu) \quad p(\mathbf{X}|\theta,\mu)$$

一个完整的EM算法执行过程从随机初始化μ_0开始，首先对$p(\theta|\mathbf{X},\mu_0)$求值，并将其作为$\theta_0$的分布，继而调整$\mu$的值从$\mu_0$到$\mu_1$使得似然概率$p(\mathbf{X}|\theta_0,\mu_1)$的期望最大化。

在本次建模问题中，我们遇到的主要困难也正是系统的复杂性，它使得单纯的参数估计方法不够优秀。下图16给出了问题一模型和问题二模型的流程图，也即其对应的贝叶斯网络。因为该流程图无环，所以有效的信息随着其中的箭头方向单向地流动。可见问题二模型相对于问题一模型的改进就是对于用户行为更精细的模拟，为了进行这些模拟而额外引进的参数即是这个模型的隐含变量，所以问题二的概率模型本质上是一个EM算法。

在这个问题中，我们将：

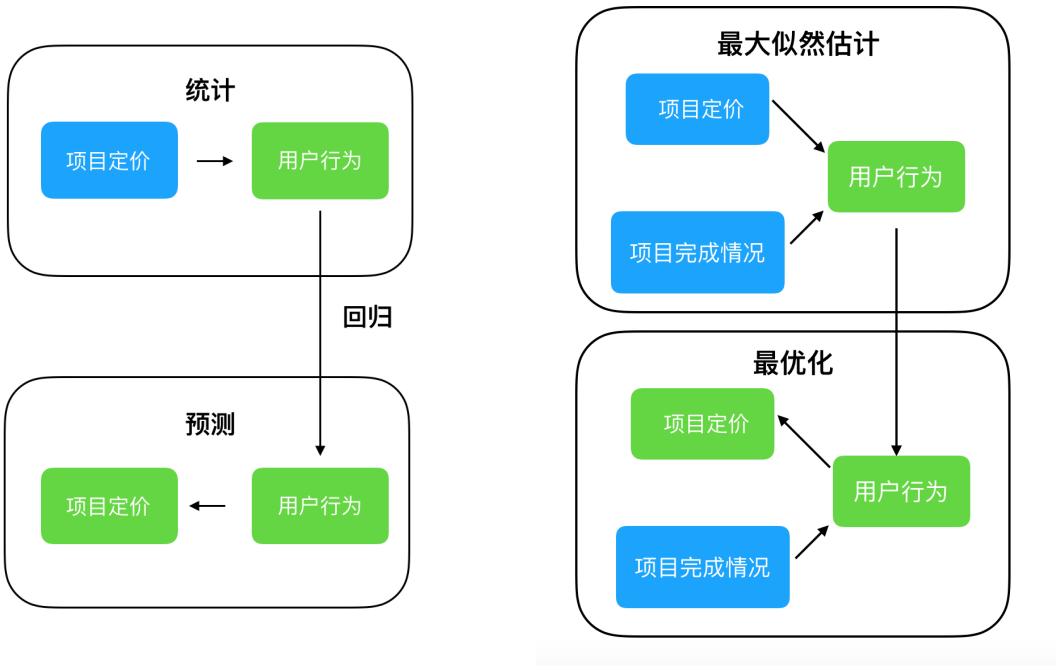

图 16: 问题一和问题二模型对应的的贝叶斯网络，蓝色和绿色分别代表已知量和未知量

- 项目的完成情况视作可观测集合$\mathbf{X}$，因为我们每每根据给定的完成情况推断系统参数，以及对完成情况进行假设并反推系统状态；
- 项目的计价策略视作模型的目标参数μ，它是我们想要求得的值；
- 用户的行为模式模型的参数视作隐含变量θ，它并非是我们必须求得的值，但却是影响系统的重要变量，并且和$\mathbf{X}$与μ深深关联着。

根据我们在问题二中做出的激励场假设，条件概率$p(\theta|\mathbf{X},\mu)$，即式(??)，可套用逻辑回归理论进行最大似然估计，所以我们直接选择最大似然估计的结果θ_{ML}来近似取代了对于整个分布的求解。这就是我们的模型相对于标准EM算法所进行的近似——和多次迭代至收敛不同，我们选择只进行一次最大似然估计。

另一个条件概率$p(\mu|\theta,\mathbf{X})$的求解就是另一个最大似然估计的过程，不过由于$\mathbf{X}$的匀质性，在我们的模型中，它被转化为了一个可以利用拉格朗日乘子法进行的最优化过程。

引入隐含元简化求值的思路是我们建立主要模型的动机，而EM算法正是求解带有隐含元的概率模型的通用方法。虽然我们通过合适的近似和假设使得优化的过程由多次迭代转化为一次最大似然估计，但是EM算法的合理性本身即是我们的主要模型最坚实的理论依据。

