基于计算博弈论与排队论的机场出租车智能动态规划模型

摘要

为解决机场出租车规划问题，本文从出租车收益、时间成本等角度分析，建立不同的数学模型，给出了司机的选择策略、乘车点的设计方案、可行的“优先”方案。

针对问题一，我们依据计算博弈论，综合考虑影响出租车司机决策的相关因素，如司机待客时间、不同时段机场的乘客数量、道路实况状态等，建立了彼此之间的函数关系。为了实现出租车司机收益的最大化，减少时间成本，我们构造了多目标函数规划模型，在最优解下给出司机的选择策略。在求解过程中，利用多数规则理论分析了传统求解目标规划的解法，为降低算法的复杂度，我们采用了 FLORA 算法，以半空间表示法的方式确定出租车司机的策略矩阵。

针对问题二，我们首先爬取了 2007 年某日上海市出租车 GPS 轨迹数据和全国航班信息来对模型进行检验。依照问题一中建立的基于博弈论的多目标决策模型，确定相关参数信息，模拟出不同时段下出租车的运载率进行模型合理性检验。另外，考虑到我们已掌握较多的出租车轨迹数据以及机场相关信息，因此我们使用了数据挖掘技术建立神经网络训练模型。在进行数据预处理、特征工程之后可以在短时期内建立决策模型，并分析影响因素对决策目标的依赖性。

针对问题三，我们在设置上车点位置的时候着重考虑了两方面的问题：首先，上车点的位置需保障车辆和乘客安全；其次，在最短时间完成最多数量的搭载行为。我们首先考虑由于机场乘坐出租车的乘客流满足泊松分布，因此可以基于排队论平衡乘客需求和服务设施之间的关系。通过建立排队模型，并根据乘车效率最高（即时间成本和服务成本最小）进行优化。以上海浦东机场统计数据为例，通过计算得到在乘车区设置 8 个上车点时乘车效率最高。

针对问题四，为了在保持出租车收益尽量均衡的前提下保证出租车的收益最多，我们建立了双目标规划模型，考虑影响载客收益的因素，确定主要因素为租车司机离开机场的的位置距离与出租车到达机场的被安排的“优先”位置。接着利用遗传算法寻找最优解，在最优解下获取位置信息，最终给出了可行的“优先”安排方案：当司机从机场出发的载客行驶距离小于 31 千米，返程继续拉客时“优先”位置被安排为第 41 个站点，可使收益的尽量均衡且司机收益最大化。

关键词 计算博弈论；排队论；神经网络；遗传算法；机场出租车

一 问题重述

1.1 问题背景

大多数乘客下飞机后要去市区（或周边）的目的地，出租车是主要的交通工具之一。国内多数机场都是将送客（出发）与接客（到达）通道分开的。送客到机场的出租车司机都将会面临两个选择：前往到达区排队等待载客返回市区或直接放空返回市区拉客。出租车司机会付出空载费用和可能损失潜在的载客收益。

1.2 问题提出

请你们团队结合实际情况，建立数学模型研究下列问题：

(1) 分析研究与出租车司机决策相关因素的影响机理，综合考虑机场乘客数量的变化规律和出租车司机的收益，建立出租车司机选择决策模型，并给出司机的选择策略。

(2) 收集国内某一机场及其所在城市出租车的相关数据，给出该机场出租车司机的选择方案，并分析模型的合理性和对相关因素的依赖性。

(3) 在某些时候，经常会出现出租车排队载客和乘客排队乘车的情况。某机场“乘车区”现有两条并行车道，管理部门应如何设置“上车点”，并合理安排出租车和乘客，在保证车辆和乘客安全的条件下，使得总的乘车效率最高。

(4) 机场的出租车载客收益与载客的行驶里程有关，乘客的目的地有远有近，出租车司机不能选择乘客和拒载，但允许出租车多次往返载客。管理部门拟对某些短途载客再次返回的出租车给予一定的“优先权”，使得这些出租车的收益尽量均衡，试给出一个可行的“优先”安排方案。

二 问题分析

2.1 问题一的分析

问题一要求分析研究出租车司机决策相关因素的影响机理，即需要首先确定影响出租车决策的因素，建立影响因素与决策的关系。由于司机决策时需要考虑到机场乘客数量的变化规律和相应的收益，并期望获得收益最大化、减少时间成本，因此我们引入博弈论，建立了有关收益和时间成本的多目标决策规划模型，并确定了影响目标函数的相关因素，通过建立它们彼此之间的函数关系，进而给出司机的选择策略。

2.2 问题二的分析

问题二要求我们收集相关数据后给出司机的选择方案并分析模型的合理性和对相关因素的依赖性。我们通过网上爬取机场某天的航班信息以及该城市当天的出租车GPS 轨迹信息，对问题一所建立的模型进行检验和分析。通过带入相关参数可以确定博弈模型中的函数关系，进而模拟在多种因素影响下司机的选择策略，与真实数据的对比，可以判断问题一所建模型的合理性。另外，考虑到我们已掌握较多的出租车轨

迹以及机场相关信息，因此可以使用数据挖掘技术建立神经网络训练模型。在进行数据预处理、特征工程之后可以在短时期内建立决策模型，并分析影响因素对决策目标的依赖性。

2.3 问题三的分析

问题三要求我们在保证车辆和乘客安全的条件下设置“上车点”，使得乘车效率最高。根据题目要求，在设置上车点位置的时候需注意两方面的问题：首先，上车点的位置需保障车辆和乘客安全；其次，在最短时间完成最多数量的搭载行为。我们考虑到由于机场乘坐出租车的乘客流满足泊松分布，因此可以基于排队论平衡乘客需求和服务设施之间的关系。因此，我们建立了 M/M/C 排队模型，并且根据乘车效率最高（即时间成本和服务成本最小）进行优化。最后以上海浦东机场统计数据为例，计算得到上海市浦东机场的最佳上车点数。

2.4 问题四的分析

问题四要求我们在保持出租车收益尽量均衡的前提下保证出租车的收益最多。虽然线性规划时解决规划问题的常用方法，但是在本题中存在有效解 “组合爆炸” 的问题，因此我们选择使用遗传算法对出租车的收益问题进行优化求解。为了给出可行的优化安排方案，需要考虑“优先权”的影响因素，然后利用遗传算法中的交叉变异运算因子对问题进行优化。

三 基本假设

1、假设所有出租车为同一车型，有同种收费方式、经营方式；
2、假设不考虑乘客合乘引起的时间成本损失；
3、假设出租车正常运营，不会出现长期停滞在机场的状况；
4、假设无论服务远近，司机不能拒绝乘客；
5、某时间段内抵达航班数对司机是已知的；
6、假设获取航班信息的飞机均为满载。

四 符号说明

符号	符号定义
s	选择策略
Z	出租车司机所获效益
r	价格费率
Y_i	第 i 时段乘客运载量

T_i	出租车待客时间
G_i	第 i 时段里程定价
u_i	运行速度
Q_i	运载率
Car_ID	出租车编号
Longitude	GPS 经度
Latitude	GPS 纬度
Speeed	瞬时速度
Direction	方向
State	出租车载客状态
d	出租车行驶距离
α	出租车乘客平均到达率
β	出租车的平均服务率
B	停车点的个数
μ	时间成本
σ	服务成本

五 问题一模型的建立与求解

5.1.基于博弈论出租车司机决策模型的建立

出租车司机到达机场后可以选择排队载客到市区或周边，也可以选择返回市区拉客，司机需要考虑对时间成本与效益，最小化时间成本提高拉客效率，与此同时增加收益大小。在这一过程即存在博弈行为，出租车司机在考虑客运量与路况等信息与时间的相互作用，根据对抵达的航班数量和“蓄车池”里已有的车辆数等因素的认知，做出更有利于提高收益、降低时间成本的决策行为。

一天中不同时间，航班的客运量不同即乘客的需求量不同，道路状况即拥挤度不同，即一天中出租车司机在面临选择时考虑的因素处于一种动态的变化过程。为了研究一天中机场乘客的变化规律以及道路状况等因素，将一天中的时间离散地划分为时

段，在每一时段内，将影响司机决策的因素锁定，建立静态模型，分析出租车司机所面临的时间成本以及相应的收益。

出租车司机到达机场后有两种选择，载客与不载客，决策的行为与时间相关，司机需要判断依据不同时间对综合因素的分析，来确定在不同时段范围内是否选择载客。记出租车司机的策略 $s\in\{0.1\}$，在不同时段选择不同策略，记 s_i 表示第 i 时段司机的选择策略。当把 L 表示上海市区内出租车司机集合，L 中的每一个元素记为 x，都面临着策略选择，衡量选择载客回市区的司机的数量，q(x)满足：

$$q(x)=\sum_{x\epsilon L} x\cdot s$$

为确定出租车司机的收益，收益大小需要考虑运营里程，以及在不同里程下对应的价格大小即为 r，这里用 Z 来衡量出租车司机的效益大小：

$$Z=\arg\max\sum_{i} z_i(q(x),r)$$

为降低时间成本，提高运营效率，需要衡量在不同时段乘客的数量，由于乘客数量随着时间变动，乘客数量的多少以及“蓄车池”出租车数量的多少影响机场出租车的运送乘客数量，同时体现为对出租车司机的候车时间的影响，为了反映出时间成本的最小化，取不同时段内机场出租车的运送乘客数量最大化。把对时间成本的衡量转化为处于载客状态出租车数量的衡量，取 Y 为乘客运载量，则有：

$$Y=\max\sum_{i} y_i\left(q(x),r\right)$$

这里，出租车司机收益的最大化与乘客运载量的最大化是两项决策目标，目标函数是以不同时段处于出租车载客状态的数量以及不同里程下的价格比率为自变量确定的函数。

考虑影响决策目标函数的相关因素，出租车司机的收益大小与不同时段单个出租车元素运送乘客来往机场的次数有关，与不同里程下费油量有关以及里程价格有关。

在确定目标函数的大小时，乘客的运载量表示为：

$$Y_i(R_i,S_i,T_i)=\overline{Y_i}\exp\{-\eta\cdot(\frac{R_i}{\alpha}+\xi S_i+\zeta T_i)\}$$

