基于数据挖掘和蒙特卡罗模拟的“拍照赚钱”APP 定价策略

摘要：为了研究任务定价与完成情况，本文首先运用数据挖掘思想，使用 Python、Matlab、Stata 等软件处理附件一、附件二中的任务和会员数据，并进行了数据可视化。由可视化结果，我们发现了任务定价与任务和会员密度的潜在联系，并以函数关系构建了任务的定价模型。

通过对未完成任务的聚类现象的分析，我们建立了基于会员行为的假设模型，并以此模型建立蒙特卡罗算法的模拟程序。在该程序基础上，我们加入了基于机器学习思想的反馈优化环节，并根据模拟结果分析了任务未完成的原因。

针对不同的任务未完成原因，我们建立了不同的新定价策略并用蒙特卡罗模型进行模拟验证。最后，我们分析了模型的特点与不足，对任务捆绑策略和新任务定价进行了分析和模拟，并向 APP 运营商和开发团队提出了主要改进方向。

关键词：数据挖掘 回归分析 距离贡献模型 价格歧视 蒙特卡罗算法 机器学习

一、问题重述

1.1 问题背景——自助式众包服务

移动互联网时代的到来给许多传统线下行业带来了新的曙光，企业进行商品检查、信息搜集的调研业正是其中之一。传统调研业虽然需求量大，但一直存在着规模大、人员庞杂、流动性强的问题，导致管理困难、价格高昂、效率低下。若能用移动平台替代企业的调研执行链，将调研劳务众包给全国大量的闲置劳动力，可能达到高效率、低成本、高质量、数据真实性强的运营效果，甚至产生规模化效应。这种劳务众包平台在为企业节省大笔资金的同时，也给闲置劳动力提供了赚取外快甚至谋生的机会，是一个具有潜力的市场。

“拍照赚钱”APP 提供了这样一种自助式的劳务众包模式。会员下载 APP，注册成为会员，从 APP 上领取需要拍照的任务，赚取 APP 对任务所标定的报酬。基于劳务人员和调研任务的供求关系，该 APP 的核心要素便是任务定价。如果定价不合理，有的任务就会无人问津，导致商品检查失败。

因此，探讨任务定价的模型，探究其定价规律、优化定价方案，对企业的会员累积、运作效果以及会员的预定决策都有着实际的价值。

1.2 需解决的问题

（1） 研究附件一中项目的任务定价规律，分析任务未完成的原因。

（2） 为附件一中的项目设计新的任务定价方案，并和原方案进行比较。

（3） 实际情况下，多个任务可能因为位置比较集中，导致会员会争相选择，一种考虑是将这些任务联合在一起打包发布。在这种考虑下，如何修改前面的定价模型，对最终的任务完成情况又有什么影响？

(4) 对附件三中的新项目给出你的任务定价方案，并评价该方案的实施效果。

二、问题一的分析

2.1 任务定价规律——距离贡献模型

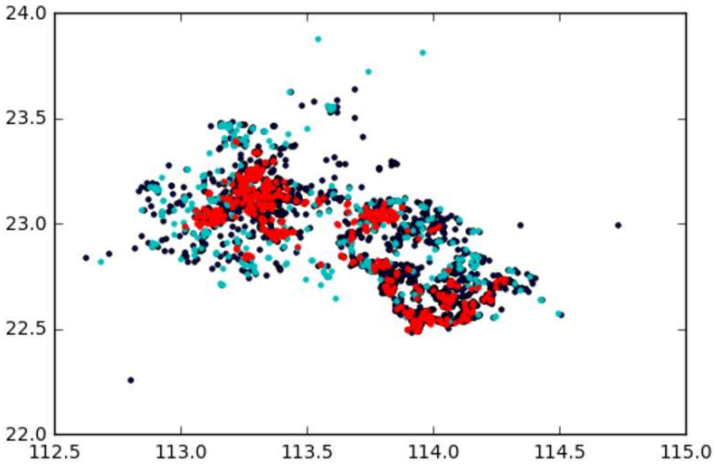
2.1.1 GPS 数据可视化

GIS（Geographic Information System 地理信息系统）是综合地理学与地

图学的一门新兴学科，依托计算机系统进行地理数据的输入、存储、查询、分析、计算、建模和可视化。我们利用 GIS **技术**，将附件一的 GPS 数据可视化得到**任务位置分布图**。从图一中可以看出附件一给出的任务所在区域为**珠三角地区**，主要覆盖广州、东莞、佛山、深圳几个城市，在广州郊区（从化区）也有零星分布。

图一 任务位置分布图

接下来，我们又用 Python 自带的 Matplotlib 库的 pyplot 方法将附件一中的任务位置分布和附件二中的会员位置分布的画在同一张散点图上，其中会员的数据进行过清理，删去了 11 个明显不在珠三角地区的数据点。图二深蓝色为会员数据点，青蓝色为高价（≥70 元）任务点，红色为低价（<70 元任务点）。



图二 任务和会员地理位置分布图

可以发现低价任务周围会员密集，高价任务周围会员相对稀疏。而且有同性相吸的特点——低价任务的周围有更多低价任务，高价任务的周围有更多高价任务。

2.1.2 定价规律逻辑分析

与滴滴出行、饿了么等APP对其提供的服务进行定价相反，“拍照赚钱”APP是一种劳务租售市场，它通过对其招标的劳务进行定价，从而吸引劳动力。这类APP的目标是在吸引更多会员预订和完成任务的同时降低企业和APP运营商的成本。参考Uber的动态提价（Surge Pricing），即供需不平衡时顾客可以通过提高价格来吸引司机接单，“拍照赚钱”APP 在定价时也会对同一项目的不同任务进行“**价格歧视**”，在劳动力可能不足的地方通过提高价格来吸引会员接单。

因此我们分析“拍照赚钱”APP的定价可能遵循以下两点：

（1） **会员密度越大，定价越低。**在会员密集的地方，任务更有可能一呼百应；在会员稀疏的地方，任务更有可能无人问津。对于没那么“抢手”的任务，APP更需要通过提高价格来吸引会员完成。

（2） **任务密度越大，定价越低。**在任务密集的地方，会员可以从APP得知更多“离我最近”的任务，而且由于任务相距不远，会员可以“顺手”做任务，那么每个会员可以做更多的任务，从而增加了这些任务被完成的几率。

