深圳大学考试答题纸

(以论文、报告等形式考核专用)

二○二三～二○二四学年度第一学期

课程编号	1501990016	课程名称	数据科学导论	主讲教师	章秦	评分	
学　　号	2022150130	姓名	林宪亮	专业年级	22 级计算机科学与技术		

教师评语：

项目名称：o2o 优惠券使用预测

摘　要：

[简单列出已经实现的功能]

1. 赛题分析
2. 数据预处理与可视化分析
3. 特征工程
4. xgb 算法模型搭建与调优
5. 其它算法的模型搭建尝试
6. 算法分析与改进思路
7. 总结

深圳大学课程项目报告

课程名称：数据科学导论

项目名称：o2o 优惠券使用预测

学　　院：计算机与软件学院

专　　业：计算机科学与技术

任课教师：章秦

报 告 人：林宪亮 、吴嘉楷、朱粤锋

学号：2022150130、2022150168、2022280374

提交时间：2024 年 1 月 1 日

教 务 处 制

一、 项目要求

以二人一组为单位，在所列五个项目中选择一项，进行数据分析、特征处理、模型构建、模型评估和模型优化：

1）新浪微博互动预测
https://tianchi.aliyun.com/competition/entrance/231574/introduction?spm=5176 .12281973.1005.10.3dd53eafkznLFT

2） 保险反欺诈预测
https://tianchi.aliyun.com/competition/entrance/531994/information

3） 心跳信号分类预测
https://tianchi.aliyun.com/competition/entrance/531883/introduction?spm=5176.12282013.0.0.2837c123qWM8jg

4） o2o 优惠券使用预测
https://tianchi.aliyun.com/competition/entrance/231593/introduction?spm=5176.12281973.1005.69.3dd53eafkznLFT

5） 天猫复购预测
https://tianchi.aliyun.com/competition/entrance/231576/information

以报告的形式展示项目的实施方案、具体流程和详细结论。代码以单独文件进行提交。

具体任务：

1. 数据分析模块：
   （1） 数据总体了解：读取数据并了解数据集大小，原始特征维度，粗略查看数据集中各特征基本统计量。
   （2） 缺失值了解：样本特征缺失值情况，特征属性缺失值情况。
   （3） 不同类别特征的分析：连续性特征、离散型特征。连续性特征的可视化展示。离散特征的基本信息。
   （4） 数据间相关关系：特征和目标变量之间的关系，根绝 y 值不同可视化 x 某个特征的分布。
2. 特征工程模块：
   （1）数据预处理：缺失值的填充；异常值检测；
   （2）特征交互：特征和特征之间组合；特征和特征之间衍生；
   （3）特征预处理：特征归一化；
   （4）特征选择：方差选择法；相关系数法；互信息法。
3. 建模与调参
   （1） 构建线性回归/分类/聚类模型
   （2） 构建非线性回归/分类/聚类模型
   （3） 模型对比与评估：划分数据集，构建评价指标，模型调优。

报告要求：

（1） 内容完整，模块清晰。
（2） 方法自选，逻辑合理。

（3） 结果符合预期。
（4） 报告格式规范，阐述清晰。
（5） 代码简洁、可读性强。

总分（100 分）：

评分标准如下：

1) 给出项目任务，数据以及目标函数的描述。**（15 分）**

2）对数据实施基本的数据统计和可视化分析**（20 分）**

3）写出比赛的思路，包括数据清洗，特征构造，模型选择（聚类，分类或者回归等），需要指出自己完成的部分。**（30 分）**

4）算法优缺点分析，根据比赛结果，给出优化方案的思考。**（20 分）**

5）汇报比赛中用到的方法，最终比赛排名，以及体会。**（15 分）**

注意事项：

1. **项目设计和项目报告由二人一组完成。如存在抄袭行为，抄袭者和被抄袭者考试成绩均为零分。**
2. **截止时间：2024 年 1 月 2 日 23：59。报告题目请以“学号_姓名_期末大作业.pdf”命名。**
3. **在截止时间之前需要提交的内容：**
   a) **提交本《项目报告》电子版；2024 年 1 月 2 日 23：59 前提交到 blackboard 系统。**
   b) **提交本《项目报告》打印版；2024 年 1 月 3 日上课时交给老师。（第一页《答题纸》单面打印，其他内容双面打印）。**
   c) **准备汇报 PPT，第十六至十八周课堂汇报（电子版提交 BB 系统，无需提交打印版）。**
   d) **逾期提交无成绩（以 blackboard 系统的电子版提交时间为准）**

二、项目知识点

1. 数据预处理；
2. 数据可视化；
3. 特征工程；
4. 模型选择；
5. 实验结果的评价；

三、实验过程

1. 赛题分析

我们选取的赛题是“o2o 优惠券使用预测”
https://tianchi.aliyun.com/competition/entrance/231593/introduction?spm=5176.12281973.1005.69.3dd53eafkznLFT

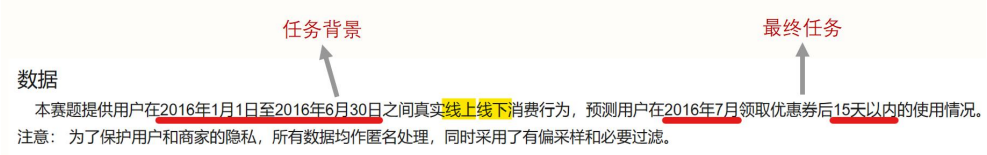
1.1 比赛背景：

随着移动设备的完善和普及，移动互联网+各行各业进入了高速发展阶段，这其中以 O2O（Online to Offline）消费最为吸引眼球。据不完全统计，O2O 行业估值上亿的创业公司至少有 10 家，也不乏百亿巨头的身影。O2O 行业天然关联数亿消费者，各类 APP 每天记录了超过百亿条用户行为和位置记录，因而成为大数据科研和商业化运营的最佳结合点之一。 以优惠券盘活老用户或吸引新客户进店消费是 O2O 的一种重要营销方式。然而随机投放的优惠券对多数用户造成无意义的干扰。对商家而言，滥发的优惠券可能降低品牌声誉，同时难以估算营销成本。 个性化投放是提高优惠券核销率的重要技术，它可以让具有一定偏好的消费者得到真正的实惠，同时赋予商家更强的营销能力。本次大赛为参赛选手提供了 O2O 场景相关的丰富数据，希望参赛选手通过分析建模，精准预测用户是否会在规定时间内使用相应优惠券。

1.2 评价方式：

使用优惠券核销预测的平均 AUC（ROC 曲线下面积）作为评价标准。 即对每个优惠券 coupon_id 单独计算核销预测的 AUC 值，再对所有优惠券的 AUC 值求平均作为最终的评价标准。

1.3 提供的数据集：



线下消费数据集：

Table 1: 用户线下消费和优惠券领取行为

Field	Description
User_id	用户ID
Merchant_id	商户ID
Coupon_id	优惠券ID：null表示无优惠券消费，此时Discount_rate和Date_received字段无意义
Discount_rate	优惠率：x \in [0,1]代表折扣率；x:y表示满x减y。单位是元
Distance	user经常活动的地点离该merchant的最近门店距离是x*500米（如果是连锁店，则取最近的一家门店），x\in[0,10]；null表示无此信息，0表示低于500米，10表示大于5公里；
Date_received	领取优惠券日期
Date	消费日期：如果Date=null & Coupon_id != null，该记录表示领取优惠券但没有使用，即负样本；如果Date!=null & Coupon_id = null，则表示普通消费日期；如果Date!=null & Coupon_id != null，则表示用优惠券消费日期，即正样本；

表格中的信息为对线下消费数据集特征的解释：

User_id	Merchant_i	Coupon_id	Discount_r	Distance	Date_rece	Date
1439408	2632	null	null	0	null	20160217
1439408	4663	11002	150:20:00	1	20160528	null
1439408	2632	8591	20:01	0	20160217	null
1439408	2632	1078	20:01	0	20160319	null
1439408	2632	8591	20:01	0	20160613	null
1439408	2632	null	null	0	null	20160516
1439408	2632	8591	20:01	0	20160516	20160613
1832624	3381	7610	200:20:00	0	20160429	null
2029232	3381	11951	200:20:00	1	20160129	null
2029232	450	1532	30:05:00	0	20160530	null
2029232	6459	12737	20:01	0	20160519	null
2029232	6459	null	null	0	null	20160626
2029232	6459	null	null	0	null	20160519
2747744	6901	1097	50:10:00	null	20160606	null

上面表格是提供的线下消费数据集的部分展示，共有七个特征。

线上消费数据集：

Table 2: 用户线上点击/消费和优惠券领取行为

Field	Description
User_id	用户ID
Merchant_id	商户ID
Action	0 点击， 1购买，2领取优惠券
Coupon_id	优惠券ID：null表示无优惠券消费，此时Discount_rate和Date_received字段无意义。“fixed”表示该交易是限时低价活动。
Discount_rate	优惠率：x \in [0,1]代表折扣率；x:y表示满x减y；“fixed”表示低价限时优惠；
Date_received	领取优惠券日期
Date	消费日期：如果Date=null & Coupon_id != null，该记录表示领取优惠券但没有使用；如果Date!=null & Coupon_id = null，则表示普通消费日期；如果Date!=null & Coupon_id != null，则表示用优惠券消费日期；

表格中的信息为对线上消费数据集特征的解释：

User_id	Merchant_i	Action	Coupon_id	Discount_r	Date_rece	Date
13740231	18907	2	100017492	500:50:00	20160513	null
13740231	34805	1	null	null	null	20160321
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	18907	0	null	null	null	20160618
14336199	38810	0	null	null	null	20160126
14336199	38810	0	null	null	null	20160126
14336199	38810	0	null	null	null	20160126
14336199	38810	0	null	null	null	20160126

上面表格是提供的线上消费数据集的部分展示，共有七个特征。

预测数据集：

Table 3：用户O2O线下优惠券使用预测样本

Field	Description
User_id	用户ID
Merchant_id	商户ID
Coupon_id	优惠券ID
Discount_rate	优惠率：x \in [0,1]代表折扣率；x:y表示满x减y.
Distance	user经常活动的地点离该merchant的最近门店距离是x*500米（如果是连锁店，则取最近的一家门店），x\in[0,10]；null表示无此信息，0表示低于500米，10表示大于5公里；
Date_received	领取优惠券日期

User_id	Merchant_i	Coupon_id	Discount_r	Distance	Date_received
4129537	450	9983	30:05:00	1	20160712
6949378	1300	3429	30:05:00	null	20160706
2166529	7113	6928	200:20:00	5	20160727
2166529	7113	1808	100:10:00	5	20160727
6172162	7605	6500	30:01:00	2	20160708
4005121	450	9983	30:05:00	0	20160706
4347394	450	9983	30:05:00	0	20160716
3094273	760	13602	30:05:00	1	20160727
5139970	450	9983	30:05:00	10	20160729
3237121	760	13602	30:05:00	1	20160703
6224386	450	9983	30:05:00	3	20160716
6488578	760	13602	30:05:00	0	20160712
4164865	450	9983	30:05:00	2	20160703
4164865	5138	8059	50:10:00	1	20160706