6.2 模型缺点

1. 根据假设1，我们假设所有项目对APP公司而言是同等重要的。虽然这一点可以通过调整式(4)中的期望计算来进行放宽，但我们没有从理论角度对这一点进行考察；

2. 根据假设2、3、4，我们认为用户之间的行为是相互独立的，然而实际上用户的行为可能受制于其他用户的影响而产生质变，这一点是概率模型无法准确捕捉的；

3. 我们的概率模型没有对于用户、项目的具体指派进行离散的假设，这使得它只能给出统计意义上的解，而不能进行细微的具体预测；

4. 我们的模型没有对于用户的选取项目开始时间进行准确的模拟，而是将其加算入激励场中，这一近似不一定是合理的。

5. 虽然采用了EM算法的思路，但是我们无法使用标准EM的迭代算法，取而代之的最大似然估计将不可避免地面临过拟合的缺陷。

参考文献

[1] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," pp. 505–506, 2016, http://www.deeplearningbook.org.

[2] D. Yang, G. Xue, X. Fang, and J. Tang, "Incentive mechanisms for crowdsensing: crowdsourcing with smartphones," pp. 1732–1744, 2016.

[3] H. Wang, S. Guo, J. Cao, and M. Guo, "Melody: A long-term dynamic quality-aware incentive mechanism for crowdsourcing," pp. 933–943, June 2017.

[4] B. Morschheuser, J. Hamari, and J. Koivisto, "Gamification in crowdsourcing: A review," pp. 4375–4384, 2016.

[5] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," pp. 100–108, 1979.

[6] M. Ester, "Density-based clustering," New York, NY, pp. 1–6, 2016. [Online]. Available: https://doi.org/10.1007/978-1-4899-7993-3_605-2

[7] V. V. Vazirani, "Approximation algorithms," pp. 28–30, 2001.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "Tensorflow: a system for large-scale machine learning," 2016.

A 启发式任务打包算法代码（C++）

```
#include <cstdio>
#include <algorithm>
#include <cmath>
using namespace std;
struct Task{
    double latitude;
    double longitude;
    double price;
    int isFinished;
}task[1000];
int taskSize = 835;
struct Member{
    double latitude;
    double longitude;
    double credit;
}member[2000];
int memberSize = 1877;
int map[1000][2000], f[1000];
double dis[2000], dd[1000][2000];
int quota[2000];
int find(int now){
    if(f[now] != now)f[now] = find(f[now]);
    return f[now];
}

double alpha = 0.0157, beta = -0.0026, gam = 0.03, eps = 0.0017;
double dist(double x1, double y1, double x2, double y2){
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
double get(int i, int j){
    return dist(task[i].longitude, task[i].latitude, task[j].
       longitude, task[j].latitude);
}
double get1(int i, int j){
    return dist(member[i].longitude, member[i].latitude, task[j].
       longitude, task[j].latitude);
}
int main(){
        freopen("task.in","r",stdin);
    for(int i = 1; i <= taskSize; i++)
```

```
        scanf("%lf%lf%lf%d", &task[i].longitude, &task[i].latitude
            , &task[i].price, &task[i].isFinished)
        , f[i] = i, dis[i] = 0;
    for(int i = 1; i <= memberSize; i++)
        scanf("%lf%lf", &member[i].longitude, &member[i].latitude)
            ;
    for(int i = 1; i <= taskSize; i++)for(int j = 1; j <=
       memberSize; j++)dd[i][j] = get1(j, i);
    puts("b");
    for(int i = 1; i <= memberSize; i++){
    //  printf("%d\n",i);
                for(int j = 1; j <= taskSize; j++){
                int tmp = 0;
        for(int k = 1; k <= taskSize; k++)if(dd[j][i]>dd[k][i])tmp
            ++;
        if(tmp <= 1)map[i][j] = 1;
                }
        }
    bool flag = true;
    int tot = 0;
        while(flag){
        tot++;
        printf("%d\n",tot);
        flag = false;
        for(int i = 1; i <= taskSize; i++){
                //if(i % 10 == 0)printf("%d\n", i);
                double min1 = 1111111, qq = 0;
            for(int j = 1; j <= taskSize; j++){
                double tmp = 0;
                int x = find(i), y = find(j);
                if(x==y)continue;
                for(int k = 1; k <= memberSize; k++){
                        tmp += alpha  + gam / get(i,j) + min(get1(
                            k, x), get1(k, y) - gam / min(dis[x]+
                            get1(k,i),
                                        dis[y]+get1(k,j));
                                }
                                if(tmp < min1){
                                        min1 = tmp;
                                        qq = y;
                                }
            }
            printf("%lf\n", min1);
            if(min1 < 0){
                int x = find(i);
```

```
                int y = find(qq);
                dis[x] += dis[y] + get(x,y);
                task[x].price += task[y].price;
                f[y] = x;
                printf("%d %d\n", x, y);
                        }
        }
    }
    freopen("niu.txt","w",stdout);
    for(int i = 1; i<= taskSize; i++)if(f[i] == i)
        {
                int tmp = 0;
                for(int j = 1; j <= taskSize; j++)if(find(j) == i)
                   tmp++;
                printf("%d ",tmp);
                for(int j = 1; j <= taskSize; j++)if(find(j) == i)
                   printf("%d %lf ",j,task[j].price);
                puts("");
        }
}
```