2.1.3 模型建立

基于以上两点，我们建立了一个“**距离贡献模型**”。由于题目给出的任务和会员地理位置都基本分布在珠三角地区，范围不大且接近赤道，可以将球面距离近似为平面距离，且假设纬度一度的距离和经度一度的距离相等（实际情况中后者是前者的$\cos\theta$，θ表示纬度，θ越接近于0，$\cos\theta$就越接近于1）。对于任意两个地理位置a,b,定义他们之间的距离$d(a,b)$为a,b间的**欧拉距离**，即假设a的经纬度坐标为(x_1,y_1)，b的经纬度坐标为(x_2,y_2)，则

$$d(a,b)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

$d(a,b)$在数值上每相差1，实际距离相差约$111\sqrt{2}=127$公里。

我们利用距离定义a,b的相互**密集程度贡献值**$w(a,b)$,距离越近，贡献越大，当距离远到一个阈值时，贡献值即为0，关系式为

$$w(a,b)=\begin{cases}0.05-d(a,b), & 0<d(a,b)<0.05\\ 0, & \text{否则}\end{cases}$$

其中0.05即前述阈值，相当于实际距离约6公里，大致符合APP的推荐任务距离范围和会员可接受的任务距离范围。

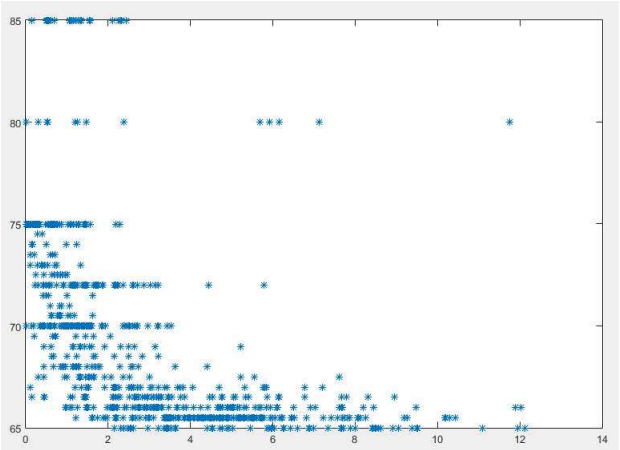
我们定义了一个**任务中心性指标F(·)**反映该任务周围的任务密度和会员密度共同作用下的“抢手程度”。将所有任务的地理位置集合记为T(Task)，所有会员的地理位置集合记为M(Member)，那么对任意的任务t ∈ T，定义

$$F(t) = \sum_{t' \in T} w(t, t') + 4 \sum_{m \in M} w(t, m)$$

其中第一项为其他任务对此任务的距离贡献之和，即任务密度；第二项为其他会员对此任务的距离贡献之和，即会员密度。两项的权重为1:4，原因在于会员密度相较于任务密度有更大的影响力。

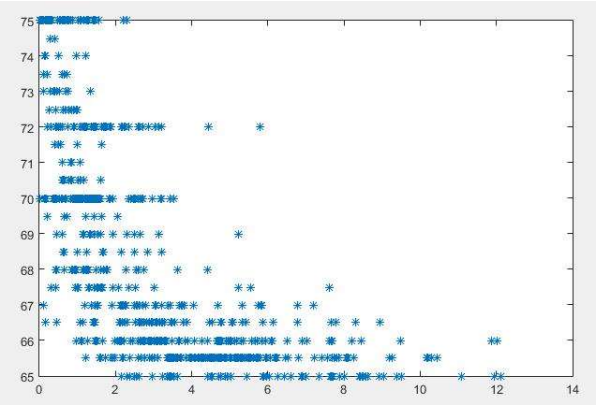
2.1.4 模型验证

我们对附件一中的任务依次计算其任务中心性，用 Matlab 软件**对任务中心性指标和任务标价进行回归分析**。图三基本呈现了**任务标价越低，中心性越高**的趋势。



图三 任务中心性和标价回归分析图

拟合优度判定系数(R^2)的值为0.5212，表明我们的模型对任务标价规律的拟合程度较好。但我们发现部分标价为 80 元和 85 元的任务偏离趋势，推测原因可能是一些任务中心性较高的任务由于其他因素（如商场大小、交通状况、任务难度等）被提高了价格。比如编号为 A0477、A0478、A0481 这几个 80 元的任务分布在任务和会员均很密集的广州市区，其周围也遍布低价任务，这一局部差异性表明这些任务很有可能是被特别加价的。为了验证我们的推测，又重新对标价为 65 至 75 元的任务进行回归分析，R^2则提高至0.6590，显著性增强（图四）。

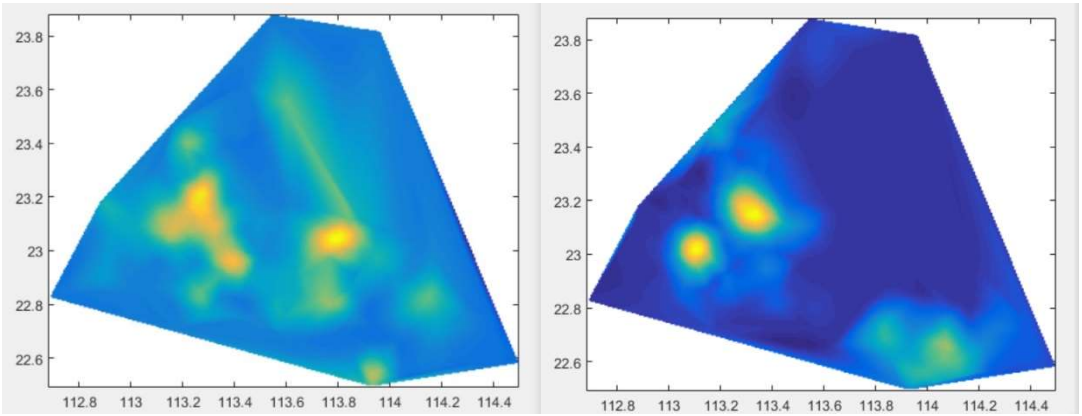


图四 任务中心性和标价（65-75 元）回归分析图

2.2 任务未完成的原因——蒙特卡罗学习模型

2.2.1 任务聚类现象

我们用 Python 将已完成和未完成任务分类，使用 Matlab 软件对已完成任务和未完成任务分别绘制了**任务中心性分布热力图**（任务中心性指标与 2.1.3 定义一致），即图五。已完成任务在广州市西北、东南部和东莞市呈现出较高的聚类现象，从化区、深圳北部密度较低，但聚类仍旧十分明显；未完成任务在广州东北、西南部呈现出很高的聚类分布，在深圳南部也呈现出密度较低的聚类分布。总体来说，任务依据其完成情况呈现出明显的聚类效应。



已完成任务　　未完成任务

图五 任务中心性热力分布图

为进一步研究任务的地域性分布情况，我们用 Matlab 软件，将 2.1.2 中的任务中心性指标绘制成**双向散点图**，其中每个点代表一个任务，红点代表已完成任务，绿点代表未完成任务。横轴表示每个任务点周围的已完成任务密度，纵轴表示任务周围的未完成任务密度。

我们统计得出，对所有已完成任务，只有60 / 522，即约11.5%的任务落

在了对角线$y = x$上方，表明未完成任务周围未完成任务更多；对所有未完成任务，只有$86 / 313$，即约27.5%的任务落在了$y = x$下方，表明已完成任务周围已完成任务更多。