1.4 需要提交的文件格式：

Table 4：选手提交文件字段，其中user_id,coupon_id和date_received均来自Table 3,而Probability为预测值

Field	Description
User_id	用户ID
Coupon_id	优惠券ID
Date_received	领取优惠券日期
Probability	15天内用券概率，由参赛选手给出

所需要通过模型预测得到的数据

1.5 解决思路：

通过上面对于赛题以及数据集的初步了解与分析，我们制定了大致的解决思路。

- 对数据集进行探索与可视化分析，初步探索特征间的关系。
- 对数据集进行预处理，包括缺失值的填充，异常值的处理。
- 特征工程，生成一些对模型训练模型预测有用的特征/
- 选用合适的算法训练模型。
- 进行预测和模型调优。

2. 数据分析:

2.1 首先，读入数据并通过 head 函数查看数据集的前几行信息：

```
def get_source_data():

    # 源数据路径
    DataPath = ''

    # 读入源数据
    off_train = pd.read_csv(os.path.join(DataPath, 'ccf_offline_stage1_train.csv'),
                            parse_dates=['Date_received', 'Date'])
    off_train.columns = ['User_id', 'Merchant_id', 'Coupon_id', 'Discount_rate',
'Distance', 'Date_received', 'Date']

    on_train = pd.read_csv(os.path.join(DataPath, 'ccf_online_stage1_train.csv'),
parse_dates=['Date_received', 'Date'])
    on_train.columns = ['User_id', 'Merchant_id', 'Action', 'Coupon_id',
'Discount_rate', 'Date_received', 'Date']

    off_test = pd.read_csv(os.path.join(DataPath,
'ccf_offline_stage1_test_revised.csv'), parse_dates=['Date_received'])
    off_test.columns = ['User_id', 'Merchant_id', 'Coupon_id', 'Discount_rate',
'Distance', 'Date_received']

    print("off_train:")
    print(off_train.head())
    print("on_train:")
```

```
    print(on_train.head())
    print("off_test:")
    print(off_test.head())

    return off_train,on_train,off_test
```

输出:

```
off_train:
   User_id  Merchant_id  Coupon_id  ... Distance  Date_received       Date
0  1439408         2632        NaN  ...      0.0            NaT 2016-02-17
1  1439408         4663    11002.0  ...      1.0     2016-05-28        NaT
2  1439408         2632     8591.0  ...      0.0     2016-02-17        NaT
3  1439408         2632     1078.0  ...      0.0     2016-03-19        NaT
4  1439408         2632     8591.0  ...      0.0     2016-06-13        NaT
```

```
on_train:
    User_id  Merchant_id  Action  ... Discount_rate Date_received       Date
0  13740231        18907       2  ...        500:50    2016-05-13        NaT
1  13740231        34805       1  ...           NaN           NaT 2016-03-21
2  14336199        18907       0  ...           NaN           NaT 2016-06-18
3  14336199        18907       0  ...           NaN           NaT 2016-06-18
4  14336199        18907       0  ...           NaN           NaT 2016-06-18
```

```
off_test
   User_id  Merchant_id  Coupon_id Discount_rate  Distance Date_received
0  4129537          450       9983          30:5       1.0    2016-07-12
1  6949378         1300       3429          30:5       NaN    2016-07-06
2  2166529         7113       6928        200:20       5.0    2016-07-27
3  2166529         7113       1808        100:10       5.0    2016-07-27
4  6172162         7605       6500          30:1       2.0    2016-07-08
```

2.2 接着是利用 info 函数，查看数据的基本信息

```
    print("off_train:")
    print(off_train.info())
    print("on_train:")
    print(on_train.info())
    print("off_test:")
    print(off_test.info())
```

输出:

```
off_train:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1754884 entries, 0 to 1754883
Data columns (total 7 columns):
 #   Column         Dtype
---  ------         -----
 0   User_id        int64
 1   Merchant_id    int64
 2   Coupon_id      float64
 3   Discount_rate  object
 4   Distance       float64
 5   Date_received  datetime64[ns]
 6   Date           datetime64[ns]
dtypes: datetime64[ns](2), float64(2), int64(2), object(1)
memory usage: 93.7+ MB
```

```
on_train:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11429826 entries, 0 to 11429825
Data columns (total 7 columns):
 #   Column         Dtype
---  ------         -----
 0   User_id        int64
 1   Merchant_id    int64
 2   Action         int64
 3   Coupon_id      object
 4   Discount_rate  object
 5   Date_received  datetime64[ns]
 6   Date           datetime64[ns]
dtypes: datetime64[ns](2), int64(3), object(2)
memory usage: 610.4+ MB
```

```
off_test:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113640 entries, 0 to 113639
Data columns (total 6 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   User_id        113640 non-null  int64
 1   Merchant_id    113640 non-null  int64
 2   Coupon_id      113640 non-null  int64
 3   Discount_rate  113640 non-null  object
 4   Distance       101576 non-null  float64
 5   Date_received  113640 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(3), object(1)
memory usage: 5.2+ MB
```

通过输出结果，我们可以了解到数据各个列的数据类型和数据的缺失情况，为后续特征工程的数据预处理工作作了充分的铺垫。

2.3 然后利用 describe 函数查看统计信息

```
print("off_train 训练集的统计信息")
print(off_train.describe())
print("on_train 训练集的统计信息")
print(on_train.describe())
print("off_test 训练集的统计信息")
print(off_test.describe())
```

输出:

```
off_train训练集的统计信息
           User_id   Merchant_id     Coupon_id      Distance
count  1.754884e+06  1.754884e+06  1.053282e+06  1.648881e+06
mean   3.689255e+06  4.038808e+03  6.815398e+03  2.361636e+00
std    2.123428e+06  2.435963e+03  4.174276e+03  3.483974e+00
min    4.000000e+00  1.000000e+00  1.000000e+00  0.000000e+00
25%    1.845052e+06  1.983000e+03  2.840000e+03  0.000000e+00
50%    3.694446e+06  3.532000e+03  7.430000e+03  0.000000e+00
75%    5.528759e+06  6.329000e+03  1.032300e+04  3.000000e+00
max    7.361032e+06  8.856000e+03  1.404500e+04  1.000000e+01
```

```
on_train训练集的统计信息
            User_id   Merchant_id        Action
count  1.142983e+07  1.142983e+07  1.142983e+07
mean   1.074683e+07  3.436686e+04  2.348193e-01
std    4.137712e+06  1.441243e+04  5.426314e-01
min    4.000000e+00  1.000100e+04  0.000000e+00
25%    1.019827e+07  2.090100e+04  0.000000e+00
50%    1.196970e+07  3.420000e+04  0.000000e+00
75%    1.373506e+07  4.741500e+04  0.000000e+00
max    1.550000e+07  6.000000e+04  2.000000e+00
```

```
off_test训练集的统计信息
           User_id    Merchant_id      Coupon_id       Distance
count  1.136400e+05  113640.000000  113640.000000  101576.000000
mean   3.684858e+06    2962.283853    9053.810929       2.328040
std    2.126259e+06    2494.450802    4145.873088       3.260755
min    2.090000e+02       6.000000       3.000000       0.000000
25%    1.844191e+06     760.000000    5023.000000       0.000000
50%    3.683266e+06    2050.000000    9983.000000       1.000000
75%    5.525845e+06    5138.000000   13602.000000       3.000000
max    7.361024e+06    8856.000000   14045.000000      10.000000
```

2.4 查看数据缺失情况

```
print("off_train:")
print(off_train.isnull().sum())
print("on_train:")
print(on_train.isnull().sum())
print("off_test:")
print(off_test.isnull().sum())
```

输出:

```
off_train:
User_id              0
Merchant_id          0
Coupon_id       701602
Discount_rate   701602
Distance        106003
Date_received   701602
Date            977900
dtype: int64
on_train:
User_id              0
Merchant_id          0
Action               0
Coupon_id        10557469
Discount_rate    10557469
Date_received    10557469
Date               655898
dtype: int64
off_test:
User_id              0
Merchant_id          0
Coupon_id            0
Discount_rate        0
Distance         12064
Date_received        0
dtype: int64
```

由输出结果可知优惠券 id，折扣率，领券日期，三者可能存在同时==null 的情况

2.5 然后做个简单统计，看一看究竟用户使用优惠券消费的情况。

```
# 有券未消费
coup_no_consume = off_train [(off_train ['Date'].isnull() & off_train
['Coupon_id'].notnull())]
# 无券未消费
no_coup_no_consume = off_train [(off_train ['Date'].isnull() & off_train
['Coupon_id'].isnull())]
# 无券消费
no_coup_consume = off_train [(off_train ['Date'].notnull() & off_train
['Coupon_id'].isnull())]
# 有券消费
coup_consume = off_train [(off_train ['Date'].notnull() & off_train
['Coupon_id'].notnull())]

print('有券未消费：{}'.format(len(coup_no_consume)))
print('无券未消费：{}'.format(len(no_coup_no_consume)))
print('无券消费：{}'.format(len(no_coup_consume)))
print('有券消费：{}'.format(len(coup_consume)))
```

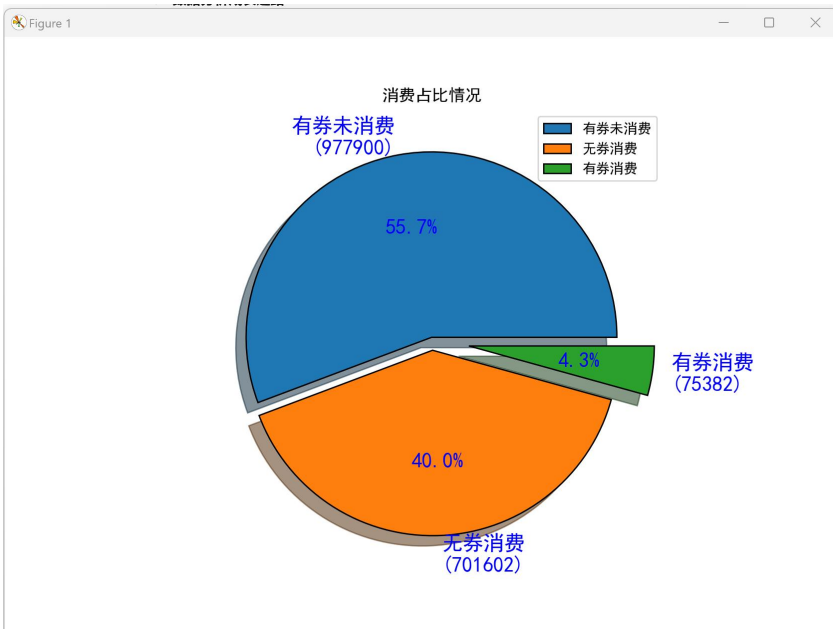
输出结果:

```
有券未消费：977900
无券未消费：0
无券消费：701602
有券消费：75382
```