乘客的运载量与出租车运营平均的票价 R_i、平均的运营时间 S_i、平均的等待时间 T_i 有关，其中系数 η 用于调节需求量，系数 ξ 与系数 ζ 是用于表示时间和价格的变化参数，α 表示出租车的平均载客量，$\overline{Y_i}$ 表示在正常情况下机场中的乘客数目。出租车平均的等待时间同样受到乘客运载量的影响，满足如下关系式：

$$T_i(Y_i,S_i,Q_i)=\frac{Q_i\cdot N-\dfrac{Y_iS_i}{\gamma\tau}}{\sigma} \tag{1}$$

出租车的待客时间是乘客运载量、离开机场后出租车平均运营里程下所需要的时间、以及在某一时段“蓄车池" 内出租车数量的比例三者的函数，出租车在机场的待客时间随着“蓄车池”内出租车的数量的增多而增大，同时随着一定运营时间下的出租车乘客运载量成负相关，并于乘客在机场待车区允许活动范围成反比。式中，N 表示蓄车池内可容纳出租车的总量，γ 表示出租车在一天时间段内的出租车司机的平均载客量，τ 为划定的时段长度。

在衡量出租车的收益时，需要基于出租车载客时的不同里程下所收取价格来建立效益关系：

$$G_i(r_i) = r_0 + r_i \cdot (\bar{d} - d_0)$$

式中 d_0 为起步定价的里程，$\bar{d}$ 表示机场内出租车载客后平均运行的距离，r_i 表示某一时段内的计价费率，r_0 表示时段内的起步价格。

在以上讨论的有关乘客运载量的函数关系、出租车司机的待客时间函数、以及不同时段下的里程函数关系，是在划分时段下的在不同影响因素下的表达关系，因为最终的目的是降低时间成本，减少出租车司机的等客时间，与此同时来增加出租车司机的收益，而以上考虑的函数关系是决策目标的影响因素，最后反映为所建立的博弈模型下的目标函数变量因子。进一步，在控制以上相关因素下，来综合考虑上述因子对目标函数的影响，定义效用函数为：

$$Z_i(r_i, Q_i) = \frac{Y_i G_i(r_i)}{\gamma N} - Q_i v_i \tau$$

其中右式第一项表示司机在不同往返机场的次数下的收入，第二项表示为司机所付出的成本如在不同时段下的考虑路况的拥挤度来刻画耗油量带来的损失成本。式中 v 表示耗油率，具体刻画耗油率与路况的函数关系。引入司机在市区与机场之间的运行速度的相关关系：

$$u_i(Q_i) = u_0 \cdot \frac{\bar{N} - (\frac{N}{\lambda} \cdot Q_i + N_i')}{\bar{N}} \quad (2)$$

$$v_i = \frac{\omega}{u_i} \quad (3)$$

其中 λ 表示往返机场接客的出租车数目占总出租车数目的比例，$\bar{N}$ 表示道路所允许的车流量，N' 表示路面上其他车辆的数目总数，u_0 表示市区与与机场之间的往返车辆的的平均速度，ω 表示耗油率与行车速度之间的比例系数。

在所划分的时段下刻画双目标决策问题，需要引入约束条件即决策规则从而确定决策方案。为了更合理的安排出租车在机场接客，并使模型更具有实际意义，确定基本的约束条件：

（1）出租车司机在机场的逗留时间不得多于n_1个时段，否则视为影响正常机场秩序。

（2）出租车司机在机场与市区的运营时间不得超过n_2个时段，否则不符合正常的出租车运营模式。

为了刻画在不同时段内的出租车的选择策略 s 随机场乘客容量、道路交通状况之间的关系，对出租车司机所组成的集合中元素分析基于不同时段下的决策选择，在每一个时段可以对出租车的选择策略赋予权重 c_i，则出租车司机载客的概率可进行如下描述：

$$Q_i = \sum c_i \cdot \delta(i)$$

当 $j \le i \le k$，$\delta(i) = 0$，不在划定时段下则有 $\delta(i) = 0$

5.2 双层多目标函数模型的求解

5.2.1 基于传统双层多目标规划求解

出租车司机在到达机场时，需要进行策略选择，选择的依据是下层双目标函数即实现效益最大以及相应时段内乘客的运载量最大，进而确定上层目标函数即是否在机场选择载客。在包含出租车元素组成的集合，不同出租车有相同的决策目标，针对本问题，采用同目标

群决策问题的解法。不妨令 $y^* \in Y$，Y 表示所有属性的集合，在本问题中出租车司机的待客时间、机场的客流量、往返机场市区道路的交通情况、不同里程对应的费率等因素都可归结为目标函数的属性，可采用多数规则求解，“多数规则”是指：如果目标函数的属性 $y^* \in Y$ 是问题的解，则不存在其他的 $y \in Y$，使决策群中有一半以上的决策人认为有 $y > y^*$，所有这样的解的集合称为多数核，在求解非劣解情况下，把属性考虑在内：

$$\max f_k(x)，\ 其中 y_k = f_k(x)$$

对于求解多目标规划[1]的非劣解，可采用约束法，对目标函数进行约束处理可求得相应的非劣解，即考虑约束问题：

基于“多数规则“来进行求解，并采用约束法求非劣解，问题转换为：

$$\max f_n(x)，\ 其中 f_k \ge y_k, k = 1,2,\cdots,n-1$$

$$\max f_k(x)，\ 其中 f_k(x) \ge \varepsilon, k = 1,2,\cdots,n$$

其中ε与 x 无关，保证约束问题有解。

记$y_n = g(y_1, y_2, \ldots, y_{n-1})$，即属性$y_n$为其他属性的函数。

从而可以得到：

$$\max Z(y_1, y_2, \cdots, y_{n-1}, g(y_1, y_2, \cdots, y_{n-1})), \text{其中} f_k(x) \geq y_k, k = 1, 2, \cdots, n-1$$

依此方法求解，可以减少变量个数，使问题得到简化。在本问题中，除了涉及多个决策人即不同出租车司机的问题，同时在不同时段出租车选择载客或不载客策略所占的权重不同，即时段影响出租车司机的选择策略。对目标函数加权可以得到综合考虑不同因素的影响效果优化，利用加权法求解非线性规划，从而刻画不同时段的影响对出租车司机选择策略的影响权重。对于给定的权向量 $c=(c_1, c_2, \cdots, c_n)$，依据库恩-塔克条件可得可行解为非劣解的必要条件，进而由权值大小判断问题的非劣解。

5.2.2 基于 FLORA 算法[2]求解决策问题

传统解法将不同时段出租车司机总体的策略选择用线性组合展现，为了简化出租车司机在不同时段的决策空间，依据策略空间本身具有的规律，减小出租车司机在策略选择下的决策空间。

考虑不同时段的出租车在机场的载客情况，这里采用完全紧凑表达法，减小算法的规模，出租车司机选择是否载客的概率，刻画了出租车的选择策略，而出租车司机的选择策略与出租车收益以及所付出的时间成本有关。FLORA 算基于出租车的载客概率为决策属性，并直接将其作为优化对象，此时下层的双目标函数与 Q 相关，为解决出租车司机在机场载客概率 Q 的优化问题，从而给出出租车司机是否选择策略，需要求解 Q 的可行域，若采用集合来描述 Q，则有：

$$Q^* = \{Q \in R^n \mid Q = \sum x_s^*, 0 \leq x^* \leq 1\}$$

x_s^*表示出租车司机关于决策 s 的选择策略下，载客策略在总混合策略得到的概率。为了以矩阵形式更清楚的表示出租车司机的选择策略，采用 KLORA 算法来变换矩阵得到选择策略的矩阵形式。将Q^*看作策略集合的凸包，采用顶点表示法或半空间表示法构造凸多面体的表达。借助半空间表示法化为 n 维线性不等式，在约束条件一二的情况下，表示凸多面体的半空间

$$Q\begin{pmatrix} I_n \\ -I_n \\ 1 \quad \cdots \quad 1 \\ 1 \quad \cdots \quad 1 \\ \ddots \quad \ddots \\ 1 \quad \cdots \quad 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ n_2 \\ n_1 \\ \vdots \\ n_1 \end{pmatrix}, \text{其中} \quad A = \begin{pmatrix} I_n \\ -I_n \\ 1 \quad \cdots \quad 1 \\ 1 \quad \cdots \quad 1 \\ \ddots \quad \ddots \\ 1 \quad \cdots \quad 1 \end{pmatrix}$$

将上式可表述为$AQ \le b$

这样一来可以将目标函数的优化转换为含有 n 个选择策略变量以及约束条件，将问题的规模减小便于求解。由于构成的凸多面体的所有顶点可以表示为$\{0，1\}^n$的子集，顶点 v 组成的向量矩阵可表示为 n 维不等式方程组的唯一解。得到增广矩阵 A^*,对矩阵进行行变换可以得到

$$\left(\begin{array}{ccc|c} 1 & & & * \\ & 1 & & * \\ & & \ddots & \vdots \end{array}\right)$$

由此可以确定 Q 的可行解即出租车司机的选择策略。

六 问题二模型的建立和求解

6.1 博弈模型的合理性分析

在博弈论的支撑下，可以对出租车司机的行为决策进行分析，在问题一中在为达到下层目标函数收益最大以及时间成本最小，我们考虑影响出租车司机决策的因素，包括不同时段内抵达机场的航班信息、市区与机场之间道路的拥堵状况、司机在机场需付出的待客时间、不同里程下的价格收益、“蓄车池“内容许的出租车数目等，设计上层决策分析，即决策目标：到达机场的出租车是否选择载客。进一步利用影响出租车决策的参数信息，基于问题一中博弈理论的分析，设计出了租车司机的选择策略。

在问题一的博弈理论模型中，分析了影响司机决策的行为的函数关系，为给出司机的选择策略，需要确定参数信息。经查阅 2007 年上海浦东机场及其他相关信息可以得到不同的参数值，如下表 1：