尤其值得注意的是，横轴上分布了众多红色任务点，它们周围几乎没有未完成的任务；相应地，纵轴上的未完成任务周围也几乎没有已完成任务。这与我们从热力图观察到的聚类效应相符。

2.2.2 模型假设

我们对会员的行为和质量分别进行假设。

（1）会员行为假设

由于附件二中不同会员有不同的任务开始预定时间，且信誉值越高可以越优先挑选任务，所以我们假设会员按照信誉值从高到低对周围的任务进行决策，一旦决定完成某个任务，又在实际执行任务时面临完不成的可能。没有被完成的任务会在一段时间后重新放出供所有会员进行选择。因此会员的行为可以用**选择任务**和**执行任务**两个阶段来描述。

a.选择任务阶段

会员在 APP 中查看任务信息，了解周围任务的报酬（reward）与任务到自己的距离（distance），通过这两个参数计算评价函数并挑选出一个最优任务。

对这一阶段，我们做出了两个假设：一、拥有更高信誉值的会员能更先进行选择；二、会员事先并不知晓所选任务的真实难度。

b.执行任务阶段

该阶段会员已接受任务，并到达任务地点开始拍照。我们假定每个任务都有一个相应的难度系数，难度越大的任务越有可能被中途放弃。同时，会员也受到任务报酬的激励，越高的报酬意味着会员越倾向于完成任务。

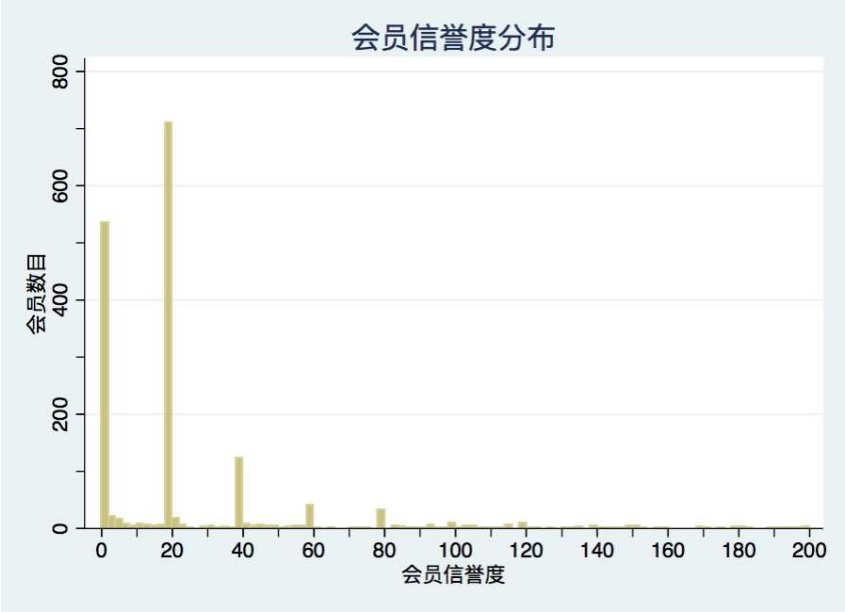
考虑到更真实的会员心理，在会员放弃一个任务时，他前往任务地点所花的时间将全部白费。也即如果这个任务离会员较远，他在决定是否放弃时，可能会考虑到自己已经投入的时间成本，而变得更倾向于完成任务。因此，我们可以假定，离会员越远的任务在执行阶段越不容易被放弃。

假如一个任务最终被放弃，我们假设被放弃的任务会被重新放出供其他会员选择，在一定的选择轮数后仍未完成的任务才会最终被视为未完成。

（2）会员质量假设

在第一步假设中，我们对所有会员一视同仁，但实际上会员的活跃度存在着巨大的个体差异。

为了研究这个问题，我们用 Stata 软件将附件二中的会员信息绘制成频数分布直方图（图二）。为了更清晰地显示主要数据的频数分布，我们进行了数据处理，删除了信誉度高于200的观测值（被删除的各区间平均占比不到0.001%）。图中，横轴代表会员信誉度的数值（以 2 为一个区间），纵轴代表该区间的会员频数。可以看出，大部分会员的信誉度位于[0,2]，[18,20]的区间内，[38,40]也分布有较多观测值。进一步观察原始数据表格，发现[0,2]区间内，信誉度大多取值为1.9923和2，[18,20]区间内的信誉度几乎全部相等，取值为19.9231。



图六 会员信誉度分布直方图

APP 开发初期，下载 APP 的会员增量最大，但会员粘度低，这些会员中有相当一部分会随时间流失，还有一部分活跃度降低，成为“隐身会员”。基于前面的发现和我们的生活经验，推测“拍照赚钱”APP 存在一个初始信誉度值，从而可以将信誉度落在[0,2]区间的观测值归为已流失会员，位于[18,20]区间的观测值归为没有完成任务记录的新会员，位于[38,40]区间的观测值归为低活跃度会员。

因此在模型中，我们扩充了分类区间，将会员大致分为 4 类：

信誉度在$[40,+\infty]$：主要会员，经常打开 APP 查看任务，概率记为p_1；

信誉度在$[20,40]$：完成任务较少的低活跃度会员，偶尔打开 APP 查看任务，概率记为p_2；

信誉度在$[18,20]$： 没有任务记录的新会员，很少打开 APP 查看任务，概率记为p_3；

信誉度在$(0,18)$：可能直接放弃第一次任务的已流失会员，几乎不打开 APP 查看任务，概率记为p_4。

2.2.3 模型建立

在这一小节里，我们将详细阐释如何通过蒙特卡罗算法模拟会员经历的选择任务和执行任务这两个阶段。

（1）选择任务阶段

在每轮放出任务时，活跃度不同的会员有不同的概率打开 APP 查看可以做的任务。对于 2.2.2 提到的四类会员，概率分别为p_1, p_2, p_3, p_4，满足$p_1 \geq p_2 \geq p_3 \geq p_4$。

我们用一个**评价函数**score(·, ·)来刻画任务的价格和距离对会员效用的贡献。会员i对任务j的评价为

$$\text{score}(i,j) = \text{reward}(j) - k * \text{distance}(i,j)$$

其中reward(j)代表任务j本身的报酬（标价），distance(i,j)代表i,j之间的距离（即 2.1.3 的欧拉距离），k是距离厌恶系数，k值越大，表示会员越不喜欢离自己更远的任务。这一假设是考虑到会员对前往距离所花费的时间计算其**机会成本**。

当前会员i将会选择能最大化评价函数score(i,j)的任务j。

（2）执行任务阶段

根据前面的假设，一个任务j被完成的概率取决于任务j本身的难度difficulty(j)，任务j的报酬reward(j)和会员i到任务j的距离distance(i,j)。

对于任务j,定义其**完成度函数**为

$$\text{complete}(j) = a * \text{reward}(j) + b * \text{distance}(i,j) - c * \text{difficulty}(j)$$

