“拍照赚钱”的任务定价策略研究

摘要

“拍照赚钱”是移动互联网下的一种自助式服务模式。这种基于移动互联网的自助式劳务众包平台，为企业提供各种商业检查和信息搜集，相比传统的市场调查方式可以大大节省调查成本，而且有效地保证了调查数据真实性，缩短了调查的周期。因此 APP 成为该平台运行的核心，而 APP 中的任务定价又是其核心要素。

在问题一，我们首先利用随机森林建立判别任务是否能完成的模型，用于后续的分析和评估。接着我们用此判别模型，以及绘制图表，对任务未完成的原因做了具体分析，得出的原因为任务点会员分布不均、任务定价与当地消费水平不匹配、会员存在限额等问题。

在问题二，为了制定新的定价策略，我们对已完成任务的数据进行拟合，这其中采用了多项式回归与岭回归，其在测试集上的准确率达到了 80.14%。接下来我们用所有任务的数据对新的定价策略进行评估，得到最后的完成率为 79.28%，提高了 17%，且总价也降低了近 200 元。

在问题三，为了进一步提高任务的完成率，借鉴了商品的打包销售方法，我们给出四条具体的打包原则：（1）打包后的定价要大于打包之前每个的定价，同时要小于各任务点的定价总和。（2）打包的任务点之间的距离要尽可能的小，同时会员到打包的所有任务点的距离不宜超过 5km（3）打包时尽量使没有完成的任务与完成的打包。（4）打包的总的任务数不宜过大，不能超过会员的限额。按照打包的原则我们把多个任务整合到一起，并把整体看作一个新的任务。

在问题四，将打包后的整体，放进原来根据附件一中的 SVR 回归模型进行训练，得到完成率的预测值，在附件三测试中的完成率提升不大，成本降低约 2 万元。综上，定价模型考虑了会员密集程度，任务集中度，任务难易情况，区域分布等因素，在任务难以完成时，一定范围内提升定价从而起到提高完成率的作用。

关键词　随机森林，多项式回归，岭回归，K-Means，SVR 回归

1 问题重述

2 问题一的建模求解

2.1 判断任务是否能完成

为了分析任务未完成的原因，以及后续评价新的定价标准，我们首先建立一个模型，用于判断某一个任务是否能被完成。

2.1.1 参数的选择

为了建立模型，首先需要确定决定模型的参数，通过分析题中所给出的条件，可以得到一个任务是否能被完成与任务位置、任务附近的会员数以及任务定价有关。任务定价可以直接得到，而任务位置与附近会员数需要进一步处理，下面将详细阐述处理过程。

任务的地理位置分析 考虑到不同的地理位置因为不同的消费水平、地理环境等条件，会对任务是否完成造成影响，所以需要将地理位置也作为参数。题中原始任务位置为经纬度，并不能直观的反映某一片区域的特性。

因此，我们使用 K-Means 算法，对所有任务点进行聚类，从而得到若干个集中区域。每个区域内的任务点具有地理位置的相似性。最终以区域编号作为参数传入模型。

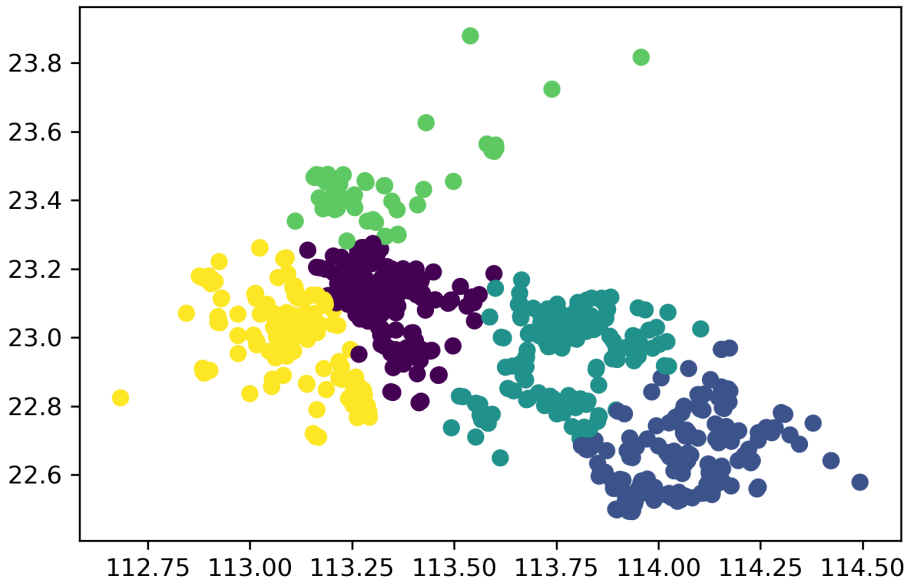

图 1: K-Means 聚类结果

如图所示，K-Means 聚类将所有任务点分为 5 类，每一类用不同的颜色表示。与实际地图对比观察，可以发现与现实中的城市分布基本吻合，从而可以验证聚类的正确性。

附近会员密集程度和定价分析 对每个任务周边一定范围内的会员人数进行统计，并绘制任务周边会员人数与定价的散点图。调整适当的范围大小，可以得到体现密集程度和定价的散点图如下：