表 1 参数信息

参数符号	参数意义	参数数值
τ	时段大小	2 小时
N	“蓄车池“可容纳出租车数量	1800 辆
d	往返机场的平均里程	25 千米
u_0	出租车的平均运行速度	65 千米/小时
T	出租车平均运营时间	3.5 小时
η	需求量因子	0.05
ξ	出租车空载运营付出时间代价	20 元/小时
α	出租车的平均载客量	2 人

d_0	起步定价里程	5 千米
r	计价费率	3 元/千米
r_0	起步价格	20 元
v	耗油率	20 元/小时

根据以上的参数信息得到函数关系的数值表达，为了刻画不同时段内从出租车从机场离开的运载概率，采用 FLORA 算法，即可求得可得到出租车司机在一定时段内的在机场的载客概率 Q。

为了验证模型建立的合理性，我们利用 2007 年上海浦东机场的出租车司机的载客信息，确定不同时段下司机的载客概率，数据如下表 2 所示。

表 2 2007 年 2 月 2 日上海浦东机场载客运营信息

时段（小时）	0-2	2-4	4-6	6-8	8-10	10-12
载客出租车数目（个数）	2	0	0	3	2	11
出租车数目总数（个数）	2	3	2	33	21	29
运载率	100%	0%	0%	9%	10%	38%
时段（小时）	12-14	14-16	16-18	18-20	20-22	22-24
载客出租车数目（个数）	10	10	14	10	10	14
出租车数目总数（个数）	33	20	22	21	19	38
运载率	30%	50%	64%	48%	53%	37%

在问题一确定的模型下确定不同时段内的载客率与 2007 年上海市浦东机场真实数据对比可以得到下图 1 关系

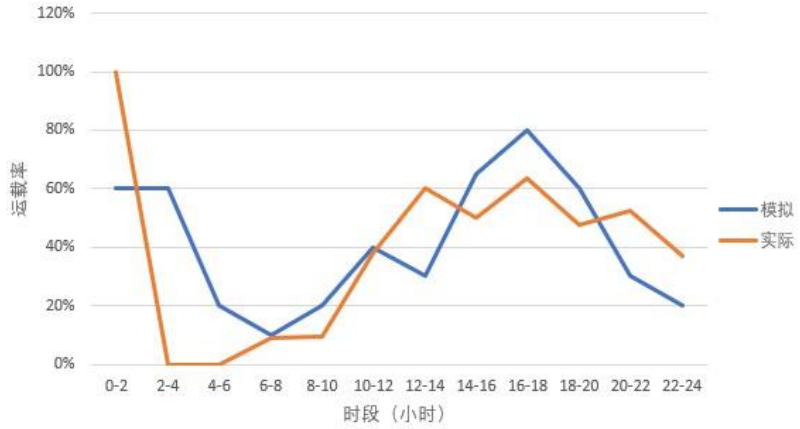


图 1　2007 年上海浦东机场出租车载客率模拟数据与真实数据

从图中可以看出，2007 年一天中模拟得到的部分数据虽然与真实数据有较大偏差，但却随时间的变化规律基本一致，说明了基于博弈论所建立的模型具有合理性，尽管出现偏差，可能 2007 年 2 月 2 日对应当年的节假日，同时存在天气等偶然因素的影响，而这些因素都会影响模型的准确度。

假设在模型一中，分别忽略候车时间和不同时段内机场航班的乘客数量对出租车司机效益的影响，利用 FLORA 算法获取出租车司机的运载率，与真实数据相对比，如下图 2、3 所示。

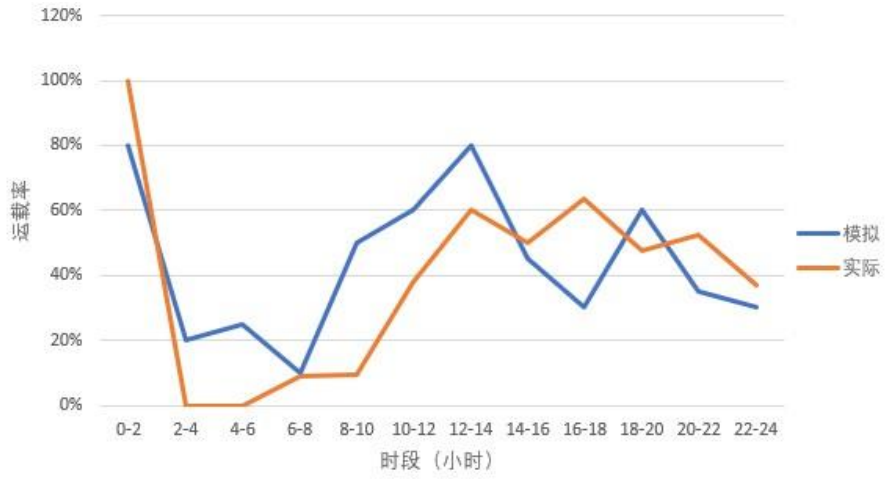
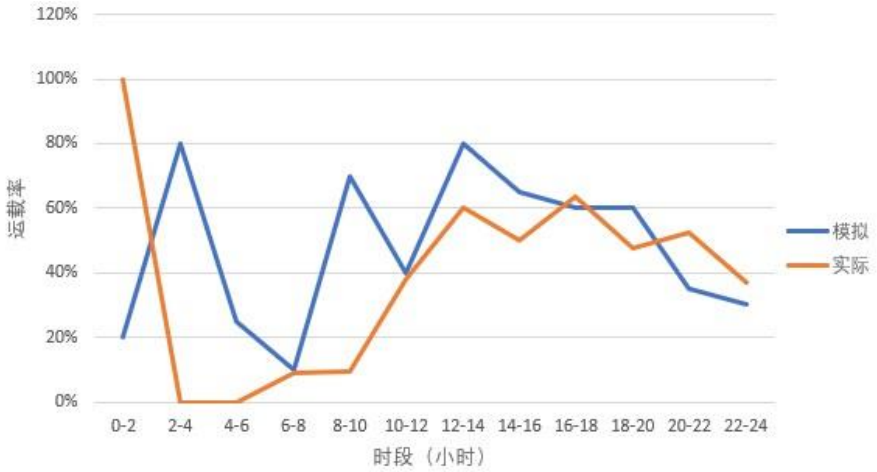


图 2 忽略候车时间模拟数据与真实数据比对



：

图 3 忽略航班乘客数量模拟数据与真实数据比对

从附中可以看出若忽略候车时间，运载率的模拟数据与真实数据相比，可以看出部分时段成趋势成反向增长，说明忽略候车时间显然出租车运载率造成影响，同理，当忽略不同时段下的航班信息，模拟数据出现上下波动与真实数据相差较大，说明航班乘客数量对司机决策也回造成影响。这说明了司机的选择策略与候车时间、不同时段内的航班乘客数量等因素相关。

6.2 基于 DNN 神经网络决策模型

基于问题一所建立的博弈模型，考虑到多种因素对载客率的影响，求解获得的结果与真实数据对比，大体上说明了模型的合理性，但是未能确切的量化不同相关因素对决策变量的相关依赖性的程度，因此在问题一的基础上，收集数据，利用计算机算法来训练模型进而量化不同影响因素对因变量的相关性。

6.2.1 获取数据

采用的出租车 GPS 实时定位数据为上海市 2007 年 2 月 20 日共 4.5 万辆出租车一天内的轨迹信息。根据一天的时间间隔分层抽样获得 4316 辆出租车轨迹信息。每辆出租车的轨迹数据共包括七个字段，其数据格式和表示意义如下表 3 所示：

表 3 出租车 GPS 数据格式

Car_ID	Time	Longitude	Latitude	Speed	Direction	State
105	2007-02-20 00:01:17	121.4681	31.2211	34	45	0
105	2007-02-20 00:01:33	121.4695	31.2216	34	45	0
105	2007-02-20 00:01:49	121.47	31.2216	4	112	0
105	2007-02-20 00:02:50	121.4695	31.2215	0	22	0
105	2007-02-20 00:03:51	121.4695	31.2218	4	157	0

其中，Car_ID 表示车辆编号，Time 表示定位时间：年-月-日 时-分-秒，Longitude 表示定位点的 GPS 经度，Latitude 表示定位点的 GPS 纬度，Speed 表示瞬时速度，单位为 Km/h，Direction 表示顺时针方向与北方的夹角，State 表示该出租车的当前状态，“0”表示该出租车是空的，“1”表示该出租车已搭乘乘客，其余所有数字都是为了模糊目的而随机设定的数字。

航班数据为 2007 年全国航班数据，字段包括 29 个字段，本文所使用的字段包括以下八个字段：出发城市、到达城市、航空公司、机型、起飞时间、降落时间、准点率和平均误点时间，其数据格式如下表 4 所示：

表 4 全国航班数据格式

出发城市	到达城市	航空公司	机型	起飞时间	降落时间	准点率	平均误点时间
贵阳	上海	海南航空	波音737(中)	21:35	00:10	0.76	17分钟
贵阳	上海	扬子江航空	波音737(中)	21:35	00:10	0.76	17分钟
昆明	上海	中国国航	空客320(中)	21:00	00:10	0.92	17分钟
昆明	上海	昆明航空	空客320(中)	21:00	00:10	0.93	11分钟
昆明	上海	深圳航空	空客320(中)	21:00	00:10	0.92	17分钟

选择上海浦东国际机场由于无法获得机场的实时人数数据，因此通过查询各种机型的载客人数，结合机型（表）和相应的航班到达时间获取每小时的进港客流量。

此处所计算的进港客流量是依据航班数所确定的。根据上海市交通委的新闻报道，“浦东机场白天约有 15%、夜间约有 40%的到港旅客需要乘坐出租车”，因此，最终每个时间段内的客流量可以用以下公式计算，

$$\text{Passenger\_flow}_i = \text{Passenger\_flight}_i \times D \times T_i$$

其中，Passenger_flow_i表示 i 时间段内等待坐出租车的客流量，$\text{Passenger_flight}_i$为 i 时间段内的出港人数，D 为上海浦东机场到港人数占所有到港人数的百分比,T 为到港人数中乘坐出租车的乘客百分比，当 8:00<i<20:00 时（白天），T_i=0.15，当 20:00<i<8:00 时（夜间），T_i=0.4。