其中a为**报酬系数**，决定报酬对任务完成度的影响，b为**距离系数**，决定会员对已走过的距离的重视程度，c为**难度系数**，决定任务本身难度对完成任务的影响。

（3）蒙特卡罗算法

算法模拟了若干轮任务选择-任务执行的过程，每一轮过程如下：

第一步：会员按照信誉度选择任务，从所有未完成且未选择的任务中按照任务选择模型选出一个估价最高的任务。将被选择的任务状态变为已选择，将无法

被其他人继续选取。

第二步：在所有会员选择完任务（或者放弃这轮）后，按照任务执行模型给出的概率判断这个任务是否完成。

第三步：从未完成的任务中移除已完成的任务，再将剩余任务的已选择标签去除，以供下一轮选择。

在若干轮结束后，我们将检视并输出所有任务的完成情况。

2.2.4 单次蒙特卡罗算法

（1）基本参数假设

我们先行使用以下参数建立了模型：

任务初始难度：25

距离厌恶系数：$k = 70$

报酬系数：$a = 1.0$

距离系数：$b = 50$

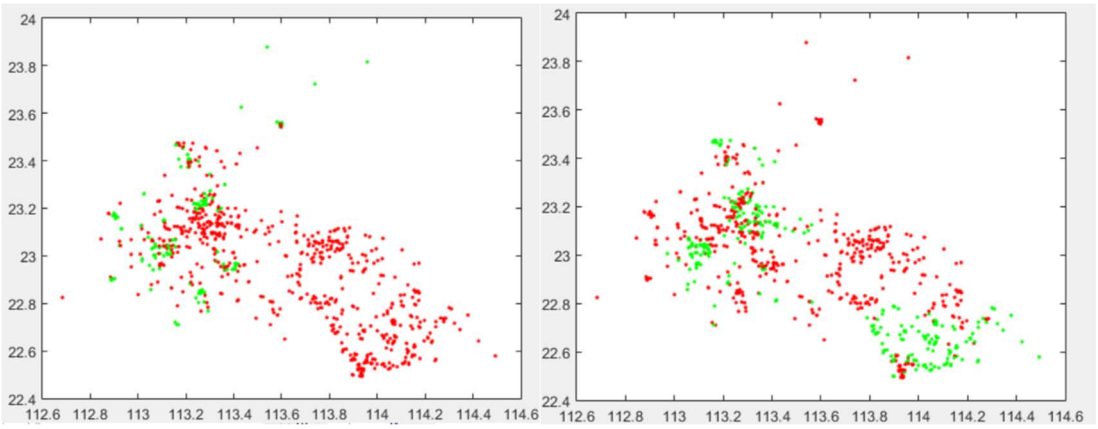
难度系数：$c = 2.0$

同时，每个任务选择的轮数为5，即一个任务会经过5轮选择-放弃的过程，直到最终被标记为未完成。对于会员打开 APP 的概率，我们定义$p_1 = 1,\ p_2 = 0.6,\ p_3 = 0.25,\ p_4 = 0.10$。

将这一组参数输入到蒙特卡罗算法中，进行模拟运算。

（2）结果与分析

由以上参数，我们生成了单次蒙特卡罗算法的结果（图七左图），其中绿色点表示未完成任务，红色点表示已完成任务。作为对比，我们把真实任务分布图放在其右侧。



图七 单次蒙特卡罗算法模拟结果

其中，红色点代表已完成任务，绿色点代表未完成任务。

我们注意到，第一轮模拟就已经出现了部分未完成任务点的聚类现象。比较真实情况与单次蒙特卡罗算法的结果，我们注意到最明显的两处异同：

a. 在**广州**，未完成任务点和真实情况出现了非常相似的聚类现象。

b. 在**深圳**，任务甚至全部被完成；但在真实数据中，深圳南部也有可观的未完成任务的分布。

考虑到当前模型并没有引入任务本身的难度差异，针对以上两处异同，我们对实际情况下任务未完成的原因提出以下分析：

a. **广州**的未完成任务应该和任务本身难度关系较小。主要原因应为这两个聚类区域的任务过多，而活跃度较高的会员数量较少，导致存在一定的人力短缺，使得一些任务无法完成。应当考虑增加这些区域会员的预定限额；加大宣传以发展新会员；或者抬高定价以吸引其他区域的会员前来完成任务。

b. **深圳**的未完成任务，则应是这里的任务难度相对较大所致。如要鼓励会员完成更高难度的任务，就只能靠提高报酬来实现了。

2.2.5 基于学习的蒙特卡罗算法

在蒙特卡罗算法运行之后，我们可以得到模拟结果与真实结果的差异比较，用这一差异比较来修正我们对任务的难度设定。

由于蒙特卡罗算法基于概率完成模拟运算，不能像标准的机器学习那样通过线性代数的手段进行优化，因此，我们只能通过自定的函数来进行难度的修正。在我们的模型中，**修正函数**如下：

a. 如果一个任务在模拟中被完成，但在真实情况下未被完成，可能是模型假定的难度比实际难度低，我们将给假定难度增加一个有一定随机成分的值。

b. 如果一个任务在模拟中未被完成，但在真实情况下被完成，可能是假定难度比实际难度高，我们将给假定难度减少一个有一定随机成分的值。

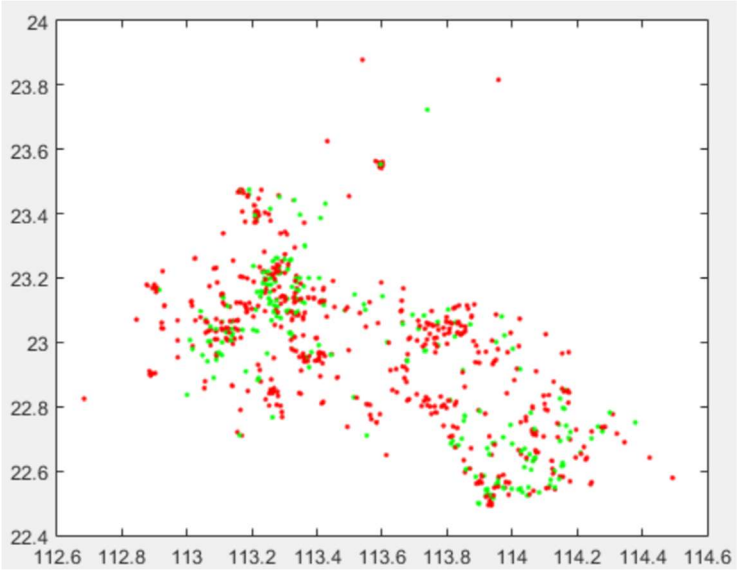
如果只有这两点修正，任务假定难度的职能单方面降低或者单方面升高，蒙特卡罗算法将迅速达到过拟合。考虑到我们采用的蒙特卡罗算法不允许划分测试集和训练集（划分任务会降低任务密度，从而降低模拟的准确性），所以必须引入一个平衡策略，防止过早出现过拟合。