并将结果可视化:

```
consume_status_dict = {'coup_no_consume': len(coup_no_consume),
                       'no_coup_consume': len(no_coup_consume),
                       'coup_consume': len(coup_consume)}
consume_status = pd.Series(consume_status_dict)
plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
# 绘制消费情况饼图
fig, ax=plt.subplots(1, 1, figsize=(8,10))
consume_status.plot.pie(ax=ax,
                        autopct='%1.1f%%',
                        shadow=True,
                        explode=[0.02,0.05,0.2],
                        textprops={'fontsize':15, 'color':'blue'},
                        wedgeprops={'linewidth': 1, 'edgecolor': 'black'},
                        labels=[
                            '有券未消费\n({})'.format(len(coup_no_consume)),
                            '无券消费\n({})'.format(len(no_coup_consume)),
                            '有券消费\n({})'.format(len(coup_consume))
                        ]
)
ax.set_ylabel('')
ax.set_title('消费占比情况')
plt.legend(labels=['有券未消费' , '无券消费' ,'有券消费'])
plt.show()
```

绘制饼图如下:



由图可知有券未消费占比 55.7%最大，说明大多数认拿完券尚未使用；无券消费用户占比 40%，说明很多人没有使用优惠券，可能优惠券吸引力不大，顾客不在意，也可

能新用户比较多；用券消费用户占比较小 4.3%，说明优惠券使用率不高，可以考虑加大优惠力度

综合消费情况可知，701602 人购买商品却没有使用优惠券，也有 977900 人有优惠券但却没有使用，真正使用优惠券购买商品的人（共 75382 人）很少。所以，这个比赛的意义就是把优惠券送给真正可能会购买商品的人。

2.6 在有券消费人群中，分析距离

```
# 各商家对应的顾客到店的平均距离
merchant_distance=coup_consume.groupby('Merchant_id')['Distance'].mean()
print(len(merchant_distance))
print(merchant_distance[merchant_distance==0])
```

打印结果如下:

```
4076
Merchant_id
3       0.0
4       0.0
13      0.0
14      0.0
18      0.0
       ...
8806    0.0
8824    0.0
8828    0.0
8849    0.0
8856    0.0
Name: Distance, Length: 1431, dtype: float64
```

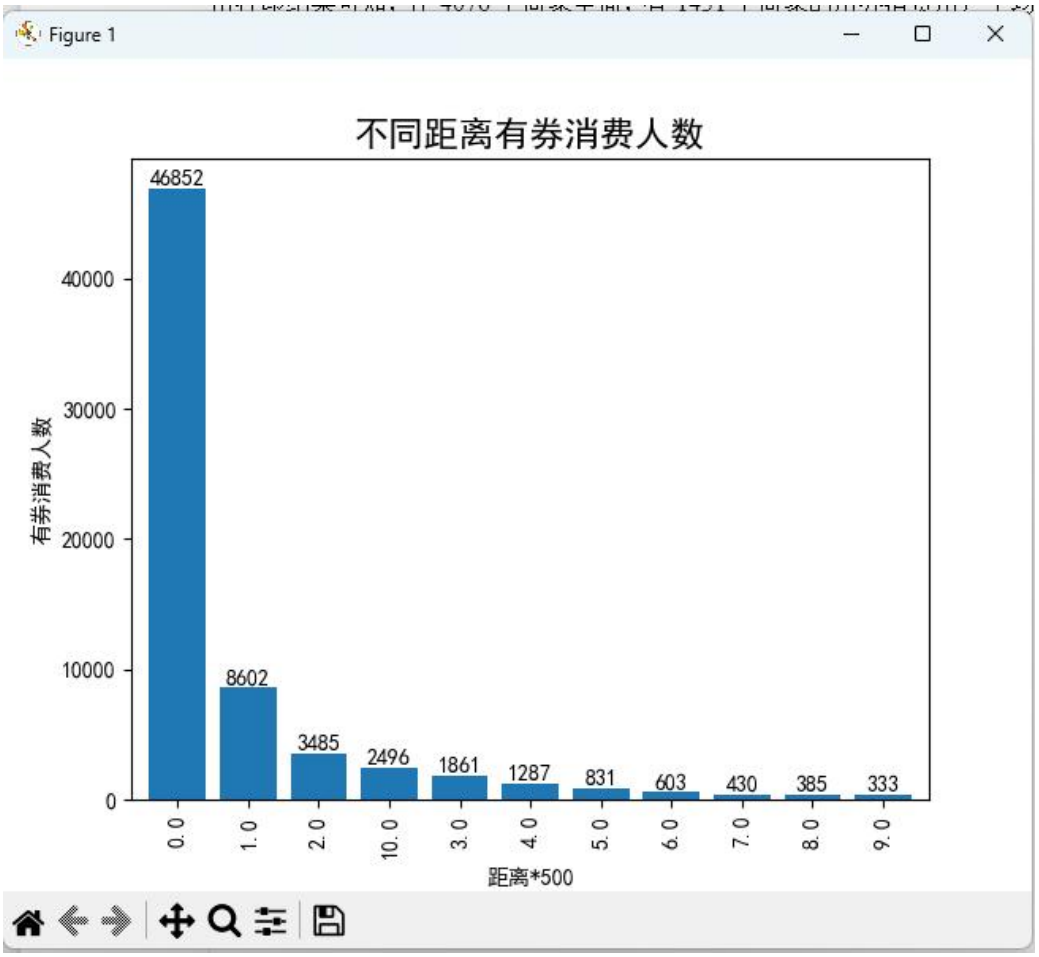
由打印结果可知，在 4076 个商家里面，有 1431 个商家的用券消费用户平均范围在 500 米以内，说明超过 1/4 的用券消费客户都是就近消费。

2.7 分析距离商家不同距离的有券消费人数

```
distances = coup_consume['Distance'].value_counts()
distances.plot(kind='bar', width=0.8)
plt.xlabel('距离*500')
plt.ylabel('有券消费人数')
plt.title('不同距离有券消费人数', fontsize=16, fontweight='bold')
for x, y in enumerate(distances):
    plt.text(x, y + 1, str(y), ha='center', va='bottom', fontsize=10)
plt.show()
```

从有券消费的用户的数据中获取不同距离下的有券消费人数进行统计，然后使用柱状图展示数据。

可视化结果如下:



距离一共分为 0-10 中不同距离，单位是 500 米，从图中可以看出大部分用券消费的用户消费范围是在 500 米内。

2.8　在有券消费人群中，分析优惠折扣

先填补缺失值:

```
off_train['Discount_rate']= off_train['Discount_rate'].fillna('null')
```

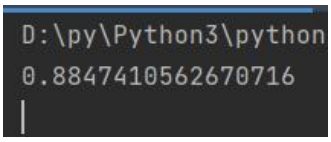
然后把存在 x:y 的折扣的形式转化为折扣:

```
def discount_rate_opt(s):
    if ':' in s:
        split = s.split(':')
        discount_rate = ((int(split[0])-int(split[1]))/int(split[0]))
        return round(discount_rate,2)
    elif s == 'null':
        return np.NAN
    else:
        return float(s)

off_train['Discount_rate']= off_train['Discount_rate'].map(discount_rate_opt)
```

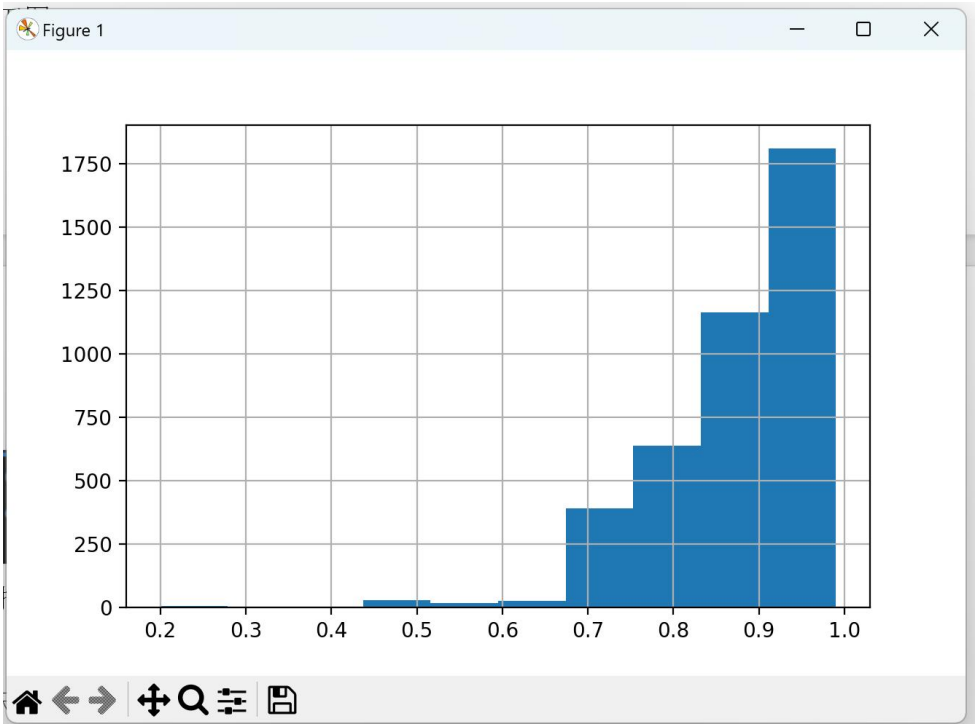
然后进行分析:

```
# 各商家对应的顾客到店消费平均折扣力度
merchant_discount_rate=coup_consume.groupby('Merchant_id')['Discount_rate'].mean()
merchant_discount_rate.sort_values()
print(merchant_discount_rate.mean())
merchant_discount_rate.hist()
plt.show()
```

打印结果如下图:



可知平均折扣大约为 0.88，然后可视化结果如下:



由图可以看出大多数商家的优惠折扣力度一般，很大程度上并不能吸引客户消费。

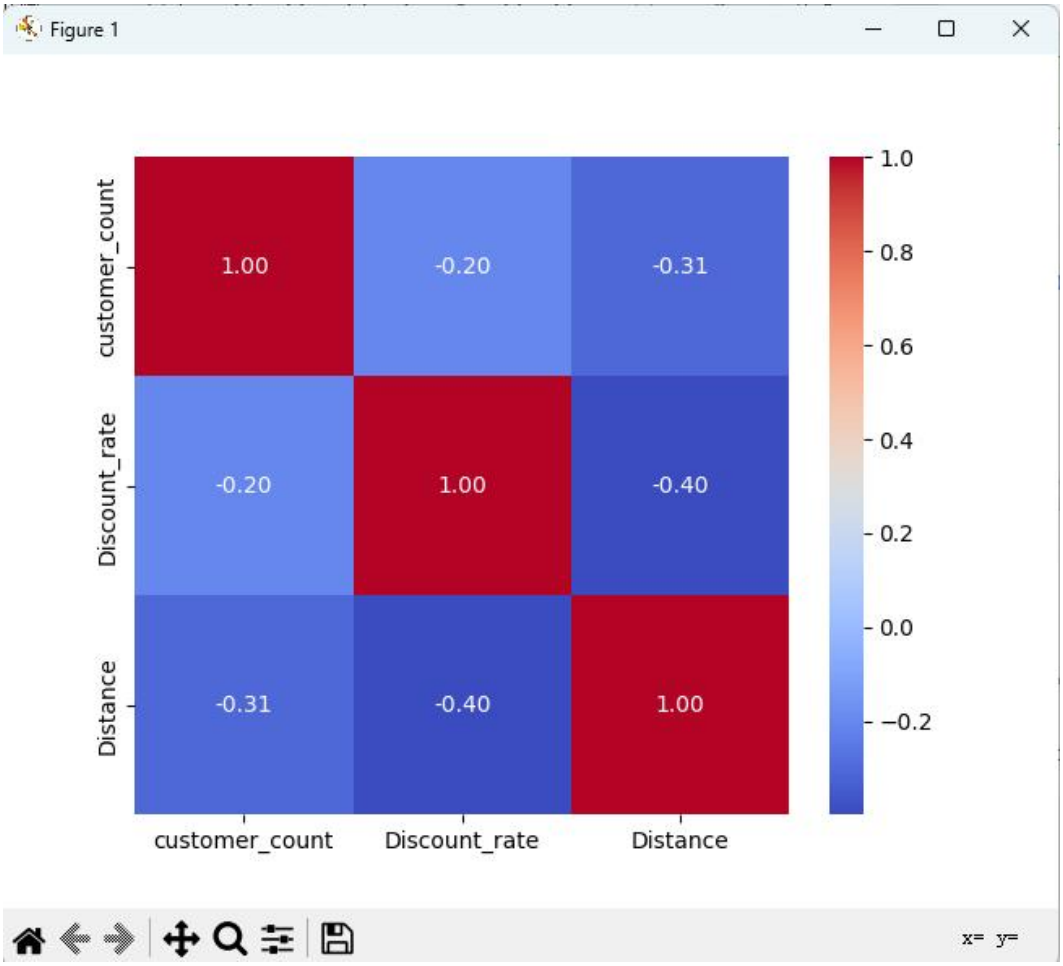
2.9 选取热门商家，计算到店消费人数与平均距离和折扣力度的相关系数

```
popular_merchant = coup_consume.groupby('Merchant_id')['User_id'].apply(lambda x:
len(x.unique())).sort_values(
    ascending=False)
# 找出持券消费人数>500 的商家 id
popular_merchant500 = popular_merchant[popular_merchant > 500]
popular_merchant500.name = 'customer_count'   # 指定列名为持券消费者数量
```

```
Merchant_500 = pd.merge(left=popular_merchant500, right=merchant_discount_rate,
on='Merchant_id', how='inner')
merchant_500 = pd.merge(left=Merchant_500,
right=coup_consume.groupby('Merchant_id')['Distance'].mean(),
                                    on='Merchant_id', how='inner')
# 绘制热力图
sns.heatmap(merchant_500.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```

先找出持券消费人数在 500 人以上商家，然后连接顾客到店平均距离和平均折扣力度，再用 corr 来计算数据中列与列的的相关性（皮尔逊相关系数），取值范围[-1,1]之间，1 表示完全正相关，-1 表示完全负相关。

可视化结果如下:



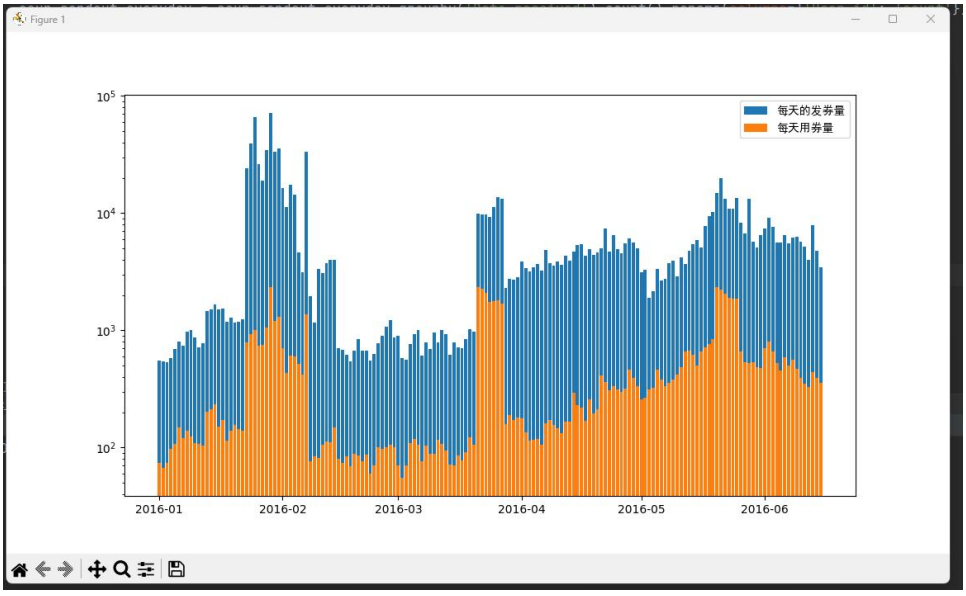
由图可知，到店消费人数与顾客到店距离之间呈现负相关，相关系数 0.31，在 0.3-0.5 之间，为低度相关；到店消费人数的多少与优惠打折力度呈现负相关，相关系数 0.2，在 0-0.3 之间，为相关程度极弱；

综上所述，这些消费人数在 500 人以上的店家之所以火爆，与距离和优惠力度相关性不大，可能是物美价廉等其他原因导致的。

2.10 分析每天优惠券总体发放和使用情况：

```
# 取出存在领券日期的记录，进行升序和去重
Date_received_sort =
off_train[off_train['Date_received'].notnull()]['Date_received'].sort_values().unique()
# 每天优惠券的使用量（持券消费人群）,coup_consume 指的是领券且有消费的用户
consume_num_everday = coup_consume[['User_id', 'Date_received']]
consume_num_everday =
consume_num_everday.groupby('Date_received').count().rename(columns={'User_id': 'count'})
# 每天发放的优惠券数量（取出所有领券日期！=null 的数据）
coup_sendout_everyday = off_train[off_train['Date_received'].notnull()][['Date_received', 'User_id']]
coup_sendout_everyday =
coup_sendout_everyday.groupby('Date_received').count().rename(columns={'User_id': 'count'})
# 绘制每天的发券量和每天的用券量
plt.figure(figsize=(18, 6))
plt.bar(x=Date_received_sort,
        height=coup_sendout_everyday['count'],
        label='每天的发券量')
plt.bar(x=Date_received_sort,
        height=consume_num_everday['count'],
        label='每天用券量')
# 对 y 轴进行对数缩放
plt.yscale('log')
plt.legend(prop={'family': 'SimHei'})
plt.show()
```

计算每天优惠券的使用量和发券量，并可视化结果如下:



由图可知，优惠券使用率最高在 16 年三月底，达到 30%左右，最低在 16 年 1 月底，最低为 3%左右。总体看优惠券使用率波动较大。

3. 特征工程：

3.1 生成交叉数据集

为什么要生成一个交叉数据集?

- 天池官方提供的数据集拥有两个数据集，一个是线上训练集，另一个是线下训练集，而我们训练的时候期望的是使用一个训练集进行训练。
- 通过生成交叉训练集合并线上线下数据集特征。
- 通过生成交叉数据集探索线上线下数据集各自内部可挖掘的特征以及之间的交互特征。
- 通过生成交叉数据集生成时序上的一些特征。

交叉训练集一：

```
# 交叉训练集一：收到券的日期大于 4 月 14 日和小于 5 月 14 日
time_range = ['2016-04-16', '2016-05-15']
dataset1 = off_train[(off_train.Date_received >= time_range[0]) & (off_train.Date_received <=
time_range[1])].copy()
dataset1['label'] = 0
dataset1.loc[
    (dataset1.Date != date_null) & (dataset1.Date - dataset1.Date_received <= datetime.timedelta(15)),
'label'] = 1
# 交叉训练集一特征 offline：线下数据中领券和用券日期大于 1 月 1 日和小于 4 月 13 日
time_range_date_received = ['2016-01-01', '2016-03-31']
time_range_date = ['2016-01-01', '2016-04-15']
feature1_off = off_train[(off_train.Date >= time_range_date[0]) & (off_train.Date <=
time_range_date[1]) | (
        (off_train.Coupon_id == 0) & (off_train.Date_received >= time_range_date_received[0]) & (
        off_train.Date_received <= time_range_date_received[1]))]
# 交叉训练集一特征 online：线上数据中领券和用券日期大于 1 月 1 日和小于 4 月 13 日
[on_train.date == 'null' to on_train.coupon_id == 0]
feature1_on = on_train[(on_train.Date >= time_range_date[0]) & (on_train.Date <=
time_range_date[1]) | (
        (on_train.Coupon_id == 0) & (on_train.Date_received >= time_range_date_received[0]) & (
        on_train.Date_received <= time_range_date_received[1]))]
```

我们使用了线下训练集中用户收到优惠券日期大于 4 月 14 日并且小于 5 月 14 日的数据项作为第一个交叉训练集的初始版本。然后使用线下数据集中满足用券日期大于 1 月 1 日并且小于 3 月 31 日，领券日期在 1 月 1 日和 3 月 31 日之间但是没有使用优惠券这两个条件之一的数据项作为该交叉训练集的第一个特

征集。使用线上数据集中满足上述条件之一的数据项作为该交叉数据集的第二个特征集。

交叉训练集二：

```
# 交叉训练集二：收到券的日期大于 5 月 15 日和小于 6 月 15 日
time_range = ['2016-05-16', '2016-06-15']
dataset2 = off_train[(off_train.Date_received >= time_range[0]) & (off_train.Date_received <=
time_range[1])]
dataset2['label'] = 0
dataset2.loc[
    (dataset2.Date != date_null) & (dataset2.Date - dataset2.Date_received <= datetime.timedelta(15)),
'label'] = 1
# 交叉训练集二特征 offline：线下数据中领券和用券日期大于 2 月 1 日和小于 5 月 14 日
time_range_date_received = ['2016-02-01', '2016-04-30']
time_range_date = ['2016-02-01', '2016-05-15']
feature2_off = off_train[(off_train.Date >= time_range_date[0]) & (off_train.Date <=
time_range_date[1]) | (
        (off_train.Coupon_id == 0) & (off_train.Date_received >= time_range_date_received[0]) & (
        off_train.Date_received <= time_range_date_received[1]))]
# 交叉训练集二特征 online：线上数据中领券和用券日期大于 2 月 1 日和小于 5 月 14 日
feature2_on = on_train[(on_train.Date >= time_range_date[0]) & (on_train.Date <=
time_range_date[1]) | (
        (on_train.Coupon_id == 0) & (on_train.Date_received >= time_range_date_received[0]) & (
        on_train.Date_received <= time_range_date_received[1]))]
```

同理，我们使用了线下训练集中用户收到优惠券日期大于 5 月 16 日并且小于 6 月 15 日的数据项作为第二个交叉训练集的初始版本。然后使用线下数据集中满足用券日期大于 2 月 1 日并且小于 5 月 15 日，领券日期在 2 月 1 日和 4 月 30 日之间但是没有使用优惠券这两个条件之一的数据项作为该交叉训练集的第一个特征集。使用线上数据集中满足上述条件之一的数据项作为该交叉数据集的第二个特征集。

3.2 测试集

因为我们训练时使用的是上述的交叉训练集，为了在测试的时候能达到更好的成绩，我们对测试集也进行了相似的处理。