结果如图蓝线所示。由于某个时间段的客流量是动态时间数据，其随着个人步速、是否托运行李、是否逛免税店有密切的关系，因此应将数据进行指数加权移动平均，使客流量数据更为稳定，较好的反应时间序列的变化趋势。另外，数据指数加权移动平均过程中权重的大小不同起到的作用也是不同，时间比较久远的变量值的影响力相对较低，时间比较近的变量值的影响力相对较高。在此处，将权重赋为 0.9。具体计算方法为：

$$p_t = \beta \times p_{t-1} + (1-\beta) \times \theta_t$$

其中，p_t表示经过滑动平均后 t 时刻的人数；β表示权重系数，此处取 0.9；p_{t-1}表示 t-1 时刻滑动平均后的的人数，θ_t表示 t 时刻原有人数。计算结果如下图 4 所示。

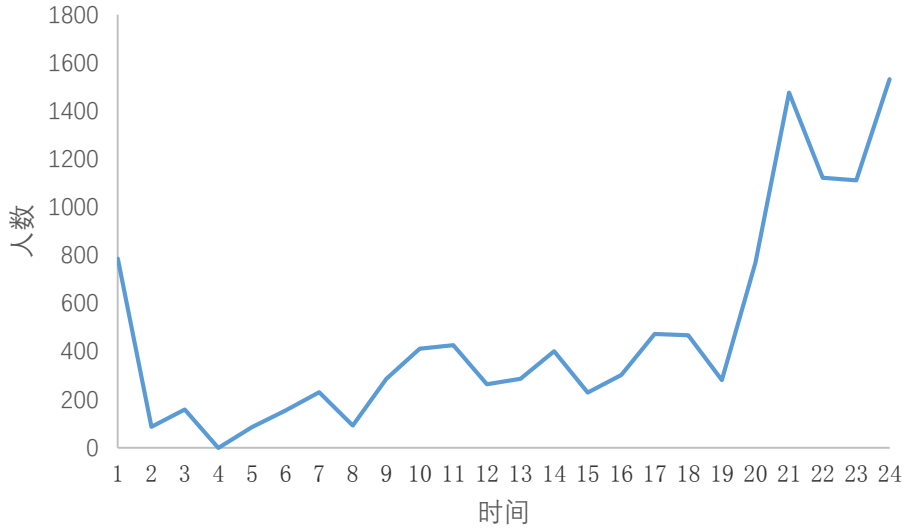


图 4 一天时间内客流量估计（2007 年）

6.2.2 数据预处理

由于 GPS 定位本身定位精度以及仪器记录过程中可能出现的问题，为了保证数据分析结果的准确性和可靠性，需要首先对数据进行清洗预处理。清洗的数据包括以下字段缺失数据和字段异常数据两类：

（1）处理字段缺失数据 剔除 Car_ID、Time、Longitude 和 Latitude 字段为 0 值的数据；

（2）字段异常数据处理 表示时间的 Time 字段应该为递增数据，因此应清除时间序列中出现递减的数据；剔除 Speed 字段全部相等的车辆；剔除 State 字段全为 0 和 1 的数据。

6.2.3 筛选数据

本题中我们的研究对象为 “送客到机场的出租车司机”，由于数据集中可能含有当天行驶路径未包含上海浦东国际机场的出租车数据。这些数据不在我们的研究范围内，因此需要通过筛选将不经过机场的出租车轨迹数据删除。通过 BIGEMAP 地图下载器所提供的高分辨率上海市遥感卫星图获得上海浦东机场的经纬度范围。如图 5 所示。为了便于数据处理，将其形状简化为长方形。浦东机场的经纬度范围如表 5 所示。

图 5 上海浦东机场范围示意图

表 5 浦东机场经纬度范围

点	纬度	经度
A	31.16728566	121.7725501
B	31.10587667	121.7930692
C	31.17790889	121.8114177
D	31.11975358	121.8475213

利用 Python 依次遍历每辆出租车的轨迹数据中的经纬度，将所有轨迹点都不位于机场范围内的出租车轨迹剔除，最终得到 233 条载客到浦东机场的出租车轨迹数据。

6.2.4 特征工程

本研究拟利用机器学习的方法对出租车司机的决策进行预测，因此需要获得与出租车司机决策有关的特征变量输入模型进行训练。如果选择的特征变量质量越好，那么得到的预测结果也会更准确。根据问题一影响司机决策的机理，可以将特征变量分为时间成本和载客收益。其中，时间成本通过出租车在机场等待的时间所确定，载客收益通过出租车载客后行驶的距离确定。

（1）计算时间成本

对于到达区排队并等待载客返回市区的出租车司机来说，其付出的时间成本即为在机场等待载客所消耗的时间。要计算出租车在机场等待时所消耗的时间，可以通过首先确定某一出租车离开机场时的点，该点的 State 字段属性值应为 1（载客状态），将满足此条件的出租车成为“目标出租车”。根据该点分别使用贪心算法向上、向下搜索 State 字段中与该点连续的属性值均为 1 的点的集合。这些点的集合是按照时间顺序排列的，在找出所有满足要求的轨迹点之后即可通过末态轨迹点的时间减去初态轨迹点的时间确定返程载客出租车所花费的时间。

（2）计算载客返回市区的出租车返程行驶距离

由 5.1 可获得载客回市区的各个出租车的轨迹，设每次定位时出租车的经度为x_{ij}，纬度为y_{ij}，则对于出租车 i，其在研究时段内所行驶的距离 d 为：

$$\mathrm{d}=\sum_{j=1}^{n}111.12\cos\{\frac{1}{sinx_{ij}\times sinx_{i(j+1)}+cosx_{ij}cosx_{i(j+1)}\cos(y_{i(j+1)}-y_{ij})}\}$$

利用车辆经纬度信息进行 Python 编程的到每个目标出租车在离开机场的第一段载客路程的距离。

6.2.5 DNN 神经网络

在本题数据中我们已知在数据集中特征与特征之间，即乘客数量的变化规律、司机收益和时间成本之间存在很强的非线性或未知关系，因此使用深度神经网络（DNN）拟合这种非线性或未知关系。其结构示意图如图 6 所示。DNN 为有很多隐藏层的神经网络，DNN 按不同层的位置划分，DNN 内部的神经网络层可以分为三类，输入层，隐藏层和输出层,如下图示例，一般来说第一层是输出层，最后一层是输出层，而中间的层数都是隐藏层。层与层之间是全连接的，也就是说，第 i 层的任意一个神经元一定与第 i+1 层的任意一个神经元相连。虽然 DNN 看起来很复杂，但是从小的局部模型来说，还是和感知机一样，即一个线性关系$\mathrm{z}=\sum w_i x_i+bz=\sum w_i x_i+b$，加上一个激活函数σ(z)。

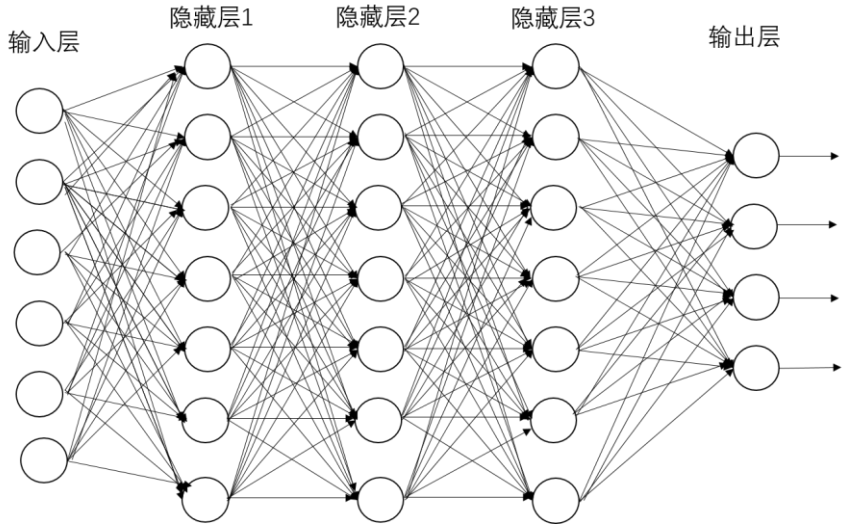


图 6 DNN 神经网络示意图

我们构建的 DNN 神经网络有 9 个神经元输入层，分别含有 6、4、2 个神经元的三个隐藏层以及含有两个神经元的输出层，利用 ADAM 优化器，迭代 200 次，L2 正则化系数为1×10^{-5}。利用 186 个数据训练神经网络，47 个数据对训练结果进行检验。设 47 个数据的预测结果分别是 pred(i)，实际结果为 fact(i),则由 error=$[\mathrm{pred(i)}-\mathrm{fact(i)}]^2$可得，error=5，即预测的准确率为 89.36%。

6.2.6 模型检验

(1) Logistic 回归模型

Logistic 回归是一直概率模型，它是以某一件事发生与否的概率 P 为因变量，以影响 P 的因素为自变量建立的回归模型，分析某事件发生的概率与自变量之间的关系，它是一种非线性的回归模型。

定义 X 为连续随机变量，X 服从 Logistic 分布，则 X 具有下列的分布函数和密度函数：

$$F(x) = P(X \le x) = \frac{1}{1 + e^{-\frac{x-\mu}{\gamma}}}$$

$$f(x) = \frac{e^{-\frac{x-\mu}{\gamma}}}{r(1 + e^{-\frac{x-\mu}{\gamma}})^2}$$

其中，μ为位置参数，γ为形状参数。

二项 Logistic 回归模型是一种分类模型，由条件 P(Y|X)表示，随机变量 Y 取 0 或 1. 定义二项 Logistic 回归模型的条件分布如下：

$$P(Y = 1|X) = \frac{e^{wx}}{1 + e^{wx}}$$

$$P(Y = 0|X) = \frac{1}{1 + e^{wx}}$$

其中，x∈Rn 是输入，Y∈{0,1}是输出，W∈Rn 和 b∈R 是参数，w 称为权重，b 称为偏置。

Logistic 回归主要适用于因变量 Y 为分类的资料，包括二元或多项分类资料。对于出租车司机的决策来说，即选择空车回城（0）或载客回城（1）的 0-1 二项分布资料。通过 Logistic 回归模型可以寻找某现象发生的影响因素，确定不同因素对某时间发生影响的相对重要性。