我们加入的**平衡策略**为：

c. 对于在模拟和真实情况下均未完成的任务，假定难度会轻微降低

d. 对于在模拟和真实情况下均已完成的任务，假定难度会轻微升高

引入了难度的学习规则后，我们进行了30轮的蒙特卡罗算法训练，训练后程序的输出结果如下：



图八 多轮蒙特卡罗算法训练后模拟结果

将结果与真实情况进行比较，我们可以对这些任务未完成的原因进行分析探讨。

1. 东莞

在学习的过程中，东莞的任务难度一直降低，但仍存在少量的未完成任务。而在实际情况中，东莞却有着超高的完成率。这一现象的出现，一方面是由于我们为防止过拟合，加入的平衡算法略微倾向于上调完成率高的任务的难度；另一方面也可能因为东莞本身存在与广州和深圳的差异。考虑到东莞的生活成本低于广州和深圳，对于同样的报报酬额，东莞的报酬系数要略高于另外两地，致使东莞有超高的任务完成度。

2. 深圳

深圳南部的任务完成度一直非常低，在学习算法中，我们不断增加深圳南部的任务难度，但同样出于防止过拟合的原因，程序模拟也不会出现大量纯粹的未完成任务。

考虑到深圳出现了大量高难度的任务，我们可以推测，这些任务的难度并不仅仅是由每个任务本身的因素导致的，而是这个区域存在一些影响难度的因素。例如：这些区域交通便利程度较低；深圳整体经济水平较高，报酬吸引力不足等。

3. 广州

广州的未完成任务的聚类现象在学习前便已出现，在优化参数的过程中，这一现象一直得到了保留。可以推测出这里的未完成任务受到的影响相对多元：一方面，区域内任务密度大，会员效率有限，难以全部完成；另一方面，任务本身的难度仍略高于初始值，即这一区域本身也存在一些影响难度的因素。

三、问题二的分析

在前文中，我们利用蒙特卡罗模拟详细分析了任务未完成的原因。下面，我们将针对性地提出新的定价方案，依据经过蒙特卡罗算法训练后的难度数据，构造新的定价函数。

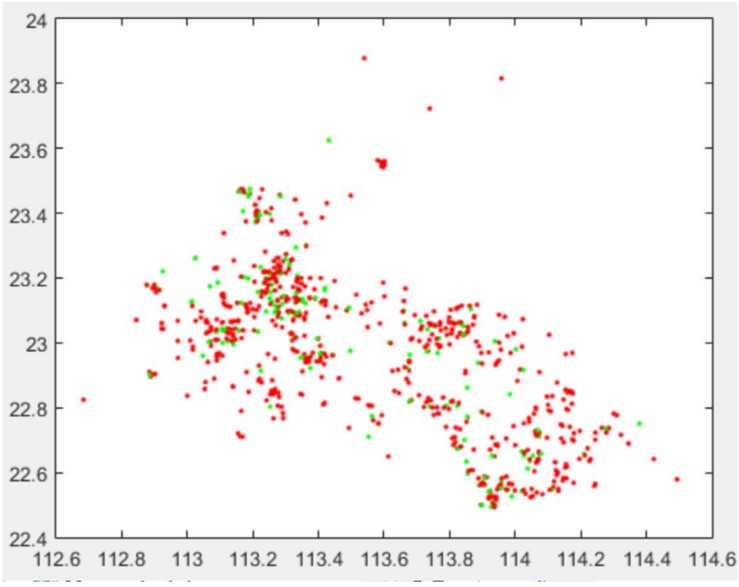
最直观的定价策略，就是对所有未完成的任务依据难度涨价，称为**“难度补偿策略”**。对每一个实际未完成的任务，我们对于难度系数的基础量$c = 2.0$(见2.2.4（1）)定义一个难度增量Δ。**价格补偿公式**如下：

$$\Delta\text{reward} = \Delta * (a/b)$$

其中，a, b 分别为2.2.4（1）中的报酬系数及难度系数。

这样，我们定义了新的价格，将其代入蒙特卡罗模拟中，观测运行效果。

使用难度补偿策略后，整个 APP 额外付出的报酬为4869元。蒙特卡罗模拟效果如下：



图九 使用难度补偿策略后的模拟结果

我们发现，在深圳这一任务高难度地区，完成度的确出现了上升；但广州的情况并不让人满意。这是因为深圳和广州任务未完成的原因不同。深圳的未完成

任务受难度影响较大，而广州则部分地由于任务密度大。所以，仅提高单个任务的报酬不能很好地改善广州的情况。

因此，我们认为任务的新定价应该由**复合策略**组成。对难度高的任务，最主要的策略仍是提高这一任务的报酬，使任务更不易被放弃。对任务较为密集的区域，若仅提高未完成任务的报酬，仍不能避免剩下未完成任务。

我们考虑一个新的因素：**任务密度因素**。这里，报酬补偿不只限于未完成任务，同样也作用在周围存在未完成任务的已完成任务上。为了不让这些任务被落下，我们先行采用一个补偿定值0.5，并观察其效果。

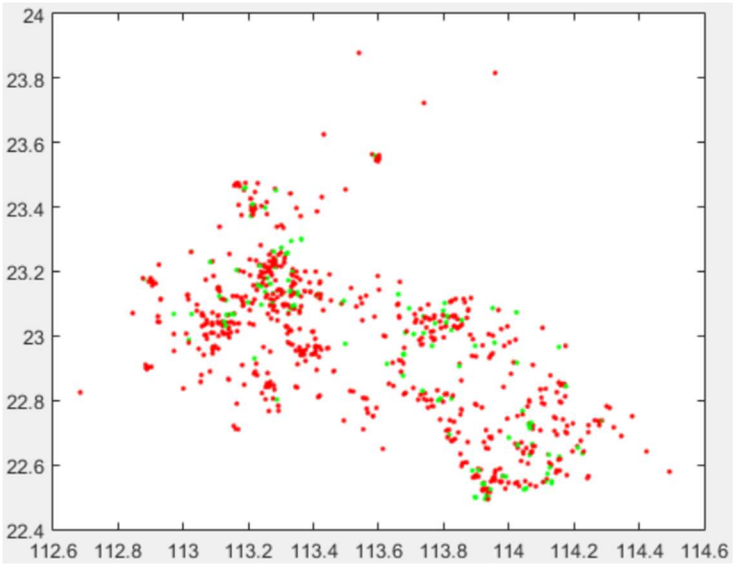
任务补偿变为：

$\Delta\text{reward} = \Delta * (a/b)$　　（该任务未完成）

$0.5 * (0.05$ 距离内未完成任务数$)$　　（该任务已完成）

（公式中的0.05即2.1.3所提到的阈值）

加上这一补偿后，APP 的额外支出达到了6100元。新的模拟结果如下：



图十 加入任务密度因素后的模拟结果

可以看到，这时的未完成任务已经相对分散了。

四、问题三的分析

将任务联合打包发布本质上就是把多个任务合并，避免出现会员活动范围过于集中的情况。

4.1 合并方案分析

4.1.1 可合并任务的类型

考虑到未完成的任务本身存在难点，如果简单地合并两个未完成的任务，产生的新任务有很大可能仍不能被完成。因为这样合并得到的新任务是由两个难度高于平均水平的任务组成，反而更不易被完成。合并任务可能需要更高的报酬，效果未必优于对单个任务提高报酬。