```
# 测试集
dataset3 = off_test
# 测试集特征 offline :线下数据中领券和用券日期大于 3 月 15 日和小于 6 月 30 日的
time_range = ['2016-03-16', '2016-06-30']
feature3_off = off_train[((off_train.Date >= time_range[0]) & (off_train.Date <= time_range[1])) | (
        (off_train.Coupon_id == 0) & (off_train.Date_received >= time_range[0]) & (
        off_train.Date_received <= time_range[1]))]
# 测试集特征 online :线上数据中领券和用券日期大于 3 月 15 日和小于 6 月 30 日的
```

```
feature3_on = on_train[((on_train.Date >= time_range[0]) & (on_train.Date <= time_range[1])) | (
        (on_train.Coupon_id == 0) & (on_train.Date_received >= time_range[0]) & (
        on_train.Date_received <= time_range[1]))]
```

对于测试集，我们需要对每一个数据项都进行预测，所以就没有根据日期筛选数据项。与训练集同理，我们选用了线下数据集中满足用券日期大于 3 月 16 日并且小于 6 月 30 日，领券日期在 3 月 16 日和 6 月 30 日之间但是没有使用优惠券这两个条件之一的数据项作为该交叉训练集的第一个特征集。使用线上数据集中满足上述条件之一的数据项作为该交叉数据集的第二个特征集。

3.3 生成特征

对于先前生成的交叉训练集一，交叉训练集二，以及训练集，我们会进一步生成以下特征。

生成线下消费相关特征:

因为预测时采用的是线下消费的数据，所以我们生成了许多与线下消费的相关特征。

主要分为用户，商家，优惠券的相关特征。

用户特征：

```
'''user features'''

# 优惠券消费次数
temp = user_coupon_consume.size().reset_index(name='u2')
X = pd.merge(X, temp, how='left', on='User_id')
#X.u2.fillna(0, inplace=True)
#X.u2 = X.u2.astype(int)

# 优惠券不消费次数
temp = coupon_no_consume.groupby('User_id').size().reset_index(name='u3')
X = pd.merge(X, temp, how='left', on='User_id')

# 使用优惠券次数与没使用优惠券次数比值
X['u19'] = X.u2 / X.u3

# 领取优惠券次数
X['u1'] = X.u2.fillna(0) + X.u3.fillna(0)

# 优惠券核销率
X['u4'] = X.u2 / X.u1

# 普通消费次数
temp = offline[(offline.Coupon_id == 0) & (offline.Date != date_null)]
temp1 = temp.groupby('User_id').size().reset_index(name='u5')
```

```
X = pd.merge(X, temp1, how='left', on='User_id')

# 一共消费多少次
X['u25'] = X.u2 + X.u5
. . . . . .
```

共有以下用户特征：

关于优惠券消费次数的特征：
- u2：用户优惠券消费次数
- u3：优惠券不消费次数
- u19：使用优惠券次数与没使用优惠券次数的比值
- u1：领取优惠券次数（包括消费和未消费）

关于消费次数的特征：
- u5：普通消费次数
- u25：总消费次数（包括优惠券消费和普通消费）

消费频率与间隔的特征：
- u6：正常消费平均间隔
- u7：优惠券消费平均间隔
- u8：15 天内平均普通消费次数
- u9：15 天内平均优惠券消费次数

优惠券使用时间特征：
- u10：领取优惠券到使用优惠券的平均间隔时间
- u11：在 15 天内使用掉优惠券的值大小
- u21：领取优惠券到使用优惠券间隔小于 15 天的次数
- u22，u23，u24：不同比例衡量用户 15 天内使用优惠券的次数

折扣率相关特征：
- u45：消费优惠券的平均折率
- u27：用户核销优惠券的最低消费折率
- u28：用户核销优惠券的最高消费折率

优惠券种类与数量特征：
- discount_type：优惠券类型编码
- u32：用户核销过的不同优惠券数量
- u47：用户领取所有不同优惠券数量
- u33：用户核销过的不同优惠券数量占所有不同优惠券的比重
- u34：用户平均每种优惠券核销多少张

用户-商家距离相关特征：

u35：核销优惠券用户-商家平均距离
u36：用户核销优惠券中的最小用户-商家距离
u37：用户核销优惠券中的最大用户-商家距离

不同优惠券类型的特征：

u41：不同优惠券领取次数
u42：不同优惠券使用次数
u43：不同优惠券不使用次数
u44：不同打折优惠券使用率
u48：满减类型优惠券领取次数
u49：打折类型优惠券领取次数

商家特征：

消费相关特征：

m0：商户的消费次数
m1：商家优惠券被领取后核销次数
m2：商户正常消费笔数（通过消费次数和核销次数计算）

优惠券领取和核销情况：

m3：商家优惠券被领取次数
m4：商家优惠券被领取后核销率
m7：商家优惠券被领取后不核销次数

当天优惠券领取情况：

m5：商户当天优惠券领取次数
m6：商户当天优惠券领取人数

优惠券折扣相关特征：

m8：商家优惠券核销的平均消费折率
m9：商家优惠券核销的最小消费折率
m10：商家优惠券核销的最大消费折率

```
'''offline merchant features'''

# 商户消费次数
temp = offline[offline.Date != date_null].groupby('Merchant_id').size().reset_index(name='m0')
X = pd.merge(X, temp, how='left', on='Merchant_id')

# 商家优惠券被领取后核销次数
temp = coupon_consume.groupby('Merchant_id').size().reset_index(name='m1')
X = pd.merge(X, temp, how='left', on='Merchant_id')
```

```
# 商户正常消费笔数
X['m2'] = X.m0.fillna(0) - X.m1.fillna(0)

# 商家优惠券被领取次数
temp = offline[offline.Date_received !=
date_null].groupby('Merchant_id').size().reset_index(name='m3')
X = pd.merge(X, temp, how='left', on='Merchant_id')

# 商家优惠券被领取后核销率
X['m4'] = X.m1 / X.m3

# 商家优惠券被领取后不核销次数
temp = coupon_no_consume.groupby('Merchant_id').size().reset_index(name='m7')
X = pd.merge(X, temp, how='left', on='Merchant_id')

# 商户当天优惠券领取次数
temp = X[X.Date_received != date_null]
temp = temp.groupby(['Merchant_id', 'Date_received']).size().reset_index(name='m5')
X = pd.merge(X, temp, how='left', on=['Merchant_id', 'Date_received'])

# 商户当天优惠券领取人数
temp = X[X.Date_received != date_null]
temp = temp.groupby(['User_id', 'Merchant_id', 'Date_received']).size().reset_index()
temp = temp.groupby(['Merchant_id', 'Date_received']).size().reset_index(name='m6')
X = pd.merge(X, temp, how='left', on=['Merchant_id', 'Date_received'])

# 商家优惠券核销的平均消费折率
temp = coupon_consume.groupby('Merchant_id').discount_rate.mean().reset_index(name='m8')
X = pd.merge(X, temp, how='left', on='Merchant_id')

# 商家优惠券核销的最小消费折率
temp = coupon_consume.groupby('Merchant_id').discount_rate.max().reset_index(name='m9')
X = pd.merge(X, temp, how='left', on='Merchant_id')

# 商家优惠券核销的最大消费折率
temp = coupon_consume.groupby('Merchant_id').discount_rate.min().reset_index(name='m10')
X = pd.merge(X, temp, how='left', on='Merchant_id')
......
```

用户相关特征：

m11：商家优惠券核销不同的用户数量
m12：商家优惠券领取过的不同用户数量
m13：核销商家优惠券的不同用户数量占领取不同的用户比重
m14：商家优惠券平均每个用户核销多少张

不同优惠券相关特征：

m15：商家被核销过的不同优惠券数量

m18：商家领取过的不同优惠券数量的比重

m19：商家被核销过的不同优惠券数量占所有领取过的不同优惠券数量的比重

优惠券核销时间和距离：

m20：商家被核销优惠券的平均时间

m21：商家被核销优惠券中的用户-商家平均距离

m22：商家被核销优惠券中的用户-商家最小距离

m23：商家被核销优惠券中的用户-商家最大距离

优惠券特征：

```
"""offline coupon features"""

# 此优惠券一共发行多少张
temp = offline[offline.Coupon_id != 0].groupby('Coupon_id').size().reset_index(name='c1')
X = pd.merge(X, temp, how='left', on='Coupon_id')

# 此优惠券一共被使用多少张
temp = coupon_consume.groupby('Coupon_id').size().reset_index(name='c2')
X = pd.merge(X, temp, how='left', on='Coupon_id')

# 优惠券使用率
X['c3'] = X.c2 / X.c1

# 没有使用的数目
X['c4'] = X.c1 - X.c2

# 此优惠券在当天发行了多少张
temp = X.groupby(['Coupon_id', 'Date_received']).size().reset_index(name='c5')
X = pd.merge(X, temp, how='left', on=['Coupon_id', 'Date_received'])

# 优惠券类型(直接优惠为0，满减为1)
X['c6'] = 0
X.loc[X.Discount_rate.str.contains(':') == True, 'c6'] = 1

# 不同打折优惠券领取次数
temp = offline.groupby('Discount_rate').size().reset_index(name='c8')
X = pd.merge(X, temp, how='left', on='Discount_rate')

# 不同打折优惠券使用次数
temp = coupon_consume.groupby('Discount_rate').size().reset_index(name='c9')
X = pd.merge(X, temp, how='left', on='Discount_rate')
```

.....

发行和使用情况：

c1：优惠券一共发行多少张

c2：优惠券一共被使用多少张

c3：优惠券使用率

c4：没有使用的优惠券数目

c5：优惠券在当天发行了多少张

优惠券类型和不同打折优惠券使用情况：

c6：优惠券类型（直接优惠为 0，满减为 1）

c8：不同打折优惠券领取次数

c9：不同打折优惠券使用次数

c10：不同打折优惠券不使用次数

c11：不同打折优惠券使用率

优惠券核销时间：

c12：优惠券核销平均时间

用户与商家交互特征：

```
'''user merchant feature'''

# 用户领取商家的优惠券次数
temp = offline[offline.Coupon_id != 0]
temp = temp.groupby(['User_id', 'Merchant_id']).size().reset_index(name='um1')
X = pd.merge(X, temp, how='left', on=['User_id', 'Merchant_id'])