(2) 依赖性分析

利用 Logistic 回归分析建立回归模型，得出影响决策的各个因素与决策变量之间的依赖性关系如表 6 所示，

表 6 依赖性关系

影响因素	依赖性关系
Latitude	-0.46887
Car_ID	-0.13578
Longitude	-0.13221
Dirction	0.133191
Speed	0.140975
Time_Long	0.38455
Income	0.38455
Total_Path	3.595152

由表 5 可以看出，司机做出决策与载客路程的正依赖性关系最强，达到 3.595，其次为收益和时间成本，其依赖性关系均为 0.384。与方向和精度的依赖性关系最弱，只有 0.133191 和-0.13221。除此之外，与纬度的负依赖性最强，为-0.468。

6.3 模型对比

(1) 模型分析理论

模型一 基于博弈理论决策模型，分析模型的合理性并给出相关分析，流程图如图 7 所示：

图 7 数学模型建模流程图

模型二 基于 DNN 神经网络训练模型，分析模型的合理性，确定模型对相关因素的相关性，流程图如图 8 所示：

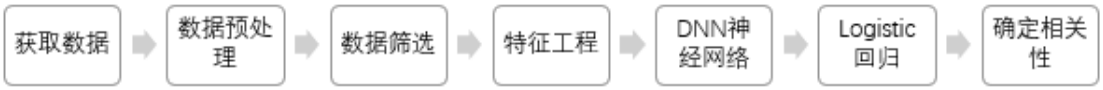


图 8 算法模型建模流程图

(2) 模型优缺点分析

表 7 模型对比分析

	基于博弈论的决策模型	DNN 神经网络模型
优点	1.不需要大量数据，不依赖数据； 2.在缺乏数据的条件下仍可以进行具有一定准确率的决策。	1.能够挖掘大数据下的潜在关系，不需要人为限制过多的因素； 2.易于商业化、市场化。
缺点	1.关系复杂，需考虑的条件、因素过多； 2.决策模型的准确度检验需要精确判断且计算复杂。	1.模型的求解需要大量的数据且计算复杂，运行时间长； 2.依赖特征工程形成较好的特征变量，往往会耗费大量时间、人力和物力。

七 问题三模型的建立与求解

7.1 基于排队论的上车点位置确定模型

排队服务系统包括输入过程、排队规则和服务规则三个部分。对于机场出租车的上客来说，即需要综合考虑构建以乘客、空车作为输入过程，停靠方式和乘客上车方式作为服务规则，泊位数量作为设施的排队规则。目前我国的出租车排队服务系统可以分为三类：单车道出租车排队服务系统、多车道（矩阵式）出租车排队服务系统、斜列式出租车排队服务系统。由于本题中机场“乘车区”现有两条并行车道，且斜列式排队服务系统和混合式出租车由于泊车位行列数较多，同时进入泊车位的乘客数和出租车数量过多，存在安全隐患问题。因此在本问题中仅考虑双向单车道或双车道出租车排队服务系统[4]。

单车道排队服务系统（如图 9 所示）适合小型机场或人流量较少的出租车上客区，其排队模型为 M/M/1 排队模型，即到达时间服从泊松过程，服务时间是指数分布，只有一部服务器并且遵循先到先服务的原则。这种情况下，当车流量和人流量增大时，将无法满足及基本的载客需求。

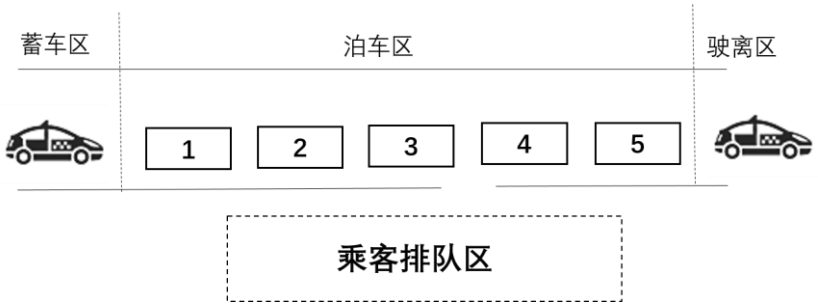


图 9 单车道出租车排队服务系统示意图

多车道（矩阵式）出租车排队服务系统拥有多个车道，具备 m× n 个泊车位，出租车按照进入系统的顺序依次进入泊位，待前两排车辆停稳后开放乘客进入通道，乘客从前往后选择出租车就坐。待乘客上车后，各个车道的车辆依次驶出。为了保证安全，待泊车位内的车辆全部驶出后令下一批出租车进入，依次循环往复。由于该系统泊车位较多，因此需配备两名秩序维持人员。一人负责泊车位上出租车的统一进入，另一人负责乘客的进入。如图 10 所示。

7.2 排队模型

7.2.1 泊松分布

设 N(t)表示在时间区间[0,t）内到达的乘客数，令$\mathrm{P}_n(t_1,t_2)$表示在时间区间[t_1，t_2）内有 n 个顾客到达的概率，即，

$$\mathrm{P}_n(t_1,t_2)=P\{N(t_1)-N(t_2)=n\},$$

在机场乘客搭乘出租车的过程中，由于该乘客流满足以下三个条件：

（1）在机场搭乘出租车的乘客在不叠加的时间区间内是相互独立的，没有后向效性；

（2）并且对于充分小的一段时间，在这一段时间区间内有一个乘客到达的概率与 t 无关，而与区间长度成正比,即

$$P_1(t,t+\Delta t)=\lambda\Delta t+o(\Delta t)$$

其中，$o(\Delta t)$，当$\Delta t\to 0$ 时，是关于Δt的高阶无穷小。$\lambda>0$，它表示单位时间有一个乘客到达的概率，称为概率强度。

（3）对于充分小的Δt，在时间区间[t,t+Δt)内有两个或两个以上乘客到达的概率极小，以至可以忽略，即

$$\sum_{n=2}^{\infty}P_n(t,t+\Delta t)=0(\Delta t)$$

综上所述，该乘客流可以看做泊松流，且乘客的相继到达的时间间隔服从指数分布。

7.2.2 繁忙率

排队系统的繁忙程度可以用繁忙率来表示，p 为系统中顾客到达率与服务率之比。

$$\mathrm{p}=\frac{\alpha}{\mathrm{B}\beta}$$

p 表示为排队系统的繁忙率，a 表示出租车乘客平均到达率，B 表示停车点的个数，β表示单个上车的平均服务率（单个停车点单位时间完成乘客的个数）

7.2.3 排队系统的运行指标

表 8 运行指标及其含义

名称	含义
平均队长	指系统内乘客数（包括正被服务的顾客与排队等待服务的顾客）的数学期望，记做 Ls
平均排队长	指系统内等待服务的乘客数的数学期望，记做 Lq
平均逗留时间	乘客在系统内逗留的时间（包括排队等待的时间和被服务的时间）的数学期望，记做 Ws
平均等待时间	一个乘客在排队系统中排队等待时间额数学期望，记做 Wq

7.2.4 建立排队模型

根据 2.3 的叙述，机场的出租车乘客流满足泊松分布的要求。因此可以根据建立排队模型，对系统的输入（乘客，空车）和输出（载客车）进行建模。当系统达到稳定状态且 C 个上车点同时工作时，排队模型中出租车乘客数为 n 的概率如下：

$$P_n(B)=\begin{cases}\dfrac{1}{n}\left(\dfrac{\alpha}{\beta}\right)^k P_0(B) & n=1,2\ldots,B\\ \dfrac{1}{B!\,B^{n-b}}\left(\dfrac{\alpha}{\beta}\right)P_0(B) & n=B+1\end{cases}$$

用乘车场中乘客排队的平均队长 Ls,和乘客的平均逗留时间 Ws

$$L_s=\frac{1}{C!}\frac{Bp^Bp}{(1-p)^2}P_0+\frac{\alpha}{\beta}$$

7.2.5 排队模型优化

为了使总的乘车效率达到最高，需要考虑乘客的时间成本以及每个上车点的服务成本。乘客所花费的时间成本可以表示为z_1=αLs，上车点的建设成本可以表示为z_2=βC，其中α为每个乘客单位时间等待时间成本，β为每个上车点的服务时间成本。要使总的成本达到最小，则时间成本和服务成本之和应该取到最小值：

$$\min Z(C)=\ \mu L_x(B)+\ \sigma B$$

7.2.6 模型求解

（1）乘客到达率α的计算

乘客到达率α为单位时间内进入排队系统的乘客人数。根据问题二中按小时统计的需要乘坐出租车的人数，求的每分钟内进入排队系统的平均值即为乘客到达率。利用 Excel 计算可得α = 8人/分钟。

（2）服务率β的计算

服务率β为单个上车点的平均服务率，即为离开机场时载客状态为 1 的出租车数量除以时间。由第二问的数据统计可得出租车数为 104 个，则$\beta=\frac{104}{60}$~2 人/分钟。

（3）上车点数计算

假设乘客等待时间成本与服务成本的比值即$\frac{\mu}{\sigma}$=400。由 Lingo 计算可得 C=8。即在考虑安全性和乘车效率最大化时，应设置 8 个上车点，如图 10 所示。