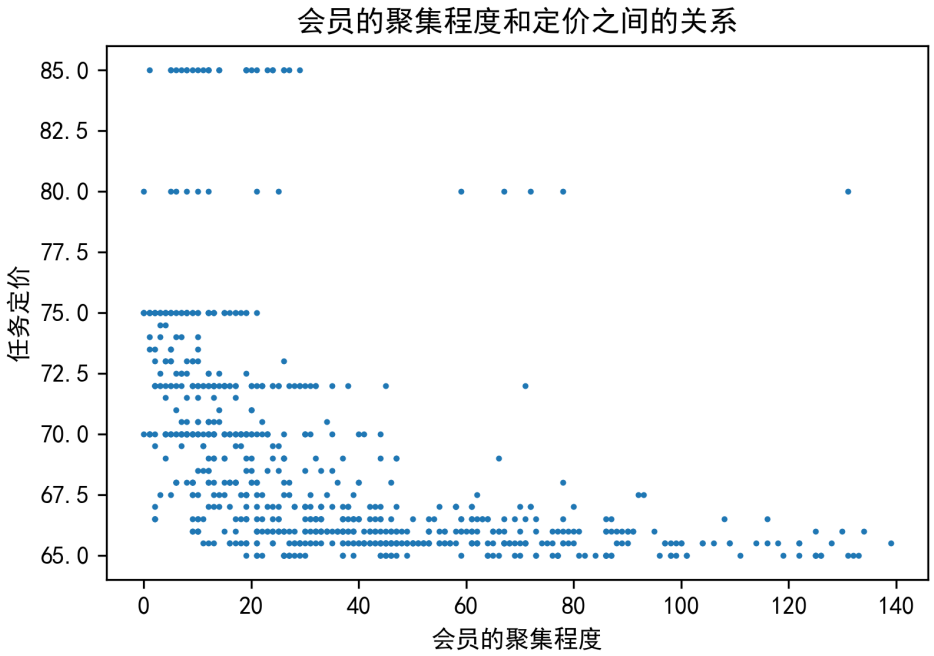

图 2: 会员的聚集程度和定价之间的关系

会员密集程度反映了会员在某一区域内的集中情况。从图中可以大概判断，会员集中程度越高，该任务被完成的可能性越大，此时该任务的定价就越低，附近会员数与定价大致呈负相关。

任务密集程度和定价分析 按照常识，对于相对集中的任务，更倾向于去一起完成，因此考虑到 app 成本的问题，可以将任务位置相近的定价设置低一些。

为了验证此猜想，绘制了每个任务密集程度，即任务一定范围内的其他任务数量，与定价之间的关系。调整实当的距离范围时，可以得到体现任务密集程度与定价关系的散点图如下：

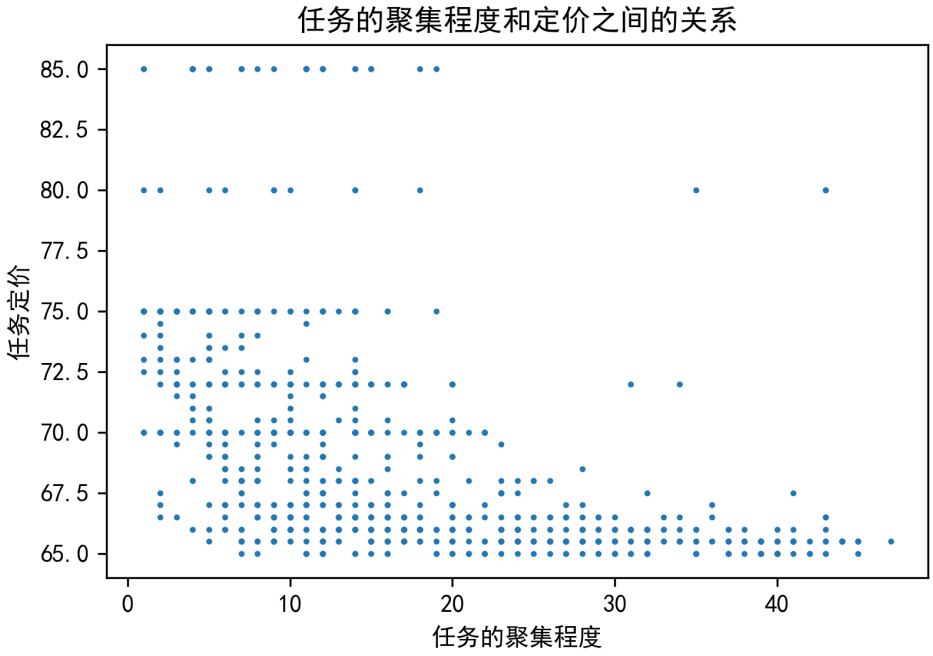

图 3: 任务的聚集程度和定价之间的关系

任务密集程度反映了任务在某一区域内的集中情况。任务的集中程度越高，供用户挑选的任务也就越多，在集中度高，距离近的情况下，任务的定价较低。

任务附近会员系数的衡量 任务点旁的会员数量、会员信誉度等对任务是否能完成也有着至关重要的影响。若附近会员数较多，则该任务就更有可能被完成。为了更加准确的刻画会员的相关影响，我们定义了三个会员系数 v_1, v_2, v_3：

$$v_1 = \sum_{x \in V}, \ v_2 = \sum_{x \in V} Q_x, \ v_3 = \sum_{x \in V} C_x \tag{1}$$