# 用户领取商家的优惠券后不核销次数
temp = coupon_no_consume.groupby(['User_id', 'Merchant_id']).size().reset_index(name='um2')
X = pd.merge(X, temp, how='left', on=['User_id', 'Merchant_id'])

# 用户领取商家的优惠券后核销次数
temp = coupon_consume.groupby(['User_id', 'Merchant_id']).size().reset_index(name='um3')
X = pd.merge(X, temp, how='left', on=['User_id', 'Merchant_id'])

# 用户领取商家的优惠券后核销率
X['um4'] = X.um3 / X.um1

# 用户对每个商家的不核销次数占用户总的不核销次数的比重
temp = coupon_no_consume.groupby('User_id').size().reset_index(name='temp')
X = pd.merge(X, temp, how='left', on='User_id')
X['um5'] = X.um2 / X.temp
X.drop(columns='temp', inplace=True)
.....
```

um1：用户领取商家的优惠券次数
um2：用户领取商家的优惠券后不核销次数
um3：用户领取商家的优惠券后核销次数
um4：用户领取商家的优惠券后核销率
um5：用户对每个商家的不核销次数占用户总的不核销次数的比重
um6：用户在商店总共消费过几次
um7：用户在商店普通消费次数
um8：用户当天在此商店领取的优惠券数目
um9：用户领取优惠券不同商家数量
um10：用户核销优惠券不同商家数量
um11：用户核销过优惠券的不同商家数量占所有不同商家的比重
um12：用户平均核销每个商家多少张优惠券

生成线下消费相关特征：

相比与线下数据集，线上数据集的重要性就没有那么强，所以生成的特征也相对较少。

```
# 用户线上操作次数
temp = online.groupby('User_id').size().reset_index(name='on_u1')
X = pd.merge(X, temp, how='left', on='User_id')

# 用户线上点击次数
temp = online[online.Action == 0].groupby('User_id').size().reset_index(name='on_u2')
X = pd.merge(X, temp, how='left', on='User_id')

# 用户线上点击率
X['on_u3'] = X.on_u2 / X.on_u1

# 用户线上购买次数
temp = online[online.Action == 1].groupby('User_id').size().reset_index(name='on_u4')
X = pd.merge(X, temp, how='left', on='User_id')

# 用户线上购买率
X['on_u5'] = X.on_u4 / X.on_u1

# 用户线上领取次数
temp = online[online.Coupon_id != 0].groupby('User_id').size().reset_index(name='on_u6')
X = pd.merge(X, temp, how='left', on='User_id')

# 用户线上领取率
X['on_u7'] = X.on_u6 / X.on_u1
......
```

用户线上操作次数：

on_u1: 用户线上操作次数

用户线上点击行为相关特征：

on_u2: 用户线上点击次数

on_u3: 用户线上点击率（on_u2 除以 on_u1）

用户线上购买行为相关特征：

on_u4: 用户线上购买次数

on_u5: 用户线上购买率（on_u4 除以 on_u1）

用户线上优惠券领取行为相关特征：

on_u6: 用户线上领取次数

on_u7: 用户线上领取率（on_u6 除以 on_u1）

用户线上优惠券核销行为相关特征：

on_u8: 用户线上不消费次数

on_u9: 用户线上优惠券核销次数

on_u10: 用户线上优惠券核销率（on_u9 除以 on_u6）

其他线上与线下行为关联特征：

on_u11: 用户线下的不消费次数占线上线下总的不消费次数的比重

on_u12: 用户线下的优惠券核销次数占线上线下总的优惠券核销次数的比重

on_u13: 用户线下领取的记录数量占总的记录数量的比重

4. 建模与调参：

4.1. 模型的对比：

4.1.1. 简单线性模型：

优点：解释性强、训练速度快、适用性广、对稀疏数据友好

但它也存在许多缺点：无法处理非线性问题、对异常值敏感、不适合高维稀疏数据、复杂度低、容易出现欠拟合

4.1.2. 决策树模型

优点：易于理解和解释、能够处理分类和回归问题、对特征的缩放不敏感

缺点：过拟合风险大、不稳定、处理缺失值困难

4.1.3. 随机森林

优点：高准确性、降低过拟合风险、不需要特征缩放、适用于大规模数据集

缺点：可解释性较差、计算资源消耗较大、不适用于稀疏数据

4.1.4. LightGBM

优点：高效性、准确性、支持类别特征、可扩展性

缺点：过拟合风险、调参难度大、不适用于小数据集

4.1.5. XGBoost（eXtreme Gradient Boosting）：是一种高效的、灵活的梯度提升树（Gradient Boosting Tree）算法。

XGBoost 算法有以下优点：高性能、准确性、可解释性、灵活性、防止过拟合、并行处理等。

缺点：内存占用较大、不直接支持类别特征

4.2. 模型的选择

由于我们在特征提取工程方面做了大量的准备工作，在完成特征提取之后，我们提取出了近 100 个数据特征项。

因此，根据数据特征的复杂性，我们选择了排除了线性模型这种简单且容易出现欠拟合现象的模型。

根据模型的特点考虑，我们可能会选择 LightGBM 和 XGBoost 中的一种。

对比 LightGBM 和 XGBoost，我们发现，虽然 XGBoost 不直接支持类别特征、效率较 LightGBM 低且内存占用较大，但是我们已经做了大量的数据特征提取工作，且我们的设备运算能力足够，于是该缺点可以忽略不计，同时 XGBoost 提供了较为清晰的特征重要性排序和模型解释能力，并且相对于 LightGBM，它对于一些具有噪声的数据或者不规则数据的适应能力更强。

读取数据并进行归一化后，根据单一特征，简单地输出对比四种模型的 AUC 值：

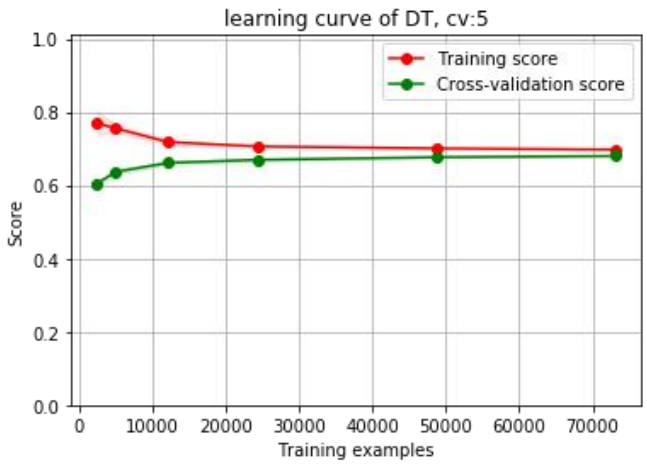
```
#所有的特征都是上一节生成的
train_f1,test_f1=read_data('f1')
#因为要使用KNN等进行测试，所以需要归一化
train_f1,test_f1=standize_df(train_f1,test_f1)
```

决策树：

```
test_model(train_f1,'DT')
```

```
DT 总体 AUC: 0.648173530235

DT Coupon AUC: 0.53032721978
```

```
plot_curve_single(train_f1,'DT',5,[0.01,0.02,0.05,0.1,0.2,0.3])
```

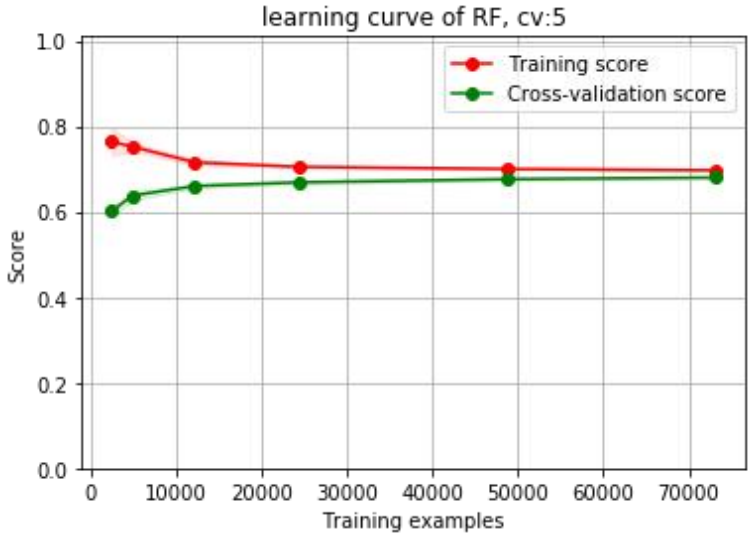


随机森林：

```
test_model(train_f1,'RF')
```

```
RF总体AUC: 0.652550874194

RFCoupon AUC: 0.531847399078
```

```
plot_curve_single(train_f1,'RF',5,[0.01,0.02,0.05,0.1,0.2,0.3])
```

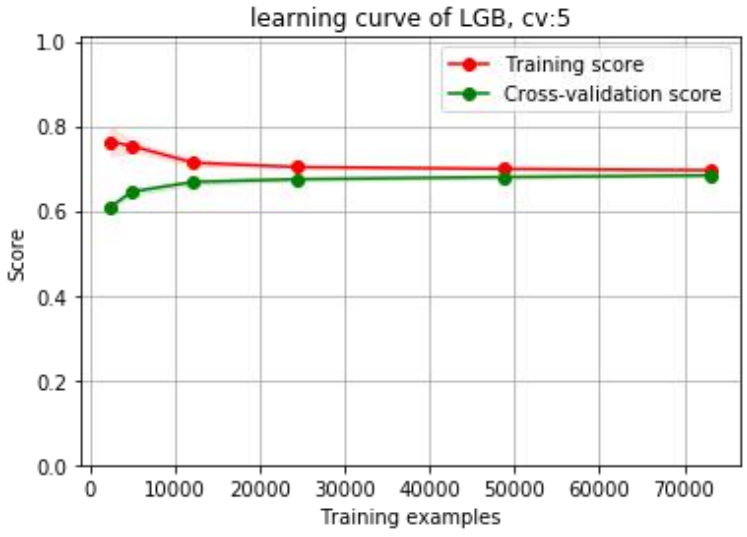


LightGBM：