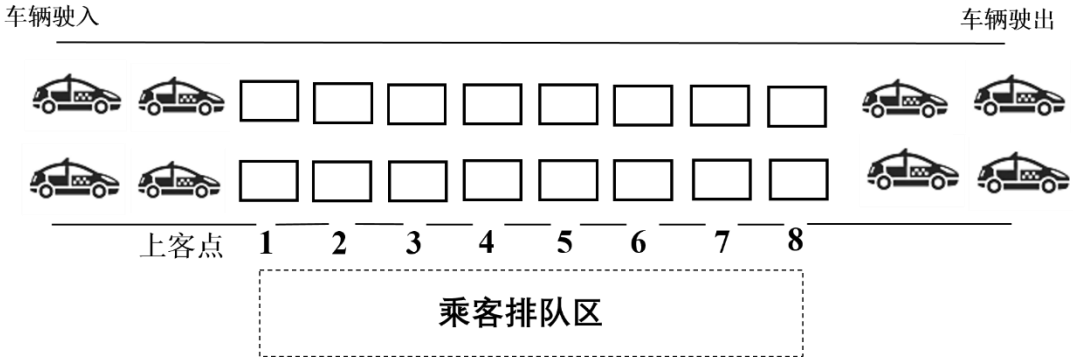


图 10 上海浦东机场出租车乘车点示意图

八 问题四模型的建立与求解

8.1 问题四模型的建立

对于机场出租车短途与长途载客，所获取的效益不同，所付出的时间成本不同，为了使出租车司机成本效益最大化，同时使出租车司机的收益尽量保持均衡，管理部门对不同里程运营的出租车到达机场时所采取的策略不同，给予车辆进入“蓄车池”的优先权不同。因此为对管理部门可行的“优先" 安排方案，建立需要从出租车的收益上考虑，建立优化模型。

首先，确定优化的目标函数：

$$Z=\max(z_i+z_j)$$

$$U=\min(|z_i-z_j|+L(\frac{1}{z_i}+\frac{1}{z_j}))$$

其中 Z 表示任意两辆出租车的最大收益，U 衡量的是任意两辆车的最小收益即为对应的损失函数，为了从数学上使本该趋于零的参数估计值尽可能向零靠近，在损失函数中加入惩罚项，其中 L 表示惩罚的权重。

当出租车从机场载客到任意位置时，出租车有两种选择返回市区拉客或前往机场接客，令出租车前往市区的概率为 p(x)，前往机场的概率为 1-p(x)，选择的概率与出租车区里机场的距离 x 有关，对出租车所产生的收益函数：

$$z_i = r_0 + rx_i - p(x_i)cv(l - x_i) - (1 - p(x_i))cvx_i + (1 - p(x_i))wT(n)$$
$$z_j = r_0 + rx_j - p(x_j)cv(l - x_j) - (1 - p(x_j))cvx_j + (1 - p(x_j))wT(n)$$

上式各项的物理意义为右边第一项代表出租车从机场载客运营行驶一定距离时所产生的效益，第二、三项表示出租车将乘客送到目的地后，分别前往市区、机场空载所产生的损失成本，第四项表示出租车前往机场载客时，依据机场管理部门的“优先权”策略，减少司机在“蓄车池”的待客时间所产生的相对收益。其中 r 为价格费率，v 是耗油率，c 表示相应的油价，l 为机场到市区之间的距离，n 表示出租车到达机场后，管理部门根据其运营里程判断，按照优先策略，安排出租车在“蓄车池”位点，T(n)表示在不同候车位点所节省的时间成本效益。

考虑机场“蓄车池“所能容纳的出租车数量为 100，即最大候车位点为 100，即有：

$$1 \le n \le 10$$

在所获取的 2007 年上海浦东机场出租车运营数据中，统计不同出租车的运营里程，做出折现图，如图 11 所示：

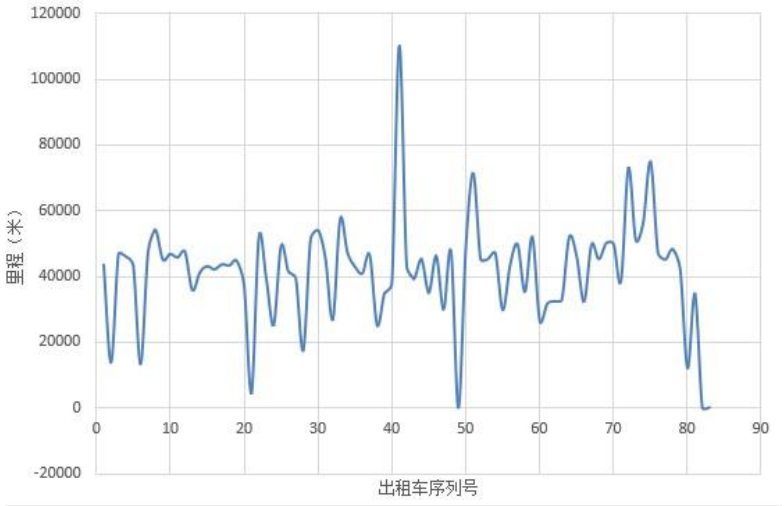


图 11　出租车在机场载客后运营里程的统计

从统计图中，可以看到出租车的运营里程大部分集中 20-40 公里区间，以此为范围来帮助机场管理部门开展优先方案：

当$x \leq 20km$时，可视为短途载客；

当$20km<x<40km$时，视为正常运营载客；

当$x \geq 40km$时，视为长途载客。

在不同划分范围内，综合考虑优化目标下，管理部门给与不同程度最佳的优先策略。

8.2 问题四模型的求解

在模型中的效益函数中，为了对目标函数进行求解，需要确定相关的参数，这里给出所需要的参数数值，如表 9 所示。

表 9 “优先”方案中参数信息

参数符号	参数意义	参数数值
r	平均价格费率	2.1元/千米
r_0	起步价格	12元
c	汽油价格	3.8元/升
l	机场到市区的距离	40千米
p	出租车前往市区的概率	1/2

费油率 v 可由问题一中确定的函数关系（2）、（3），并结合相关参数进行确定，在不同候车位点所节省的时间成本效益 T(n)同样可由问题一所建立的待客时间成本函数关系(1)确定。

为了求得目标函数的最优解，采用遗传算法[5]来寻得最优解，具体的算法流程图如图 12 所示。

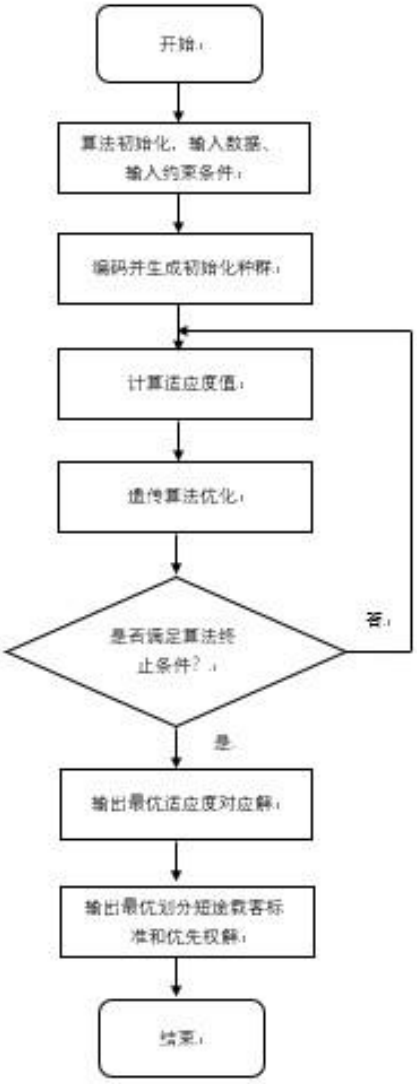


图 12 遗传算法流程图

设置变量个数为 2 即“蓄车池”位点 n、距离机场的位置 x，确定种群规模为 100，交叉概率为 0.6，变异概率为 0.01，遗传运算的终止进化代数为 100。用 python 进行编程实现，输入 2007 年出租车自机场出发的运营里程数据即种群规模，根据“优先“安排方案，可以得到当出租车距离机场的距离为 31km 时，安排所在蓄车池的位点为第 41 个时，此时可以得到任意两辆出租车司机的最大效益的收益最大分别为 94 元、74 元，此时有最小的收益差距。

九 模型推广

模型名称	推广
模型一	该模型依据博弈理论，建立多目标规划模型，考虑不同因素对出租车效益的影响，并建立相应的函数关系，当相关因素发生变化时，利用函数关系依然可以确定相关因素对效益的影响程度，从而在目标函数取得最优解时，可以给出司机的选择策略。
模型二	该模型采用深度神经网络具有很强的表达性，可以广泛的使用于有关大数据挖掘等方向问题，并且该模型可以完成端到端的学习任务，从而减少传统的机器学习中有关特征工程的问题。
模型三	该模型使用 M/M/C 排队模型，可以将其用于服务的对象以及服务者无穷的情况下，并且该模型对于数据的依赖性不大，可以通过其获得较好的结果。另外，可以用于有关设置用户电车充电点的问题中。
模型四	该模型使用遗传算法解决解存在过多的情况，遗传算法不但可以作为一个求解优化的问题，还可以使用于有关特征工程的问题

十 参考文献

[1]许伦辉,郭雅婷,黄宝山,李雪.基于多目标模型的城市公交越站调度研究[J].内蒙古公路与运输,2019(03):48-52+62.

[2]甘家瑞.基于计算博弈论的出租车市场最优定价研究[D].2015.7714

[3]彭辉. 空港综合交通枢纽客流预测分析[A].中国科学技术协会、交通运输部、中国工程院:中国公路学会,2018:11.

[4]魏中华,王琳,邱实.基于排队论的枢纽内出租车上客区服务台优化[J].公路交通科技(应用技术版),2017,13(10):298-300.