其中，V 为某一任务点五公里内的会员集合，Q_x 为会员 x 的任务限额，C_x 为会员 x 的信誉度。

2.1.2 模型的建立

考虑到目前已有数据量较小，且参数较少，为了避免过拟合现象，使模型的泛化性更强，我们采用随机森林模型。

在具体训练时，将已有数据按 4:1 的比例划分为训练集和测试集。训练集用于训练模型，测试集用于检验模型的有效性。最终，我们的模型在训练集上的准确率达到 98.27%，在测试集上的准确率达到 87.26%。考虑到拥有的数据量较少，已经达到了不错的效果。

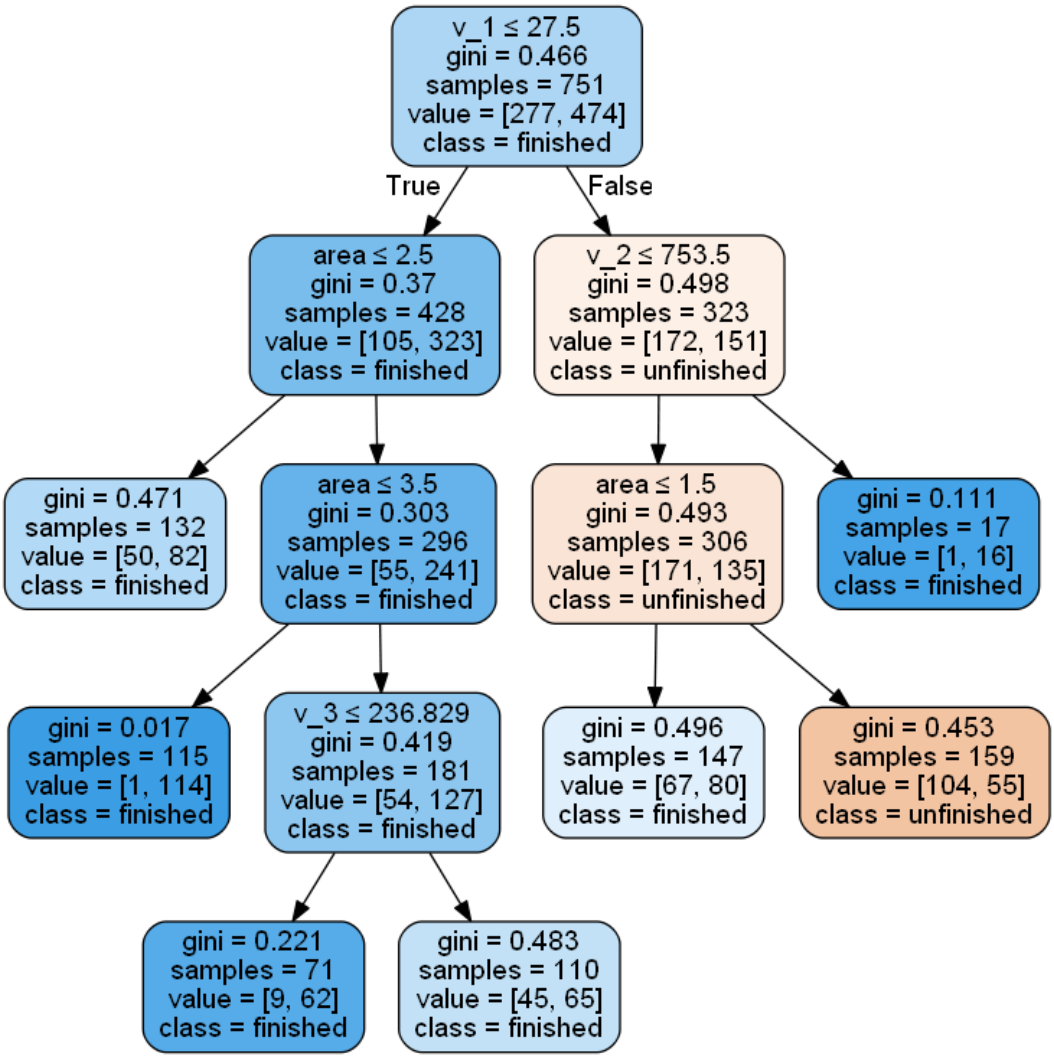

图 4: 随机森林结果

上图展示了随机森林中其中一棵决策树。由图可知，决策树从根部开始，根据 gini 系数选择合适的参数来分类，最终决定任务是否能完成。后续需要对新的定价策略进行评价时，只需将其参数传入该决策树，便可得到最终结果。