B Tensorflow代码实例——线性回归

```
import tensorflow as tf
x = tf.constant([read from files])
y = tf.constant([read from files])
xt = tf.transpose(x)

xtx = tf.matmul(xt, x)

xtxi = tf.matrix_inverse(xtx)

w = tf.matmul(tf.matmul(xtxi, xt), y)

sess = tf.Session()

print(sess.run(w))

predict = tf.matmul(x, w)

print(sess.run(y))

print(sess.run(predict))
```

```
print(sess.run(tf.reduce_sum(tf.square(tf.matmul(x, w) - y))))
```

C Tensorflow代码实例——逻辑回归

```
import tensorflow as tf

x = tf.constant([read from files])

y = tf.constant([read from fiels])


W = tf.Variable(tf.random_normal())

b = tf.Variable(tf.random_normal())


sess = tf.Session()

sess.run(tf.initialize_all_variables())

p = tf.softmax(tf.matmul(x, W) + b)

loss = -tf.reduce_sum(y * tf.log(p))

train_step = tf.train.GradientDescentOptimizer(alpha).minimize(
    loss)

for step in range(times):

        sess.run(train_step)

        if (step % 100) == 0:

                print(sess.run(loss))
```

D 问题四的求解与可视化代码

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <vector>
#include <unistd.h>
#include <stdlib.h>
#include <map>
#include <random>
#include <utility>
```

```
#include <fstream>
#include <stdio.h>

using namespace cv;
using namespace std;
class task
{
public:
        float x;
        float y;
        float price;
        bool su;
        task(float x_, float y_, float p_, bool s) : x(x_ - 22.4),
            y(y_ - 112.6), price((p_ - 65.0) * 2), su(s) {};
};

class worker
{
public:
        float x;
        float y;
        int up;
        worker(float x_, float y_, int u) : x(x_ - 22.4), y(y_ -
           112.6), up(u) {};
};

float distance(const task& t, const worker& w)
{
        return sqrt((t.x - w.x) * (t.x - w.x) + (t.y - w.y) * (t.y
            - w.y));
}

float distance(const worker& w, const task& t)
{
        return sqrt((t.x - w.x) * (t.x - w.x) + (t.y - w.y) * (t.y
            - w.y));
}

float distance(const task& t1, const task& t2)
{
        return sqrt((t1.x - t2.x) * (t1.x - t2.x) + (t1.y - t2.y)
           * (t1.y - t2.y));
}

extern int HEIGHT;
```

```
extern int WIDTH;
extern float INIT_X;
extern float END_X;
extern float INIT_Y;
extern float END_Y;
extern float DELTA;
extern Scalar BLACK;
extern Scalar WHITE;
extern Scalar RED;
extern Scalar BLUE;
extern Scalar GREEN;
extern Scalar PURPLE;
extern Scalar YELLOW;
extern Scalar BROWN;

extern float y1(float);
extern float y2(float);
extern float y3(float);
extern float y4(float);
extern float y5(float);
extern float y6(float);
map<float, float> m1;
map<float, float> m2;
map<float, float> m3;
map<float, float> m4;
map<float, float> m5;
extern float z1(float, float);
extern float z2(float, float);
extern float z3(float, float);
extern float noise();

void draw_line(Mat, const Point&, const Point&, const Scalar&, int
    thickness = 3);

void draw_point(Mat, const Point&, const Scalar&, int thickness =
   9);

void draw_dashed(Mat, const Point&, const Point&, const Scalar&,
   int segmentation = 20, int thickness = 3);

void draw_segment(Mat, const Point&, const Point&, const Scalar&,
   int thickness1 = 3, int thickness2 = 9);

void draw_arrow(Mat, Point, Point, const Scalar&, int len = 30,
   int alpha = 20, int thickness = 3, int lineType = CV_AA);
```