```
test_model(train_f1,'LGB')
```

```
LGB总体AUC: 0.653383754968

LGBCoupon AUC: 0.532266808736
```

```
plot_curve_single(train_f1,'LGB',5,[0.01,0.02,0.05,0.1,0.2,0.3])
```

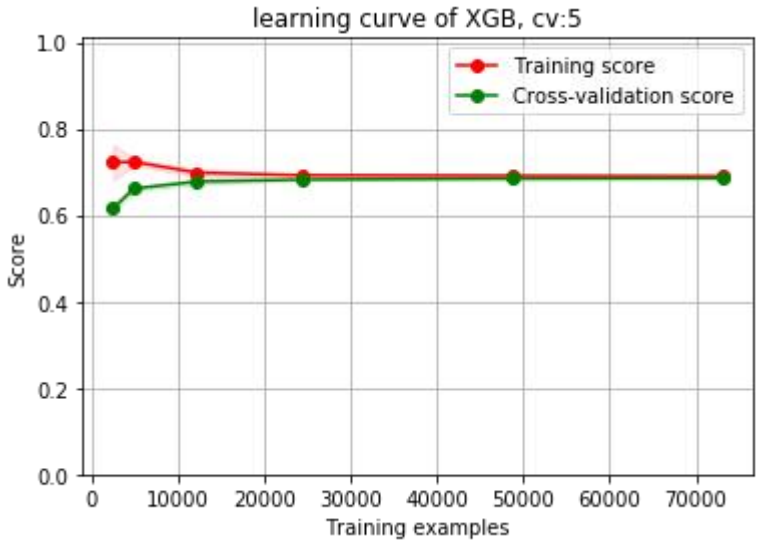


XGBoost：

```
test_model(train_f1,'XGB')
```

```
XGB总体AUC: 0.655617912093

XGBCoupon AUC: 0.534181454485
```

```
plot_curve_single(train_f1,'XGB',5,[0.01,0.02,0.05,0.1,0.2,0.3])
```



从评估结果来看，XGBoost 比 LightGBM 略胜一筹！

于是，我们选择 XGBoost 算法迭代地构建决策树来学习数据中的复杂模式，从而构建目标模型。

4.3. XGBoost 模型的搭建与训练

4.3.1. 读取特征提取后的数据并进行部分合并

```
def get_processed_data():
    dataset1 = pd.read_csv('./ProcessDataSet1.csv')
    dataset2 = pd.read_csv('./ProcessDataSet2.csv')
    dataset3 = pd.read_csv('./ProcessDataSet3.csv')

    dataset1.drop_duplicates(inplace=True)
    dataset2.drop_duplicates(inplace=True)
    dataset3.drop_duplicates(inplace=True)

    dataset12 = pd.concat([dataset1, dataset2], axis=0)

    dataset12.fillna(0, inplace=True)
    dataset3.fillna(0, inplace=True)

    return dataset12, dataset3
```

4.3.2. 转换数据格式以便模型训练

```
# 将数据转化为 dmatric 格式
dataset12_x = dataset12.drop(
        columns=['User_id', 'Merchant_id', 'Discount_rate', 'Date_received',
'discount_rate_x', 'discount_rate_y',
                 'Date', 'Coupon_id', 'label'], axis=1)
dataset3_x = dataset3.drop(
        columns=['User_id', 'Merchant_id', 'Discount_rate', 'Date_received',
'discount_rate_x', 'discount_rate_y',
                 'Coupon_id'], axis=1)

train_dmatrix = xgb.DMatrix(dataset12_x, label=dataset12.label)
predict_dmatrix = xgb.DMatrix(dataset3_x)
```

4.3.3. 使用 xgboost.cv 交叉验证优化模型参数

```
#xgboost 模型训练
params = {'booster': 'gbtree',
                  'objective': 'binary:logistic',
                  'eval_metric': 'auc',
                  'gamma': 0.1,
                  'min_child_weight': 3.3,
                  'max_depth': 6,
                  'lambda': 10,
                  'subsample': 0.7,
                  'colsample_bytree': 0.7,
                  'colsample_bylevel': 0.7,
                  'eta': 0.005,
                  'tree_method': 'auto',
                  'n_gpus': '-1',
                  'seed': 0,
                  'nthread': cpu_jobs,
                  'predictor': 'cpu_predictor'
                  }

# 使用 xgb.cv 优化 num_boost_round 参数
cvresult = xgb.cv(params, train_dmatrix, num_boost_round=30000, nfold=2,
metrics='auc', seed=0, callbacks=[
        xgb.callback.print_evaluation(show_stdv=False),
        xgb.callback.early_stop(100)
])
num_round_best = cvresult.shape[0] - 1
print('Best round num: ', num_round_best)
```

4.3.4. 使用优化后的模型进行预测并保存模型数据

```
# 使用优化后的 num_boost_round 参数训练模型
watchlist = [(train_dmatrix, 'train')]
model = xgb.train(params, train_dmatrix, num_boost_round=num_round_best,
evals=watchlist)

model.save_model('train_dir_2/xgbmodel')
params['predictor'] = 'cpu_predictor'
model = xgb.Booster(params)
model.load_model('train_dir_2/xgbmodel')

# predict test set
dataset3_predict = predict_dataset.copy()
dataset3_predict['label'] = model.predict(predict_dmatrix)
```

4.3.5. 进行数据整理后写入到结果数据集

```
# 标签归一化
dataset3_predict.label = MinMaxScaler(copy=True, feature_range=(0,
1)).fit_transform(
        dataset3_predict.label.values.reshape(-1, 1))
dataset3_predict.sort_values(by=['Coupon_id', 'label'], inplace=True)
dataset3_predict.to_csv("train_dir_2/xgb_preds.csv", index=None,
header=None)
print(dataset3_predict.describe())
```

4.3.6. 进行模型预测的结果评估

```
def myauc(test):
        testgroup = test.groupby(['Coupon_id'])
        aucs = []
        for i in testgroup:
                tmpdf = i[1]
                if len(tmpdf['label'].unique()) != 2:
                        continue
                fpr, tpr, thresholds =
          roc_curve(tmpdf['label'],tmpdf['pred'], pos_label=1)
                aucs.append(auc(fpr, tpr))
        return np.average(aucs)

# 输出评估结果
temp = dataset12[['Coupon_id', 'label']].copy()
temp['pred'] = model.predict(xgb.DMatrix(dataset12_x))
temp.pred = MinMaxScaler(copy=True, feature_range=(0,
1)).fit_transform(temp['pred'].values.reshape(-1, 1))
print(myauc(temp))
```

4.4. 其它模型的尝试

4.4.1 GradientBoostingClassifer 算法

简介：GradientBoostingClassifier（梯度提升分类器）是一种集成学习算法，它属于集成学习中的提升方法之一。它通过串行训练多个弱分类器（通常是决策树），每次训练都试图修正前一个模型的错误，从而逐步提升整体模型的性能。

```
def train_gbdt(model=False):
        global log

    params = grid_search_gbdt(True)
     clf = GradientBoostingClassifier().set_params(**params)

     if model:
          return clf

     params = clf.get_params()

        return train(clf)
```

grid_search_gbdt 方法是一个寻找较优秀参数的方法。
train 是模型的训练方法。
但是，这个算法最终提交得到的 auc 只有 0.77 左右，并没有突破。

4.4.2 LGBMClassifier 算法

简介：LGBMClassifier 基于梯度提升决策树（GBDT）算法，但它采用了一些优化策略，例如基于直方图的决策树学习和 Leaf-wise 的树生长策略，以提高模型训练速度和准确性。它还支持并行学习和处理大规模数据集，适用于高维稀疏特征。

```
def train_lgb(model=False):
        global log

     params = grid_search_lgb(True)

      clf = LGBMClassifier().set_params(**params)

      if model:
            return clf

      params = clf.get_params()

      return train(clf)
```

grid_search_lgb 用于探索模型的较优的参数。
train 方法用来训练模型。
最后这个算法能得到的 auc 成绩大概是 0.78 分。

4.4.3 CatBoostClassifier

简介：CatBoostClassifier 基于梯度提升决策树（GBDT）算法，但它具有一些独特的特性和优势。该库针对类别特征（categorical features）进行了优化，无需对类别特征进行额外的预处理或独热编码，它能够自动处理类别特征，并且具有较强的抗过拟合能力。

```
def train_cat(model=False):
        global log

          params = grid_search_cat(True)

        clf = CatBoostClassifier().set_params(**params)

          if model:
            return clf

          params = clf.get_params()

            return train(clf)
```

grid_search_cat 用来探索该模型的较优参数。

train 方法用来训练模型。

最后模型的 auc 分数有 0.786 左右，依旧是不如 xgb 算法。

4.4.4 ExtraTreesClassifier 算法

简介：ExtraTreesClassifier 是 scikit-learn 库中的一个机器学习分类器，属于集成学习中的一种，它基于极端随机树（Extremely Randomized Trees）的概念。

与常规的随机森林（Random Forest）相似，ExtraTreesClassifier 也是一种集成学习算法，使用多个决策树进行预测，但在构建每棵树的节点时具有一些不同之处。

在 ExtraTrees 中，与随机森林不同的是，它对每个节点的特征进行分割时采用了更强的随机性。对于每个节点的特征选择和分割点的选择，ExtraTrees 使用随机的特征子集和随机的分割点。这种额外的随机性可以增加模型的多样性，可能在一定程度上降低了方差，有时可以提高模型的泛化能力。

```
def train_et_gini(model=False):
        clf = ExtraTreesClassifier().set_params(**grid_search_et('gini', True))
          if model:
              return clf
          return train_et(clf)
```

grid_search_et 方法用来探索较优参数。

这个算法的结果并不理想，只有 0.75 分左右。

4.4.5 模型融合

```
skf = StratifiedKFold()
clfs = ['gbdt', 'xgb', 'lgb', 'cat',
                    ]

blend_X_train = np.zeros((X.shape[0], len(clfs)))
blend_X_test = np.zeros((X_submission.shape[0], len(clfs)))

for j, v in enumerate(clfs):
    clf = eval('train_%s' % v)(True)

    aucs = []
    dataset_blend_test_j = []

    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        clf = fit_eval_metric(clf, X_train, y_train)

        y_submission = clf.predict_proba(X_test)[:, 1]
        aucs.append(roc_auc_score(y_test, y_submission))

        blend_X_train[test_index, j] = y_submission
        dataset_blend_test_j.append(clf.predict_proba(X_submission)[:, 1])

    log += '%7s' % v + ' auc: %f\n' % np.mean(aucs)
    blend_X_test[:, j] = np.asarray(dataset_blend_test_j).T.mean(1)
```

尝试了对 gbdt，xgb，lgb，cat 四个模型进行融合，期望能够得到一个拟合效果更好的模型。但是融合的结果似乎并不是那么理想，效果并不如单一使用 xgb 算法好。

4.4.6 神经网络

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, num_hidden_layers, output_size,
dropout_prob=0.5):
        super(NeuralNetwork, self).__init__()
        self.layers = nn.ModuleList()
        self.layers.append(nn.Linear(input_size, hidden_size))

        # 添加隐藏层和 dropout 层
        for _ in range(num_hidden_layers - 1):
            self.layers.append(nn.Linear(hidden_size, hidden_size))
```

```
            self.layers.append(nn.ReLU())
            self.layers.append(nn.Dropout(p=dropout_prob))

        self.output_layer = nn.Linear(hidden_size, output_size)
        self.activation = nn.ReLU()

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        x = self.output_layer(x)
        return x
```

搭建了一个比较简单的神经网络模型，带有一些 dropoutlayer 防止过拟合问题。经过调参后 auc 分数也能达到 0.78 左右，但是还是无法突破 0.80。