2.2 任务未完成原因分析

通过经纬度坐标，绘制出任务是否完成的位置图如下：

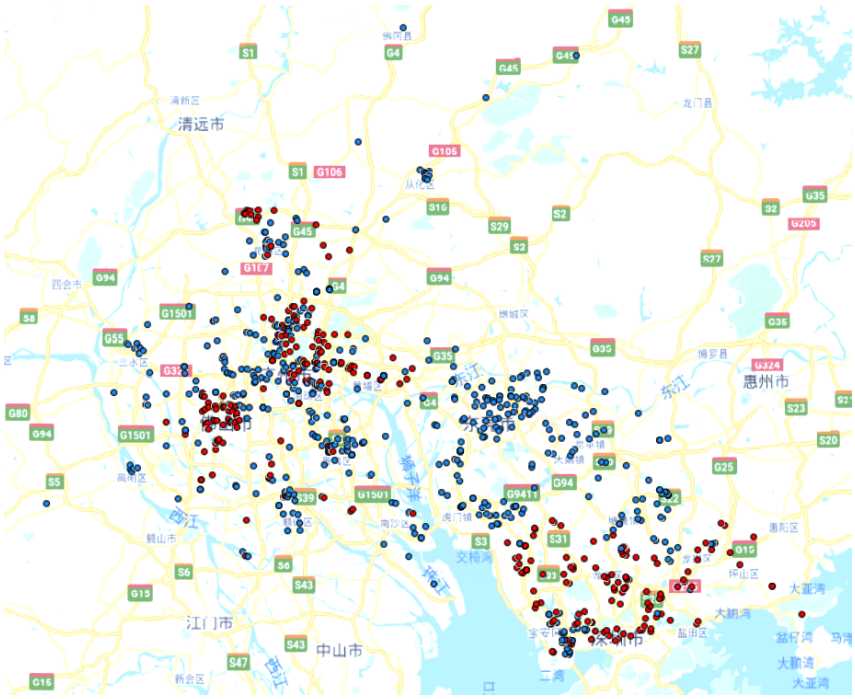

图 5: 任务完成情况蓝：完成红：未完成

通过经纬度坐标，绘制出会员分布的位置图如下：

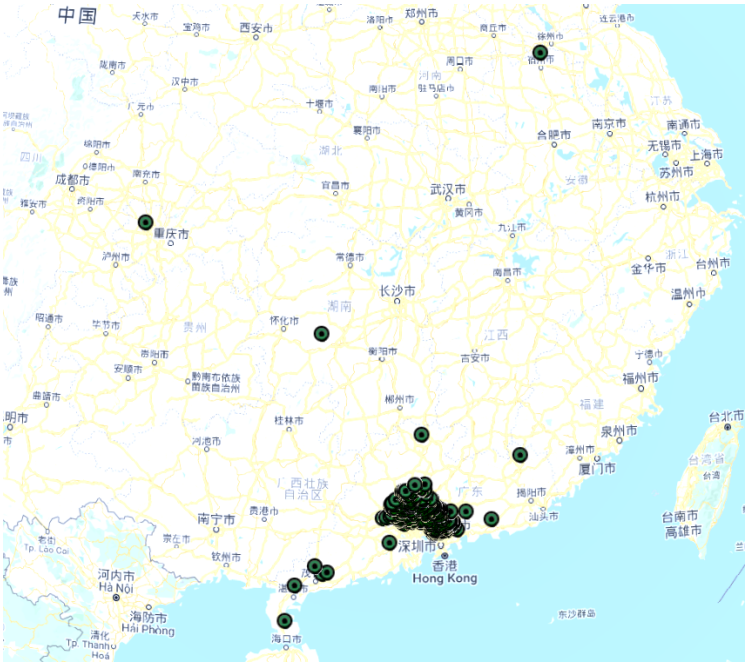

图 6: 会员分布情况绿：会员

可以看出，个别会员的位置远离研究主要区域，即广州市、佛山市、东莞市和深圳市。由此，可以去除掉远离研究区域的点，以及在画图过程中发现的编号为 B1175 的错误会员信息（经纬度坐标与其余左边相差过大，应为经纬度坐标写反），总共 14 个会员信息后，再次描绘会员的分布位置图如下：