[5]李旭阳,田铭兴,孙立军,潘存磊.城市轨道交通列车定时节能优化方法研究[J/OL].铁道标准设计:1-7[2019-09-15]

十一 附录

表 1 机型与乘客数关系

机型	人数范围	计算值
CRJ（小）	86	86
ERJ（小）	78	78
ERJ-190（中）	98-114	106
JET	50-60	55
波音 737（中）	130-160	145
波音 747（大）	350-400	375
波音 757（中）	200-228	214
波音 767（大）	203-290	240
波音 777（大）	350	350
波音 787（大）	242-335	288
空客 319（中）	112-134	123
空客 320（中）	123-180	150
空客 321（窄体机）	174-220	175
空客 321（中）	174-220	190
空客 330（宽体机）	256-412	305
空客 380（大）	555	555
庞巴迪 CRJ900	50-86	70
新舟 60（小）	50-60	55
其他机型	267	250

```
import numpy as np
import pandas as pd
import os


# 11 2.1
data = pd.read_excel("last_one_data_path_time.xlsx")
go_taxi = r"go_taxi"
back_taxi = r"back_taxi"

# 以路程作为比较的标准
data["Time_Cost"] = 0
```

```
    data["Income"] = 0

    # 计算出租车的时间损失
    for i in range(len(data)):
        num_id = data["Vehicle_SimID"][i]
        if data["Passenger_State"][i] == 1:
            average_v = np.average(pd.read_excel(os.path.join(go_taxi, "Taxi_" +
str(num_id)+".xlsx"))["GPS_Speed"])
            data["Time_Cost"] = data["Time_Long"] / 60 * average_v * 1000
        else:
            average_v = np.average(pd.read_excel(os.path.join(back_taxi,
"Taxi_"+str(num_id)+".xlsx"))["GPS_Speed"])
            data["Time_Cost"] = data["Time_Long"] / 60 * average_v * 1000

    # 计算出租车收益
    for i in range(len(data)):
        if data["Total_Path"][i] > 0:
            if data["Total_Path"][i] / 1000 <= 3:
                data["Income"][i] = 11
            else:
                data["Income"][i] = 11 + (data["Total_Path"][i] / 1000 - 3) * 2.1

    data.to_excel("last_result.xlsx",index=None)


    ///
    import numpy as np
    import pandas as pd
    import os
    from math import radians, cos, sin, asin, sqrt


    def haversine(lon1, lat1, lon2, lat2):  # 经度1，纬度1，经度2，纬度2 （十进制度
数）
        """
        Calculate the great circle distance between two points
        on the earth (specified in decimal degrees)
        """
        # 将十进制度数转化为弧度
        lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
```

```
        # haversine 公式
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
        c = 2 * asin(sqrt(a))
        r = 6371  # 地球平均半径，单位为公里
        return c * r * 1000


    path_total = {}
    path_path = r"path_taxi"
    out_path = r"cal_path"
    all_data = os.listdir(path_path)
    label = ["Vehicle_SimID", "GPS_Time", "GPS_Longitude", "GPS_Latitude",
"GPS_Speed", "GPS_Direction",
             "Passenger_State"]

    for file in all_data:
        df = pd.read_excel(os.path.join(path_path, file))
        tot = 0
        for i in range(1, len(df)):
            l = haversine(float(df["GPS_Longitude"][i]), float(df["GPS_Latitude"][i]),
float(df["GPS_Longitude"][i - 1]),
                          float(df["GPS_Latitude"][i - 1]))
            tot += l
        print(str(file.split("_")[1].split(".")[0]), tot)
        path_total[file.split("_")[1].split(".")[0]] = tot

    data = pd.read_excel("last_one_data.xlsx")
    data_new = data.copy()
    data["Total_Path"] = 0.0
    for i in range(len(data)):
        if str(data.loc[i, :].Vehicle_SimID) in path_total.keys():
            data["Total_Path"][i] = path_total.get(str(data.loc[i, :].Vehicle_SimID))
    # print(data)
    data.to_excel("last_one_data_path.xlsx", index=None)
    # 11 2.1


    a = [
        786,
```

```
    88,
    148,
    0,
    74,
    148,
    223,
    93,
    287,
    412,
    427,
    265,
    287,
    401,
    230,
    303,
    473,
    468,
    282,
    770,
    1476,
    1123,
    1112,
    1532,
]
theta = 0
# v1 = 0.8 * a[0] + 0.2 * theta
# v2 = 0.8 * a[1] + 0.2 * v1
# v3 = 0.8 * a[2] + 0.2 * v2


for i, v in enumerate(a):
    theta = 0.98 * a[i] + 0.02 * theta
    a[i] = theta
print(a)



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

POP_SIZE = 100  # population size
```

```
    CROSS_RATE = 0.6  # mating probability (DNA crossover)
    MUTATION_RATE = 0.01  # mutation probability
    N_GENERATIONS = 300  # the times

    #
    DNA_SIZE = 2
    ASCII_BOUND = [20, 40, 20, 100]


    class GA:
        def __init__(self, DNA_size, DNA_bound, cross_rate, mutate_rate, pop_size):
            self.DNA_size = DNA_size
            DNA_bound[1] += 1
            DNA_bound[3] += 1
            self.DNA_bound = DNA_bound
            self.cross_rate = cross_rate
            self.mutate_rate = mutate_rate
            self.pop_size = pop_size
            self.data = pd.read_excel("last_result.xlsx")
            self.data = self.data[self.data["Passenger_State"] == 1]
            # self.pop = np.random.randint(*DNA_bound, size=(pop_size,
DNA_size)).astype(np.int8)
            self.pop = np.hstack((np.random.randint(self.DNA_bound[0],
self.DNA_bound[1], size=(self.pop_size, 1)),
                                  np.random.randint(self.DNA_bound[2],
self.DNA_bound[3], size=(self.pop_size, 1))))

        def get_fitness(self):
            match_count = np.zeros((self.pop_size, 1))
            cnt = 0
            for one in self.pop:
                x = one[0]
                n = one[1]
                h_data = self.data[(self.data["Total_Path"] / 1000 > x) &
(self.data["Total_Path"] != 0)].iloc[
                         np.random.randint(
                             len(self.data[(self.data["Total_Path"] / 1000 > x) &
(self.data["Total_Path"] != 0)])), :]
                l_data = self.data[(self.data["Total_Path"] / 1000 <= x) &
(self.data["Total_Path"] != 0)].iloc[
                         np.random.randint(
```

```
                            len(self.data[(self.data["Total_Path"] / 1000 <= x) &
(self.data["Total_Path"] != 0)])), :]
                l_C1 = 0
                h_C1 = 0
                if l_data["Total_Path"] / 1000 <= 3:
                    l_C1 = 11
                elif l_data["Total_Path"] / 1000 > 3:
                    l_C1 = 11 + (int(l_data["Total_Path"] / 1000) + 1) * 2.1
                l_C1 += n
                if h_data["Total_Path"] / 1000 <= 3:
                    h_C1 = 11
                elif h_data["Total_Path"] / 1000 > 3:
                    h_C1 = 11 + (int(l_data["Total_Path"] / 1000) + 1) * 2.1
                match_count[cnt][0] = (l_C1 + h_C1) / (2 * l_C1 * h_C1) + (np.abs(h_C1
- l_C1))
                cnt += 1
            return match_count

        def select(self):
            fitness = self.get_fitness() + 1e-4
            fitness = fitness.ravel()
            idx = np.random.choice(np.arange(self.pop_size), size=self.pop_size,
replace=True, p=fitness / fitness.sum())
            return self.pop[idx]

        def crossover(self, parent, pop):a$
            if np.random.rand() < self.cross_rate:
                i_ = np.random.randint(0, self.pop_size, size=1)
                cross_points = np.random.randint(0, 2, self.DNA_size).astype(np.bool)
                parent[cross_points] = pop[i_, cross_points]
            return parent

        def mutate(self, child):
            for point in range(self.DNA_size):
                if np.random.rand() < self.mutate_rate:
                    child[point] = np.random.randint(self.DNA_bound[2 * point],
self.DNA_bound[2 * point + 1])
            return child

        def evolve(self):
            pop = self.select()
```

```
            pop_copy = pop.copy()
            for parent in pop:
                child = self.crossover(parent, pop_copy)
                child = self.mutate(child)
                parent[:] = child
            self.pop = pop


    if __name__ == '__main__':
        ga = GA(DNA_size=DNA_SIZE, DNA_bound=ASCII_BOUND, cross_rate=CROSS_RATE,
mutate_rate=MUTATION_RATE,
                pop_size=POP_SIZE)
        nn = []
        kk = []
        data = pd.read_excel("last_result.xlsx")
        data = data[data["Passenger_State"] == 1]

        for generation in range(N_GENERATIONS):
            fitness = ga.get_fitness()
            best_DNA = ga.pop[np.argmin(fitness)]
            print("Gen", generation, ": ", best_DNA)
            x = best_DNA[0]
            n = best_DNA[1]
            h_data = data[(data["Total_Path"] / 1000 > x) & (data["Total_Path"] !=
0)].iloc[
                     np.random.randint(
                         len(data[(data["Total_Path"] / 1000 > x) &
(data["Total_Path"] != 0)])), :]
            l_data = data[(data["Total_Path"] / 1000 <= x) & (data["Total_Path"] !=
0)].iloc[
                     np.random.randint(
                         len(data[(data["Total_Path"] / 1000 <= x) &
(data["Total_Path"] != 0)])), :]
            l_C1 = 0
            h_C1 = 0
            if l_data["Total_Path"] / 1000 <= 3:
                l_C1 = 11
            elif l_data["Total_Path"] / 1000 > 3:
                l_C1 = 11 + (int(l_data["Total_Path"] / 1000) + 1) * 2.1
            l_C1 += n
            if h_data["Total_Path"] / 1000 <= 3:
```

```
                h_C1 = 11
            elif h_data["Total_Path"] / 1000 > 3:
                h_C1 = 11 + (int(l_data["Total_Path"] / 1000) + 1) * 2.1
            print(l_C1, h_C1)
            print((np.abs(h_C1 - l_C1)))
            nn.append(best_DNA[0])