```
void XY_display(Mat, const Scalar&, float (*y)(float), const
   Scalar&, float, float, float, float, float, bool seg = false,
   float r1 = 1.0 / 6.0, float r2 = 5.0 / 6.0, float r3 = 1.0 /
   8.0, float r4 = 7.0 / 8.0);

void XY_animate(Mat, int, const Scalar&, float (*y)(float), const
   Scalar&, float, float, float, float, float, bool seg = false,
   float r1 = 1.0 / 6.0, float r2 = 5.0 / 6.0, float r3 = 1.0 /
   8.0, float r4 = 7.0 / 8.0);

void XY_map(Mat, const Scalar&, map<float, float>&, const Scalar&,
    float, float, float, float, float, bool seg = false, float r1
   = 1.0 / 6.0, float r2 = 5.0 / 6.0, float r3 = 1.0 / 8.0, float
   r4 = 7.0 / 8.0);

void XYZ_display(Mat, const Scalar&, const Point&, const Point&,
   const Point&, const Point&, const Scalar&, float (*f)(float,
   float), float end_x = 5.0, float end_y = 5.0, float end_z =
   1.0, float delta_x = 0.15, float delta_y = 0.15);

void XYZ_animate(Mat, const Scalar&, const Point&, const Point&,
   const Point&, const Point&, const Scalar&, float (*f)(float,
   float), float end_x = 5.0, float end_y = 5.0, float end_z =
   5.0, float delta_x = 0.1, float delta_y = 0.1);

void XYZ_point(Mat src, const vector<float>& p, const Scalar&
   color1, const Point& o, const Point& ox, const Point& oy, const
    Point& oz, const Scalar& color2, float end_x = 5.0, float
   end_y = 5.0, float end_z = 5.0);

void XYZ_point2(Mat src, const vector<float>& p, const Scalar&
   color1, const Point& o, const Point& ox, const Point& oy, const
    Point& oz, const Scalar& color2, float end_x = 5.0, float
   end_y = 5.0, float end_z = 5.0);

int main()
{
        Mat canvas(800, 800, CV_8UC3, WHITE);
        Mat can(600, 800, CV_8UC3, WHITE);
        vector<task> tasks;
        tasks.clear();
        vector<worker> workers;
        workers.clear();
        vector<task> new_tasks;
```

```
new_tasks.clear();
ifstream is;
is.open("file1.txt");
float a, b, c, d, e;
float min = 10.0, max = -1.0;
while (is)
{
        is >> a >> b >> c >> d;
        tasks.push_back(task(a, b, c, d));
}
is.close();
is.open("file2.txt");
while (is)
{
        is >> a >> b >> c;
        workers.push_back(worker(a, b, c));
}
is.close();
is.open("file3a.txt");
while (is)
{
        is >> a >> b >> c >> d;
        new_tasks.push_back(task(a, b, c, d));
}
tasks.pop_back();
workers.pop_back();
new_tasks.pop_back();
is.close();
//predict whether a task be done
//utility threshold for a worker
float loss = 0.0;
vector<float> grades_for_tasks;
float measure;
int wrong = 0;
int su = 0;
float sum = 0.0;
vector<float> ct;
ct.clear();

//S_{\theta}
float sum_p = 0.0;
for (int i = 0; i < new_tasks.size(); ++i)
{
        measure = 0.0;
        measure += 0.01568 * new_tasks[i].price;
```

```
        float temp = 0.0;
        for (int j = 0; j < workers.size(); ++j)
        {
                temp += (1900 - j) / (500.0) * (1.0 /
                   distance(new_tasks[i], workers[j]));
        }
        temp /= 200.0;
        measure += temp * -0.0049;
        temp = sqrt(temp);
        measure += temp * 0.02914;
        temp = sqrt(temp);
        measure += temp * 0.00312;
        measure += 0.00171;
        sum += (measure - 0.01568 * tasks[i].price);
        ct.push_back(measure - 0.01568 * new_tasks[i].
           price);
//      cout << (1.0 / (1 + exp(0.0 - measure))) << ":" <<
    tasks[i].su << endl;
        measure = 1.0 / (1 + exp(0.0 - measure));
        sum_p += measure;
}
sum_p /= new_tasks.size();
cout << sum_p << endl;
float fu = 0.0;
for (int i = 0; i < new_tasks.size(); ++i)
{
        fuck += ct[i];
}
cout << "fu:" << fu << endl;
fuck = (16982 + fu / 0.01568) / 2066;
cout << "final" << (1.0 / (1 + exp(0.0 - fuck))) << endl;

for (int i = 0; i < tasks.size(); ++i)
{
        draw_point(canvas, Point(tasks[i].x * 400, tasks[i
           ].y * 400), GREEN, 3);
}
for (int i = 0; i < new_tasks.size(); ++i)
{
        draw_point(canvas, Point(new_tasks[i].x * 400,
           new_tasks[i].y * 400), Scalar(255, 0, 0), 3);
}
for (int i = 0; i < tasks.size(); ++i)
{
        if (!tasks[i].su)
```

```
            draw_point(canvas, Point(tasks[i].x * 400, tasks[i
                ].y * 400), RED, 3);
        }
        imshow("X", canvas);
        waitKey(0);
        imwrite("Q4.jpg", canvas);
        return 0;
}
```