将已完成任务和未完成任务捆绑，让会员单次完成更多任务，这一策略在广州这类任务密度较高的地方可以采用，鼓励会员完成更多的“冷门”任务。但这一策略也不能完全解决问题。广州未完成任务周围有一定量的已完成任务，可以捆绑；但深圳的未完成任务周围的已完成任务较少，难以找到适当的捆绑目标。

若将已完成任务进行捆绑，意义相对不大，甚至可能导致任务过于繁重，无人愿意完成，反而会降低任务的完成率。

总体来说，任务的捆绑有一定的价值，但也有其局限性，并非所有地区都能通过任务捆绑来提高效率。

4.1.2 可合并任务的定价策略

（1）合并任务给予额外补助：一个会员完成一个合并任务获得的报酬高于原有零散任务之和。尤适用于将低难度任务和高难度任务捆绑的策略。

（2）不给予额外补助：失去了合并任务鼓励会员完成的效益，加上难度增加，对任务完成度改进不大。

4.2 合并策略

对**广州**的未完成任务，捆绑周边的邻近任务，给予较低的合并补助（如 5 元），既能够降低任务密度，又有适当补偿。

对**东莞**的任务，由于完成率极高，考虑将任务进行减价合并（如 2 个 65 元任务合并为一个 120 元任务），通过三级价格歧视压低成本。

对**深圳**的任务，因为聚类现象过于明显，难以找到较好的捆绑措施。对于深圳地区，还是建议以增加任务酬金以吸引其他人前来。

4.3 基于模型的策略分析

捆绑后的任务报酬较高，按照原有算法，会员极有选择捆绑后的任务。虽然仍不知单个任务的难度，但可以在选择时识别出捆绑的任务。需要修改的是原模型中选择任务阶段。

执行任务阶段也存在一定问题。例如，会员在完成了捆绑任务的一部分后放弃，收益如何；会员可能将捆绑任务分配在不同日期完成。原模型的执行阶段也需修改。

这些设定将会在模型中引入大量的常量，对这些常量进行修改，可以使模型呈现一系列不同的效果（从会员热衷于完成捆绑任务，到捆绑任务无人问津）。我们目前没有用于修正参数的数据，无法用蒙特卡罗模拟分析捆绑策略的效果。

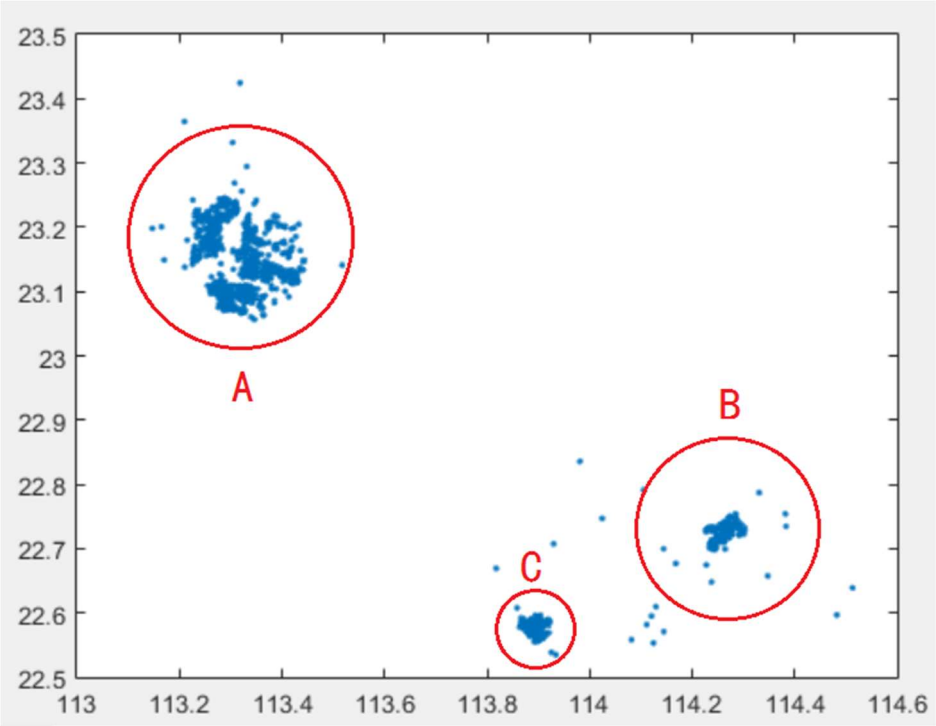
不过，应用厂商可以进行试点任务捆绑，以获得更多的数据来完善修正模型，寻找更优方案。

五、对问题四的分析

基于前文的分析，我们给出了一个新定价策略，并简单评估了捆绑策略的可能效果。现在我们将这一新定价策略应用在新的任务数据中。

5.1 新项目分布分析

我们利用 Matlab 软件，绘制出新项目的分布图。新项目的聚集程度很高，主要集中在广州市区和深圳的两个区域。也就是说，附件一中来自东莞的数据失去了参考价值，我们将重点放在广州和深圳上。



依据图中的地理位置，新增项目大致可分为 3 个集群：A 集群，即广州集群；B 集群位于深圳东部，附件一中，这一带任务密度较稀疏；C 集群位于深圳西南部。值得注意的是，在附件一的数据中，深圳南部除开一小块区域（113.9E，22.5N）以外，任务完成率极低，而 C 集群距离这一区域非常近。

对不同的集群，考虑不同的定价策略。A 集群任务密度高，会员密度也高，

可以先行代入一个较低价格（如 65）进行计算模拟。B 集群落在附件一项目中任务较稀疏的区域，我们推测是这一区域任务难度较大所致，应按照问题二中的新定价策略进行估计。C 集群落在了附件一项目中任务完成率较高的区域，且任务密度大，可以合理推测这一区域应该是深圳的繁华区，与 B 集群采取相同策略。