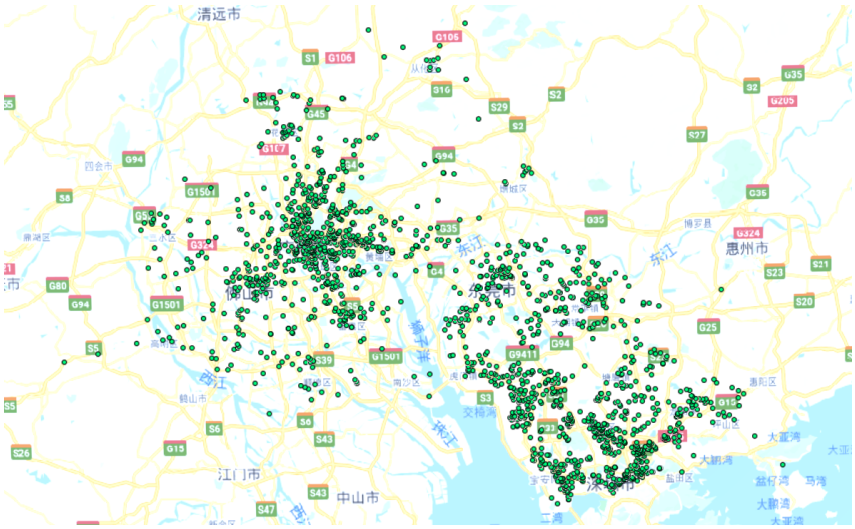

图 7: 筛选后会员分布情况绿：会员

可以较为清晰地看出位置分布与任务分布大体上呈现相同分布规律。

分析原因如下：

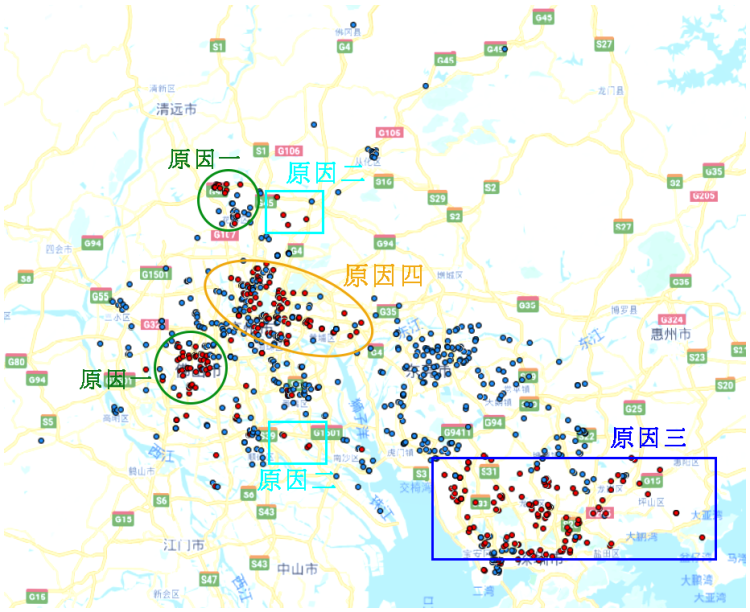

图 8: 任务未完成原因蓝：完成红：未完成

原因一：某些任务邻近区域内，任务密集，而会员人数远少于任务数，人均任务量远远大于其实际完成能力。

原因二：任务本身定价较低，使会员完成该任务的收益达不到期望值。某些任务远离会员密集地，若会员要完成需花费较高的费用，且任务定价较低，使得回报低于成本，故出现无人接单的情况。

原因三：高收入地区会员对收益要求较高，导致很多任务没人接单。例如深圳地区居民普遍收入较高，对收益也有较高的期望值，使得在其他地区能够完成的任务产生的

收益值不足以吸引他们去完成。

原因四：高信誉会员的限额较高，预约过多任务，导致很多任务没及时完成。某些任务周围有较多低信誉会员，但此任务被高信誉会员优先预约，而高信誉会员由于多任务在身或此任务距离较远而没有完成。

3 问题二的建模求解

在问题二中，我们可以发现对于原有的任务定价方案，完成率较低，只可达到62.51%。为此，我们需要设计新的定价策略：对于那些未完成的任务，可以适当提高其定价以吸引会员完成；对于已完成的任务，可以适当降低其定价来节约成本。