            kk.append(best_DNA[1])
            ga.evolve()
        plt.plot(np.arange(len(nn)), nn)
        plt.plot(np.arange(len(kk)), kk)
        plt.show()

    import numpy as np
    import pandas as pd
    import os

    go_path = r"go_taxi"
    path_path = r"pre_taxi"
    out_path = r"path_taxi"
    all_data = os.listdir(go_path)
    label = ["Vehicle_SimID", "GPS_Time", "GPS_Longitude", "GPS_Latitude",
"GPS_Speed", "GPS_Direction",
             "Passenger_State"]
    cnt = 0
    line = 0
    for file in all_data:
        go_data = pd.read_excel(os.path.join(go_path, file)).iloc[-1, :]
        data = pd.read_excel(os.path.join(path_path, file))
        data.columns = label
        flag = True
        left_data = []
        n_flag = True
        for i in range(len(data)):
            if (data.loc[i, :].GPS_Time != go_data["GPS_Time"]) and n_flag:
                pass
            else:
                n_flag = False
                if (data.loc[i, :].GPS_Time == go_data["GPS_Time"] and flag) or (
                        data.loc[i, :].Passenger_State == 1 and flag):
                    left_data.append(data.loc[i, :])
                    line += 1
```

```
                else:
                    flag = False
        data = pd.DataFrame([i for i in left_data])
        data.to_excel(os.path.join(out_path, file),index=None)
        cnt += 1

    print("处理文件数量为:", cnt)
    print("行数,", line)

    import pandas as pd
    import numpy as np
    import os

    # 依次导入数据集
    data_path = r"taxi\Taxi_070220"
    out_path = r"pre_taxi"
    all_data = os.listdir(data_path)
    label = ["Vehicle_SimID", "GPS_Time", "GPS_Longitude", "GPS_Latitude",
"GPS_Speed", "GPS_Direction",
             "Passenger_State"]

    """
    浦东机场 经度纬度范围
    31.17790889 121.7725501     31.17790889 121.8475213
    31.10587667 121.7725501     31.10587667 121.8475213
    """
    # 经过机场的车辆数
    cnt_in = 0
    for i in all_data:
        num = i.split("_")[1]
        # 分别读入数据
        df = pd.read_csv(os.path.join(data_path, i))
        # 给数据添加标签
        df.columns = label
        # 数据预处理
        # 字段缺失数据处理
        df.dropna(subset=["Vehicle_SimID", "GPS_Time", "GPS_Longitude",
"GPS_Latitude"], inplace=True)
        # 判断车辆是否经过浦东机场
        GPS_Longitude_in = ((121.7725501 <= df["GPS_Longitude"]) &
(df["GPS_Longitude"] <= 121.8475213))
```

```
        GPS_Latitude_in = ((31.10587667 <= df["GPS_Latitude"]) & (df["GPS_Latitude"]
<= 31.17790889))
        GPS_in = GPS_Latitude_in & GPS_Longitude_in
        # 经过机场的车辆 , 字段异常数据处理
        if np.sum(GPS_in.astype(int)) != 0 and df["GPS_Time"].is_monotonic_increasing
and np.sum(
                (df["Passenger_State"]) == 2) != len(df["GPS_Latitude"]) and
np.sum((df["Passenger_State"]) == 0) != len(
            df["GPS_Latitude"]):
            cnt_in += 1
            # print(df.describe())
            df.to_excel(os.path.join(out_path, "Taxi_" + str(num) + ".xlsx"),
index=None)

    print("数据预处理完成")
    print("剩余可用文件数量为：", cnt_in)



    from pulp import *

    prob = LpProblem("problem1", LpMaximize)
    x1 = LpVariable("x1", 0, None, LpContinuous)
    x2 = LpVariable("x2", 0, None, LpContinuous)
    prob += 40 * x1 + 90 * x2
    prob += 9 * x1 + 7 * x2 <= 56
    prob += 7 * x1 + 20 * x2 <= 70
    prob += x1 + 0 * x2 <= 4
    prob += 0 * x1 + x2 <= 2
    prob.writeLP("problem1.lp")
    prob.solve()
    print("最大值 z 为，", value(prob.objective), "个单位")
    for v in prob.variables():
        print("最优值", v.name, ":", v.varValue, "个单位")

    prob = LpProblem("problem2", LpMaximize)
    C = LpVariable("C", 0, None, LpContinuous)

    import numpy as np
    import pandas as pd
    from sklearn.svm import SVC
    from sklearn.preprocessing import StandardScaler
```

```
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import classification_report
    from sklearn.neural_network import MLPClassifier
    from sklearn.linear_model import LogisticRegression

    labels = ["Vehicle_SimID", "GPS_Longitude", "GPS_Latitude", "GPS_Speed",
"GPS_Direction",
              "Total_Path", "Time_Long", "Time_Cost", "Income"]
    data = pd.read_excel("last_result.xlsx")
    x = data[labels]
    label = data["Passenger_State"]
    data = pd.concat([x, label], axis=1)
    train_data, test_data = train_test_split(data, train_size=0.8, random_state=42)
    train_x = train_data.to_numpy()[:, 0:8]
    train_y = train_data.to_numpy()[:, 9]
    test_x = test_data.to_numpy()[:, 0:8]
    test_y = test_data.to_numpy()[:, 9]
    scale = StandardScaler().fit(train_x)
    train_x = scale.transform(train_x)
    test_x = scale.transform(test_x)

    # turn_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100,
1000]},
    #                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
    # # clf 分离器
    # # 使用网格搜索法调超参数
    # # 训练集做 5 折交叉验证
    # # clf = GridSearchCV(SVC(C=1), turn_parameters, cv=5)
    # clf = SVC(C=1000, gamma=0.001, kernel="rbf")
    # # 用前一半 train 数据再做 5 折交叉验证
    # # 因为之前的 train_test_split 已经分割为 2 份了
    # # fit-拟合
    # clf.fit(train_x, train_y)
    # # 超参数
    # print(clf.best_params_)
    # # 得分
    # for scores in clf.cv_results_["mean_test_score"]:
    #     print(scores)
    # 分类报告
    log = LogisticRegression()
```

```
    log.fit(train_x, train_y)
    print(log.coef_)
    # clf = MLPClassifier(solver="adam", alpha=1e-5, hidden_layer_sizes=(6, 4, 2),
random_state=1)
    # clf.fit(train_x, train_y)
    # print(clf.n_iter_)
    # y_true, y_pred = test_y, clf.predict(test_x)
    # print(y_true)
    # print(y_pred)
    # print(classification_report(y_true, y_pred))



    import numpy as np
    import pandas as pd
    import os

    go_path = r"go_taxi"
    back_path = r"back_taxi"
    all_go_data = os.listdir(go_path)
    all_back_data = os.listdir(back_path)
    label = ["Vehicle_SimID", "GPS_Time", "GPS_Longitude", "GPS_Latitude",
"GPS_Speed", "GPS_Direction",
             "Passenger_State", "Total_Path"]
    # 没拉客每个车在机场的花费的时间
    time_in_place = {}
    # 拉客每个车在机场的花费的时间
    time_not_place = {}

    for file_go in all_go_data:
        df = pd.read_excel(os.path.join(go_path, file_go))
        time = (pd.to_datetime(df["GPS_Time"].max()).minute -
pd.to_datetime(df["GPS_Time"].min()).minute) + (
                pd.to_datetime(
                    df["GPS_Time"].max()).hour -
pd.to_datetime(df["GPS_Time"].min()).hour) * 60
        time_in_place[df["Vehicle_SimID"][0]] = time

    for file_back in all_back_data:
        df = pd.read_excel(os.path.join(back_path, file_back))
        ime = (pd.to_datetime(df["GPS_Time"].max()).minute -
pd.to_datetime(df["GPS_Time"].min()).minute) + (
```

```
                pd.to_datetime(
                    df["GPS_Time"].max()).hour -
pd.to_datetime(df["GPS_Time"].min()).hour) * 60
        time_not_place[df["Vehicle_SimID"][0]] = time

    # 读入已知的数据集
    data = pd.read_excel("last_one_data_path.xlsx")
    data["Time_Long"] = -1
    for i in range(len(data)):
        if not time_in_place.get(data["Vehicle_SimID"][i], None) is None:
            data["Time_Long"][i] = time_in_place.get(data["Vehicle_SimID"][i], None)
        if not time_not_place.get(data["Vehicle_SimID"][i], None) is None:
            data["Time_Long"][i] = time_not_place.get(data["Vehicle_SimID"][i], None)

    data.to_excel("last_one_data_path_time.xlsx", index=None)

    import pandas as pd
    import numpy as np
    import os

    # 依次导入数据集
    data_path = r"pre_taxi"
    out_path1 = r"go_taxi"
    out_path2 = r"back_taxi"
    all_data = os.listdir(data_path)
    label = ["Vehicle_SimID", "GPS_Time", "GPS_Longitude", "GPS_Latitude",
"GPS_Speed", "GPS_Direction",
             "Passenger_State"]

    # 参数
    # a_为在出租车在机场的时间中的最后的 0.3 时间中
    a_ = 0.3
    cnt1 = 0
    cnt2 = 0
    a = []
    # data = pd.DataFrame(columns=label)
    for file in all_data:
        num = file.split("_")[1]
        df = pd.read_excel(os.path.join(data_path, file))
        # 限定为在机场内的所有时刻
```

```
        GPS_Longitude_in = ((121.7725501 <= df["GPS_Longitude"]) &
(df["GPS_Longitude"] <= 121.8475213))
        GPS_Latitude_in = ((31.10587667 <= df["GPS_Latitude"]) & (df["GPS_Latitude"]
<= 31.17790889))
        GPS_in = GPS_Latitude_in & GPS_Longitude_in
        in_data = df[GPS_in]
        if 1 == ((in_data["Passenger_State"].values[::-1])[0]):
            # in_data.to_excel(os.path.join(out_path1, "Taxi_" + str(num)),
index=None)
            # 提取所有文件的最后一行数据
            a.append(in_data.iloc[-1, :])
            cnt1 += 1
        else:
            # in_data.to_excel(os.path.join(out_path2, "Taxi_" + str(num)),
index=None)
            # 提取所有文件的最后一行数据
            a.append(in_data.iloc[-1, :])
            cnt2 += 1
    # 提取所有文件的最后一行数据
    data = pd.DataFrame([i for i in a])
    # 提取所有文件的最后一行数据
    data.to_excel("last_one_data.xlsx",index=None)
    print("go", cnt1)
    print("back", cnt2)

    model:
    lamda=6;mu=4;rho=lamda/mu;
    P_wait=@peb(rho,s);
    L_q=P_wait*rho/(s-rho);
    L_s=L_q+rho;
    min=2*s+1000*L_s;
    @gin(s);@bnd(2,s,7);
    end
```