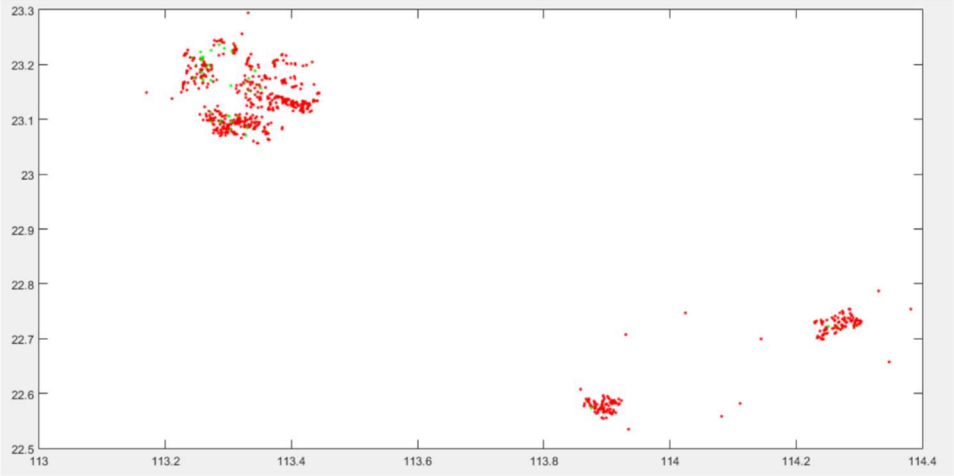
5.2 模型问题分析

在我们的蒙特卡罗算法中，任务完成情况不单由每个任务的地理位置决定，同时还由其他任务的位置和密度、会员的活跃度和密度共同决定。也就是说，在不改变会员分布的情况下，能完成的任务数目是非常有限的。如果仍然沿用通过附件一的 800 组任务数据学习得到的模型，这一轮的 2000 组数据在正常定价范围内被完成的量都非常小。

换句话说，想要更好地提高任务完成度，最主要的目标应该是扩充会员人数并提高会员活跃度，这样才能从根本上解决任务完成情况的问题。

5.3 修改评估方案

由于无法预测未来的会员情况，我们继续使用当前的会员数据。新任务数量大致是附件一的 2.5 倍，因此也给予会员 2.5 倍的执行时间和任务预定限额，进行一次模拟。运行效果如下：



可以看到，这一新模型对任务完成情况有显著提高。

附录

xls 解析器（python）

```
import xlrd

while (1):
    name = input()
    nameIn = name + '.xls'
    nameOut = name + '.txt'

    data = xlrd.open_workbook(nameIn)
    table = data.sheet_by_index(0)

    print(str(table.nrows))
    XLSData = open(nameOut, "w")
    for rows in range(1, table.nrows):
        rowValue = table.row_values(rows)
        for item in rowValue:
            XLSData.write(str(item) + " ")
        XLSData.write("\n")
```

密度计算器 （python）

```
import math

def sqr(Num):
    return Num * Num

def GetDis(Node1, Node2, Type):    #0 is Manhattan, 1 is Euclid
    if (Type == 1):
        SS = abs(Node1.pX - Node2.pX) + abs(Node1.pY - Node2.pY);
    else:
        SS = math.sqrt(sqr(Node1.pX - Node2.pX) + sqr(Node1.pY - Node2.pY))
    return SS

def DisToPoint_MtM(dis, Node1, Node2):
    Flag = 0.05
    if (dis > Flag):
        return 0
    return (Flag - dis)

def DisToPoint_PtM(dis, Node1, Node2):
```

```
    Flag = 0.05
    if (dis > Flag):
        return 0
    return 4*(Flag - dis)

class Node:
    def __init__(self, posX, posY, Cost):
        self.pX = posX
        self.pY = posY
        self.Cost = Cost
        self.Point = 0
        self.Point2 = 0

    def AddPoint(self, Point):
        if (Point > 0):
            self.Point += Point

    def AddPoint2(self, Point):
        self.Point2 += Point

MarketData = open("Mission_End.txt", "r")
Cnt = 0
MarketList = []
for line in MarketData:
    Cnt = Cnt + 1
    Data = line.split()
    if (float(Data[4]) > 0.5):
        MarketList.append(Node(float(Data[1]), float(Data[2]), float(Data[3])))
    else:
        MarketList.append(Node(float(Data[1]), float(Data[2]), float(Data[3])))

PersonData = open("MemberInfo.txt", "r")
PersonList = []
for line in PersonData:
    Data = line.split()
    PersonList.append(Node(float(Data[1]), float(Data[2]), float(Data[3])))

for Node1 in MarketList:
    for Node2 in MarketList:
        if (Node1 != Node2):
            Dis = GetDis(Node1, Node2, 0)
            Poi = DisToPoint_MtM(Dis, Node1, Node2)
            Node1.AddPoint(Poi)
```

```
for Node1 in MarketList:
    for Node2 in PersonList:
        Dis = GetDis(Node1, Node2, 0)
        Poi = DisToPoint_PtM(Dis, Node1, Node2)
        Node1.AddPoint(Poi)

Data2 = open('Points222.txt', "w")
for Node in MarketList:
    if (Node.Cost > 75):
        continue
    Data2.write(str(Node.Cost) + " " + str(Node.Point) + "\n")
    #输出视需求调整
```