3.1 参数的选择

参考问题一的判断模型的参数，我们依然采用会员系数 v_1, v_2, v_3 以及任务点所在的区域作为模型的参数。

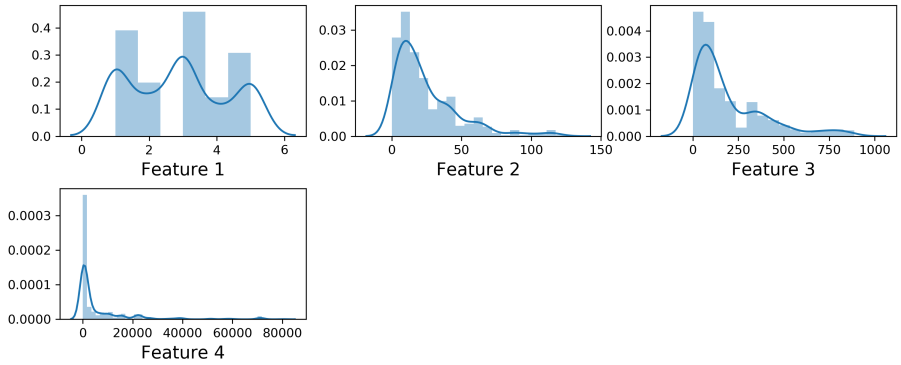

图 9: 参数的直观理解

从上图可以看到，各个参数的分布均呈现近似正态分布，我们对这些参数进行标准化处理，使其均值为 0，方差为 1，便于后续模型的训练。

3.2 模型的建立

考虑到这是一个回归问题，我们采用多项式回归对已完成任务的数据进行拟合，以此得到拟合函数，从而得到新的定价策略。

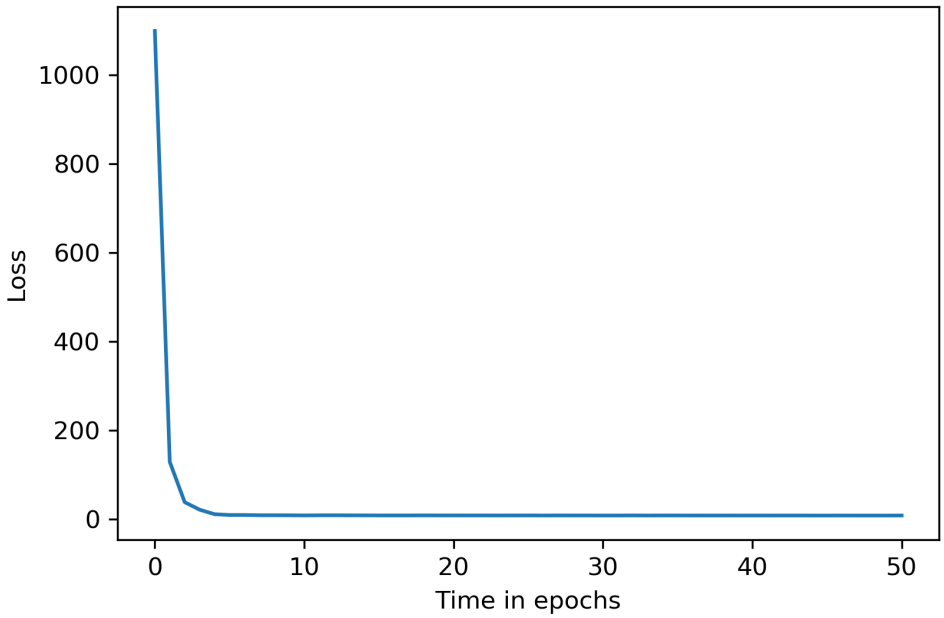

图 10: 多项式回归训练过程

在训练过程中，数据集依然以 4:1 的比例划分为训练集和测试集，多项式的阶数为 4。为了避免过拟合，在损失函数中加入 L2 正则。最终在训练集上的准确率达到 84.95%，在测试集上的准确率为 80.14%（允许正负值为 2）。

3.3 新定价策略的评估

在建立新的定价策略模型后，我们需要对该模型进行评估。这里我们使用的是在第一问中已建立的随机森林模型，来判断某一任务是否可以被完成。评估结果如下表所示：

表 1: 新定价方案

	原方案	新方案
完成率	62.51%	79.28%
成本 (元)	57561	57707

由表可知，我们的模型对原策略进行了优化，不仅将任务完成率提高至 79.28%，同时还将任务总定价降低了近 200 元，效果十分不错。

4 问题三的建模求解

4.1 打包的原则

本问题需要我们将多个任务联合在一起发布，这样既减少了会员之间竞争的程度，也降低了公司的成本。在打包的过程中我们要遵循几个原则：

（1）打包后的定价要大于打包之前每个的定价，同时要小于各任务点的定价总和，设 P 为打包后的价格，则 $P_i < P < \sum P_i$

（2）打包的任务点之间的距离要尽可能的小，同时会员到打包的所有任务点的距离不宜超过 5km. 即 $\min\ d(i,j)$ 而且，$d(i,k) < 5km$

（3）打包时尽量使没有完成的任务与完成的打包。通过没有完成的任务带动完成的任务，从而提高任务的完成率。

（4）打包的总的任务数不宜过大，不能超过会员的限额。

4.2 打包的算法以及结果

最终我们选用了以距离为度量的聚类算法，所打包的任务之间的距离首先不可超过2km，其次一个聚类中的点数不宜过多，我们选择了 9 个，即 $S \leq 9$，因为聚类中点数过多时会导致任务量太大从而使会员不愿意接单。最终经 python 处理后结果如下：

0	A0001	A0029	A0362	A0444						
1	A0002	A0364	A0446							
2	A0003	A0008	A0360							
3	A0004	A0006								
4	A0005	A0028	A0033							
6	A0007	A0013	A0019							
8	A0009	A0018								
9	A0010	A0016	A0124							
10	A0011	A0012	A0017	A0022	A0023	A0027	A0032	A0036	A0356	A0462
11	A0014									
13	A0015	A0020	A0021	A0026						
23	A0024	A0369	A0449							
24	A0025									
29	A0030	A0034	A0466							
30	A0031	A0035								

图 11: 打包后的分布情况

5 问题四的建模求解

5.1 打包之后定价模型的求解

在对打包问题的定价时，我们首先要遵循第三问所假设的原则，同时，将附件三中的新数据引用，运用整体整合的方法，我们把打包后的任务看成一个整体，选取打包点的中心点为距离. 即

$$s_j = \frac{\sum_k s_{jk}}{k} \tag{2}$$

然后我们继续沿用前两问的求解方式，将打包后的任务看成一个整体，分析整体所对应的会员数，会员限额以及会员信誉度。根据任务完成的难易情况，利用 SVR 进行模型拟合，最终推测出新的定价方式。然后将得到的模型运用到附件三中的新数据上进行求解。

5.2 最终定价情况下的成本以及完成率

最终经拟合以及检测的结果如下：

5.3 模型的结果分析

可以看到，打包后与打包前，任务的完成率并没有显著的变化，可能是因为大部分没完成任务的地点较为偏僻，无法进行打包，同时也就无法起到将其归入一些完成任务

表 2: 打包后的效果

	原方案	新方案
完成率	71.26%	73.81%
成本 (元)	132504	11142

中的情况，另一部分原因可能是当地会员数较少或者限额较低，有一些打包后的任务超出了当地会员的限额，导致了大的任务无法认领，无人认领的情况。

而打包后的成本下降较为明显，给企业减轻了压力，同时也减少了会员间的竞争。

6 模型的评价与推广

6.1 模型的优点

该模型充分利用了各种机器学习模型，以严谨的数学模型对问题进行建模，并且在参数的选择上做了许多思考，使之更加符合实际情况。最终的结果可解释性、可信度较强。

6.2 模型的缺点

该模型最大的缺点在于数据量过少，不足以支撑一些模型的精确训练，可能会导致结果的随机性较大，鲁棒性不够优异。

6.3 模型的推广

如果可以对市场进行进一步的调研与资料收集，构建一个更大的数据集用于训练，那么结果的准确率和参考价值可以得到进一步的提高。考虑到当前网络众包的快速发展，我们的模型可以为相关公司提供优化算法，在提高效率的同时降低成本。

附录

问题一：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

#导入任务地理位置数据
path='./Problem/附件一：已结束项目任务数据.xls'
origin=pd.read_excel(path)
data=origin.iloc[:,1:3]
data.head(5)

#将地理信息存入列表
location=[]
for index in data.index:
  location.append(data.loc[index].tolist())
```

```
#使用K-Means算法聚类
SSE=[]
axis=[i for i in range(1,10)]
for n in range(1,10):
  kmeans=KMeans(n_clusters=n, random_state=0).fit(location)
  SSE.append(kmeans.inertia_)
plt.plot(axis,SSE)
plt.xlabel('K')
plt.ylabel('SSE')
plt.savefig('./figure/k_select.png', dpi=300)
plt.show()

#画出聚类分布图
kmeans=KMeans(n_clusters=5, random_state=0).fit(location)
predict=kmeans.labels_
location=np.array(location)
plt.scatter(location[:,1], location[:,0], c=predict)
plt.savefig('./figure/areas.png', dpi=300)
plt.show()

#保存数据
column=['area']
result=pd.DataFrame(columns=column, data=predict+1)
result.to_excel('任务区域.xls')

import numpy as np
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
import random

#导入数据
area=pd.read_excel('任务区域.xls').iloc[:,1].to_list()
vip_1=pd.read_excel('会员系数详细.xlsx').iloc[:,2].to_list()
vip_2=pd.read_excel('会员系数详细.xlsx').iloc[:,3].to_list()
vip_3=pd.read_excel('会员系数详细.xlsx').iloc[:,4].to_list()
price=pd.read_excel('./Problem/附件一：已结束项目任务数据.xls').iloc[:,3].to_list()
label=pd.read_excel('./Problem/附件一：已结束项目任务数据.xls').iloc[:,4].to_list()

data=[]
for i in range(0,len(area)):
    carry=[]
    carry.append(area[i])
    carry.append(vip_1[i])
    carry.append(vip_2[i])
    carry.append(vip_3[i])
    carry.append(price[i])
    carry.append(label[i])
    data.append(carry)

#将样本顺序打乱
random.shuffle(data)

#划分训练集、测试集
bench=int(0.99*len(data))
data=pd.DataFrame(data)
```

```
X_train=np.array(data.iloc[:bench,:5])
X_test=np.array(data.iloc[bench:,:5])
y_train=data.iloc[:bench,5]
y_test=data.iloc[bench:,5]

from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_jobs=-1, criterion='gini', n_estimators=1000)
clf.fit(X_train,y_train)
# 测试模型
print(clf.score(X_train,y_train))
print(clf.score(X_test,y_test))

#对新的定价策略进行分析
area=pd.read_excel('任务区域.xls').iloc[:,1].to_list()
vip_1=pd.read_excel('会员系数详细.xlsx').iloc[:,2].to_list()
vip_2=pd.read_excel('会员系数详细.xlsx').iloc[:,3].to_list()
vip_3=pd.read_excel('会员系数详细.xlsx').iloc[:,4].to_list()
price=pd.read_excel('新定价策略.xls').iloc[:,1].to_list()

data_new=[]
for i in range(0,len(area)):
    carry=[]
    carry.append(area[i])
    carry.append(vip_1[i])
    carry.append(vip_2[i])
    carry.append(vip_3[i])
    carry.append(price[i])
    data_new.append(carry)

data_new=pd.DataFrame(data_new)
y=clf.predict(data_new)

amount=len(data)
print('新定价策略的完成率为',sum(y)/amount)
print('旧定价策略的完成率为',sum(data.iloc[:,5])/amount)
print('新定价策略的总价为',sum(price))
print('旧定价策略的总价为',sum(X_train[:,4])+sum(X_test[:,4]))
```

6.4 问题二：

```
import numpy as np
import pandas as pd
from sklearn.utils import shuffle
from sklearn import linear_model
from sklearn import preprocessing
import matplotlib.pyplot as plt
import sys
from io import StringIO
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn import svm

#导入数据
area=pd.read_excel('任务区域.xls').iloc[:,1].to_list()
vip_1=pd.read_excel('会员系数详细.xlsx').iloc[:,2].to_list()
vip_2=pd.read_excel('会员系数详细.xlsx').iloc[:,3].to_list()
vip_3=pd.read_excel('会员系数详细.xlsx').iloc[:,4].to_list()
price=pd.read_excel('./Problem/附件一：已结束项目任务数据.xls').iloc[:,3].to_list()
```