蒙特卡罗算法模拟器(python)

```
    import math
    import random

    global Train_Times
    global ExtraCnt
    TrainTimes = 0              #How many times you want to train

    class Market:
        def __init__(self, posX, posY, Price, RealDone):                    #Add more if we
needed
            self.pX = posX
            self.pY = posY
            self.Price = Price
            self.Done = 0
            self.RealDone = RealDone
            self.Picked = False                                    #If picked this turn,
nobody can pick it again
            self.Diff = 25                                        #Start with 25 diff

    class Person:
        def __init__(self, posX, posY, MissionLimit, Credit):          #Add more if needed
            self.pX = posX
            self.pY = posY
            self.ML = MissionLimit
            self.Cd = Credit
            self.LastFail = None                                  #Last mission this guy
tried but failed
            self.MF = 0                                           #Mission finished
```

```
    def InitMarket():
        Data = open("Mission_End.txt", "r")
        global MarketList
        MarketList = []
        for line in Data:
            Data = line.split()
            MarketList.append(Market(float(Data[1]), float(Data[2]), float(Data[3]),
float(Data[4])))

    def InitNewMarket():
        Data = open("Mission_New.txt", "r")
        global MarketList
        MarketList = []
        for line in Data:
            Data = line.split()
            if (float(Data[1]) > 114.2):
                MarketList.append(Market(float(Data[1]), float(Data[2]), 70, 0))
            else:
                MarketList.append(Market(float(Data[1]), float(Data[2]), 65, 0))

    def InitPerson():
        Data = open("MemberInfo.txt", "r")
        global PersonList
        PersonList = []
        for line in Data:
            Data = line.split()
            PersonList.append(Person(float(Data[1]), float(Data[2]), float(Data[3]),
float(Data[5])))

    def GetDis(Person, Market):
        Dis2 = (Person.pX - Market.pX) * (Person.pX - Market.pX) + (Person.pY - Market.pY)
* (Person.pY - Market.pY)
        return math.sqrt(Dis2)

    def GetDis2(Market1, Market2):
        Dis2 = (Market1.pX - Market2.pX) * (Market1.pX - Market2.pX) + (Market1.pY -
Market2.pY) * (Market1.pY - Market2.pY)
        return math.sqrt(Dis2)

    def GetPoi(Person, Market):                    #how likely the person will change his
favorite
        return Market.Price - 70 * GetDis(Person, Market)

    def MarketCheck(Person, Market):               #weather the person finish the task in
```

```
market
            # With little data, we have to use magic numbers
            # Just hope this will work fine ...
            Poss = -Market.Diff * 2.0
            Poss += Market.Price * 1.0
            Poss += 50 * GetDis(Person, Market)
            if (random.random() * 100 < Poss):
                return True
            return False

        def Remark():
            global    ExtraCnt
            for Market in MarketList:
                if (Market.RealDone == 0):
                    Extra = (Market.Diff - 25)
                    if (Extra < 0):
                        Extra = 0
                    if (Extra > 10):
                        Extra = 10
                    Market.Price += 4 * Extra
                    ExtraCnt = ExtraCnt + 4 * Extra
                else:
                    for Market2 in MarketList:
                        if (Market2.RealDone == 0):
                            if (GetDis2(Market, Market2) < 0.05):
                                Market.Price += 0.5
                                ExtraCnt += 0.5

        def MonteCarlo():
            for i in range(1, 13):                  #for Question 4
        #       for i in range(1, 6):                   #for Question 1, 2, 3
                    for Person in PersonList:
                        if (Person.MF == Person.ML):
                            continue
                        #Those people can not do more missions
                        if (Person.Cd < 40):
                            if (random.random() < 0.4):
                                continue
                        if (Person.Cd < 20):
                            if (random.random() < 0.6):
                                continue
                        if (Person.Cd < 19):
                            if (random.random() < 0.6):
                                continue
```

```
                    # Those people with low cridit are easily give up missions, so we have a
chance to skip them ...

                    Market_Chosen = None
                    Market_Point = 0
                    for Market in MarketList:
                        if (Market.Picked):
                            continue
                        if (Market.Done == 1):
                            continue
                        #if (Market == Person.LastFail):
                        #       continue
                        SS = GetPoi(Person, Market)
                        if (Market_Point < SS):
                            Market_Point = SS
                            Market_Chosen = Market

                    if (Market_Chosen != None):
                        Market_Chosen.Picked = True
                        if (MarketCheck(Person, Market_Chosen)):
                            Person.MF += 1
                            Market_Chosen.Done = 1
                        else:
                            Person.LastFail = Market_Chosen

                for Market in MarketList:
                    Market.Picked = False

    def MonteCarlo_Traing():
        for Market in MarketList:
            if ((Market.Done == 1) & (Market.RealDone == 0)):                    #Finish
which should not be finished
                Market.Diff += random.random() * 5 / Train_Times
            if ((Market.Done == 0) & (Market.RealDone == 1)):                    #Give up
which shout not be given up
                Market.Diff -= random.random() * 5 / Train_Times
            if ((Market.Done == 0) & (Market.RealDone == 0)):
                Market.Diff -= random.random() * 2 / Train_Times
            if ((Market.Done == 1) & (Market.RealDone == 1)):
                Market.Diff += random.random() * 2 / Train_Times
            if (Market.Diff < 0):
                Market.Diff = 0
            Market.Done = 0
        for Person in PersonList:
```

```
                Person.LastFail = None
                Person.MF = 0

        def MorePeople():
            for Person in PersonList:
                Person.ML *= 3;

        def PrintResult():
            Data2 = open('MC_Result.txt', "w")
            for Market in MarketList:
                Data2.write(str(Market.pX) + " " + str(Market.pY) + " " + str(Market.Done) + " "
+ str(Market.Diff) + "\n")

        InitMarket()
        InitPerson()

        Train_Times = 0
        for i in range(0, TrainTimes):
            Train_Times = i
            if (Train_Times < 4):
                Train_Times = 3
            if (Train_Times > 10):
                Train_Times = 5 + Train_Times*0.5
            MonteCarlo()
            MonteCarlo_Traing()

        #ExtraCnt = 0
        #Remark()                #Remark if we need
        #print (str(ExtraCnt))
        #the up three is for question 3

        InitNewMarket()
        MorePeople()
        #the up two is for question 4
        MonteCarlo()
        PrintResult()
```

热力图绘制工具 （matlab）

```
S = load('Points222.txt');
x = (1:835);
y = (1:835);
z1 = (1:835);
```

```
z2 = (1:835);
for i = 1:835
    x(i) = S(i,1);
    y(i) = S(i,2);
    z1(i) = S(i,3);
    z2(i) = S(i,4);
end;
 maxx = max(x);
 minx = min(x);
 maxy = max(y);
 miny = min(y);
 figure(1)
 [X,Y,Z1] = griddata(y, x, z1, linspace(miny, maxy, 300), linspace(minx, maxx, 300)', 'cubic');
 Draw1 = pcolor(X, Y, Z1)
 set(Draw1, 'LineStyle','none');

 figure(2)
 [X,Y,Z2] = griddata(y, x, z2, linspace(miny, maxy, 300), linspace(minx, maxx, 300)', 'cubic');
 Draw2 = pcolor(X, Y, Z2)
 set(Draw2, 'LineStyle','none');
```

分布聚类散点绘制 (matlab)

```
S = load('Points222.txt');
Q = (1:835);
K = (1:835);
T = (1:835);
for i = 1:835
    Q(i) = S(i,1);
    K(i) = S(i,2);
    T(i) = S(i,3);
end;
S1 = 0;
S2 = 0;
S3 = 0;
S4 = 0;
for i = 1:835
    if (T(i) > 0)
        S2 = S2 + 1;
        if (Q(i) < K(i))
            S1 = S1 + 1;
        end;
        plot(Q(i), K(i), '.', 'Color', [1, 0, 0])
    else
```

```
            S4 = S4 + 1;
            if (Q(i) > K(i))
                S3 = S3 + 1;
            end;
            plot(Q(i), K(i), '.', 'Color', [0, 1, 0])
        end;
        hold on
end;
S1
S2
S3
S4
S1 / S2
S3 / S4
```

蒙特卡罗结果绘制(matlab)

```
S = load('MC_Result.txt');
X = S(:, 2);
Y = S(:, 1);
T = S(:, 3);

S1 = 0;
S2 = 0;
for i = 1:835
    if (T(i) > 0)
        S1 = S1 + 1;
        plot(X(i), Y(i), '.', 'Color', [1, 0, 0])
    else
        S2 = S2 + 1;
        plot(X(i), Y(i), '.', 'Color', [0, 1, 0])
    end;
    hold on
end;
S1
S2
```