```
label=pd.read_excel('./Problem/附件一：已结束项目任务数据.xls').iloc[:,4].to_list()

data_finish=[]
data_unfinish=[]
for i in range(0,len(area)):
    carry=[]
    carry.append(area[i])
    carry.append(vip_1[i])
    carry.append(vip_2[i])
    carry.append(vip_3[i])
    carry.append(price[i])

    if label[i]==1:
        data_finish.append(carry)
    else:
        data_unfinish.append(carry)
data_finish=pd.DataFrame(data_finish)
data_unfinish=pd.DataFrame(data_unfinish)

#将样本顺序打乱
data_finish=shuffle(data_finish)

X=data_finish.iloc[:,:4]
y=data_finish.iloc[:,4]

#准备多项式特征
poly=preprocessing.PolynomialFeatures(4)
X=poly.fit_transform(X)
scaler = StandardScaler()
X=pd.DataFrame(scaler.fit_transform(X))

#划分训练集、测试集
bench=int(0.85*len(data_finish))
X_train=X.iloc[:bench]
X_test=X.iloc[bench:]
y_train=y.iloc[:bench]
y_test=y.iloc[bench:]

def evaluate_model(model,X,y):

    y_pred=model.predict(X)
    count=0
    for i in range(len(y_pred)):
        if y_pred[i]<=y.iloc[i]+3 and y_pred[i]>=y.iloc[i]-3:
            count=count+1
    return count/len(y_pred),metrics.mean_squared_error(y,y_pred)

# 模型训练
ridge_reg=linear_model.Ridge()
ridge_reg.fit(X_train,y_train)

# 训练集准确率和均方差
accuracy_rate_pr_train, mse_pr_train = evaluate_model(ridge_reg, X_train, y_train)
print('模型训练准确率为:', accuracy_rate_pr_train)
print('模型训练均方误差为:', mse_pr_train)

# 测试集准确率和均方差
accuracy_rate_pr_test, mse_pr_test = evaluate_model(ridge_reg, X_test, y_test)
print('模型测试准确率为:', accuracy_rate_pr_test)
```

```
print('模型测试均方误差为:', mse_pr_test)

ridge_reg.coef_
```

问题三：

```
from geopy.distance import geodesic
from sklearn import preprocessing
import pandas as pd
import csv

data=pd.read_excel('E:/大二下/数模/2020培训练习3/会员信息数据.xlsx')
data2=pd.read_excel('E:/大二下/数模/2020培训练习3/已结束项目任务数据.xls')
print(data)

print(data['会员编号'])

#将经纬度换成距离
def distance(x,y):
  return geodesic(x,y).km

def judge(member,seat):
  str_list=str(member).split()
  tup=(str_list[0],str_list[1])
  dist=distance(tup,seat)
  return dist


seat_list=[]
new=[[]for i in range(data2.shape[0])]
result_list=[]
for i in range(data2.shape[0]):
data_se1=(data2.iloc[i,1],data2.iloc[i,2])
k=0

if data2.iloc[i,0] not in result_list:
  new[i].append(data2.iloc[i,0])

result_list.append(data2.iloc[i,0])

for j in range(data2.shape[0]):
  data_se2=(data2.iloc[j,1],data2.iloc[j,2])
  result=distance(data_se1,data_se2)
if k<9 and (data2.iloc[j,0] not in result_list) and result<2:
  result_list.append(data2.iloc[j,0])
  new[i].append(data2.iloc[j,0])
  k=k+1

new_df=pd.DataFrame(new)
new_df.to_excel("new.xlsx")
```

问题四：

```
data_list=[]
for i in range(data2.shape[0]):
sum_limit=0#会员限额
sum_credit=0#会员信誉值
```

```
sum_number=0#会员数量

a=0.3
b=0.02
c=0.0001
sclaer=preprocessing.MinMaxScaler()

for j in range(data.shape[0]):
  data_seat=(data2.iloc[i,1],data2['任务gps经度'][i])
  result=judge(data.iloc[j,1],data_seat)
  if result<5:
  sum_number+=1
  sum_limit+=data.iloc[j,2]
  sum_credit+=data.iloc[j,4]

xishu=a*sum_number+b*sum_limit+c*sum_credit
#xishu=a*sclaer.fit_transform(sum_number)+b*sclaer.fit_transform(sum_limit)
#+c*sclaer.fit_transform(sum_credit)

#print(data2.iloc[i,0],xishu)
data_list.append([data2.iloc[i,0],xishu,sum_number,sum_limit,sum_credit])

df=pd.DataFrame(data_list,columns=['任务号码','会员系数','会员数','会员限额','会员信誉度
                                   '])
df.to_excel("会员系数2.xlsx",index=False)

# 模型训练
ridge_reg=linear_model.Ridge()
ridge_reg.fit(X_train,y_train)

# 训练集准确率和均方差
accuracy_train, mse_pr_train = evaluate_model(ridge_reg, X_train, y_train)
print('模型训练准确率为:', accuracy__train)
print('模型训练均方误差为:', mse_pr_train)

# 测试集准确率和均方差
accuracy_rate_pr_test, mse_pr_test = evaluate_model(ridge_reg, X_test, y_test)
print('模型测试准确率为:', accuracy_rate_pr_test)
print('模型测试均方误差为:', mse_pr_test)
```