4.5. 模型调优

4.5.1. 深入了解模型参数

- booster: **模型类型**，如 'gbtree'（梯度提升树）和 'gblinear'（梯度提升线性模型）。
- objective: 目标**函数**，这里选择的是二分类逻辑回归问题。
- eval_metric: **评估指标**，这里选择的是 AUC（Area Under Curve）。
- gamma: 用于控制树的叶子节点的**正则化参数**，值越大，模型越简单。
- min_child_weight: 每个叶子节点中**最小的样本数量**，值越大，模型越复杂。
- max_depth: 树的**最大深度**，值越大，模型越复杂。
- lambda: **L2 正则化参数**，值越大，模型越简单。
- subsample: 用于控制样本**采样比例**，值越大，采样比例越小。
- colsample_bytree: 用于控制**特征采样比例**，值越大，采样比例越小。
- colsample_bylevel: 用于控制树节点**分裂时的特征采样比例**，值越大，采样比例越小。
- eta: **学习率**。
- tree_method: 选择树的**构建方法**，如 'auto'、'exact' 和 'approx'。
- n_gpus: 使用的 **GPU 数量**，值为-1 表示使用所有可用的 GPU。
- seed: **随机数种子**，用于复现结果。
- nthread: **线程数量**，用于控制并行计算。
- predictor: **预测器类型**，如 'cpu_predictor'和'gpu_predictor'。

4.5.2. 关键参数确定

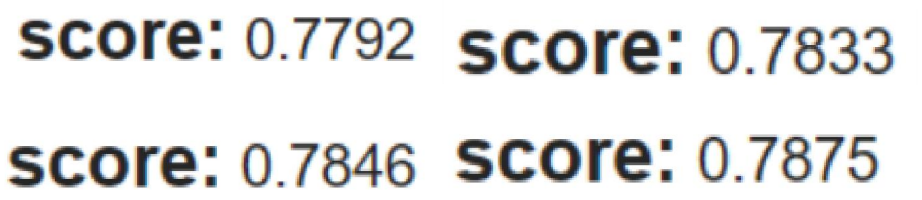
由于 xgboost 是一种梯度提升树算法，我们得出“max_depth”参数应该是一个关键性参数，决策树的深度对模型复杂度的影响应该是最大的。根据论坛上给出的参考，我们分别尝试了三种可能性取值：

```
'max_depth': 5,  'max_depth': 6,  'max_depth': 7,
```

使'max_depth': 5,保持不变，改变其他影响模型复杂度的参数：gamma、min_child_weight、lambda、eta，不断训练模型预测数据，并提交结果，发现该参数得到的 AUC 值波动较大，如下：

使'max_depth': 6,保持不变，改变其他影响模型复杂度的参数：gamma、min_child_weight、lambda、eta，不断训练模型预测数据，并提交结果，发现该参数得到的 AUC 值波动较小，且 AUC 值普遍较高，基本都在 0.79 以上，如下：

使'max_depth': 7,保持不变，改变其他影响模型复杂度的参数：gamma、min_child_weight、lambda、eta，不断训练模型预测数据，并提交结果，发现该参数得到的 AUC 值普遍较低，基本都在 0.79 以下，如下：



因此，我们选择了 6 作为关键参数“max_depth”的取值。

4.5.3. 调整其他参数

首先，我们尝试手动调整参数，手动运行，手动提交结果，在一次又一次的提交中优化模型参数。

可是，在此过程中，我们发现其中需要大量的人力、算力成本，调参过程其实是一个重复性的工作。此时，我们思考是否有一种方式可以自动化的去调整参数。

通过网络和老师的指导，我们了解到了“网格调参法”：

网格调参法（Grid Search）是一种超参数调优技术，它通过系统地遍历给定的超参数组合来寻找最佳的模型配置。这个过程通过指定要尝试的参数组合，然后使用交叉验证来评估每种组合的性能，最终选择性能最优的一组超参数。

这是一种“穷举”的暴力方法，但无疑解放了人力成本。

于是，我们开始借鉴这种思想：

首先定义一个参数列表，列表里的元素便是一组又一组的参数，参数由人工确定。列表的长度为 10，代表我们每运行一次都可以测试 10 组数据。

然后，我们把模型训练的代码放置到 for 循环中，每次循环中，模型的参数对应一开始定义的参数列表里面的元素（一组参数），执行后保存到结果数据集中。

注意：数据集命名需要动态更改，防止被覆盖。如下代码：

dataset3_predict.to_csv(f"train_dir_2/xgb_preds{i}.csv", index=None, header=None)

考虑到设备的资源消耗，我们在每次循环后，即没运行完一组参数后都会让程序睡眠 1000s time.sleep(1000) 从而防止设备运行过载。

大致思路如下图：

```
# 参数列表集
params_list = [...]
i = 1; #第几份数据
# 循环暴力调参
for params in params_list:...
```

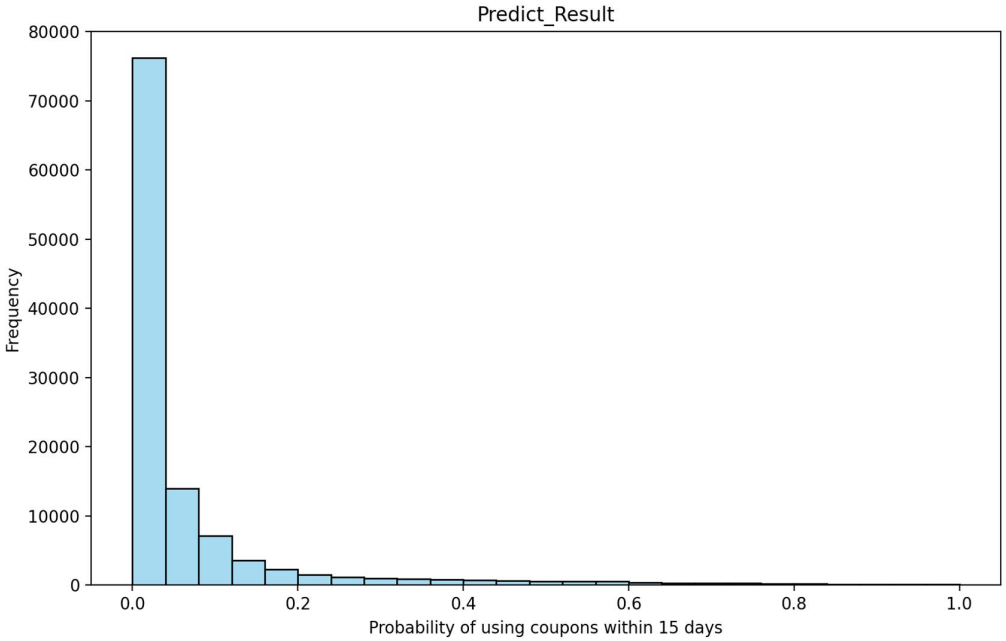
四、项目结果

预测结果可视化，利用直方图对结果进行可视化:

```
def draw_result():
    column_names=['User_id','Coupon_id','Date_received','Probability']

df=pd.read_csv('train_dir_2/xgb_preds.csv',header=None,names=column_names)

    # 绘测预测结果直方图
    plt.figure(figsize=(10,6))
    sns.histplot(df['Probability'],bins=25,color='skyblue')

    plt.title('Predict_Result')
```

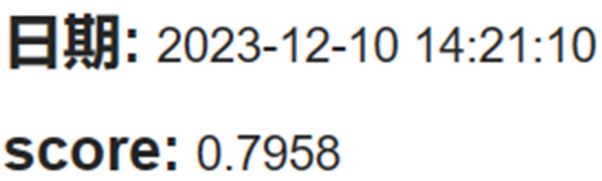
```
    plt.xlabel('Probability of using coupons within 15 days')
    plt.ylabel('Frequency')
    plt.show()
```

可视化结果:



通过预测结果，我们可以了解到用户在领取优惠券后 15 天内的使用情况，由图可知大部分用户在领取优惠券后的 15 天内，几乎不使用优惠券的用户占大多数，只有极少数的用户有使用优惠券。

最终比赛成绩:



算是一个相对不错的成绩。

五、改进思路

比较遗憾的就是这次比赛没能进入前百分之一，我们也在积极思考如何改进我们的模型以取得更好的成绩。

第一个方向，我们将进一步提高模型融合的处理，我们认为模型融合有着巨大的潜力突破，不过有些细节如模型融合的数量，一些重要的参数还需要我们继续探索试错。

第二个方向，对特征工程进行修改，我们生成的特征还是比较多的，有些特征可能对于模型训练并无帮助，所以我们将对生成的特征进行重要性评估，删除一些不重要的特征，留下重要的特征可能可以获得更好的成绩。

第三个方向，进一步优化我们的神经网路模型，比如使用别的激活函数，使用不同的优化器，加入别的 layer 来更好的防止过拟合问题。当然这些可能无法得到一些重大的突破，我们还将去搜寻一些与本赛题相关的论文，使用别人提出的网络框架以得到更好的成绩。

六、项目总结

通过这个比赛我们学到了很多关于机器学习的基础知识和实践技巧。以下是我们项目的总结。

首先，我们进行了数据预处理。这包括处理缺失值、异常值和重复数据，以确保数据的完整性和准确性。我们还对数据进行了归一化或标准化处理，以消除不同特征之间的量纲差异。通过数据预处理，我们保证了后续步骤的可靠性和准确性。

接着，我们进行了特征工程。我们根据对数据的分析，对原始数据进行转换、提取和组合，创建了新的特征变量，将原始数据转化为更有信息量、更适合模型建模的特征。它能够帮助我们挖掘数据中的潜在信息、降低维度、处理异常值和缺失值，提高模型的性能和解释能力。

最后，我们进行了模型选择，从众多模型中，我们尝试了线性模型、决策树模型、 随机森林、LightGBM 和 XGBoost，神经网络等模型，通过探索和比较，最终选择了 XGBoost 模型。然后对数据进行训练，根据提交结果的 auc 值调整模型的参数，如学习率、决策树深度、叶子节点最小权重和、正则化参数等。经过不断地调参，从手动调参到网格调参法，一点一点地提升 AUC 值，最终取得了的 **0.7958** 成绩，排名 **847 / 24392** 。

通过这个项目，我们对机器学习的整个流程有了更深入的理解。从数据预处理到特征工程，再到模型选择，每个环节都扮演着重要的角色。我们学到了如何处理和准备数据，如何查看数据统计量，如何通过可视化分析洞察数据，如何构建有意义的数据特征，如何选择合适的模型进行预测，并根据模型复杂度调参，从而优化模型。

这次比赛对我们来说是一个宝贵的机会，通过实践我们巩固了机器学习的基础知识，并学会了如何应用这些知识解决实际问题。我们通过团队合作、积极探索和互相学习取得了不错的成绩。这次经历让我们对机器学习产生了更大的兴趣，让我们在真正的比赛环境的实践中掌握了很多有用的知识，拓宽了学科的见

识，积累了一些比赛方面的经验，也激发了对数学建模竞赛的兴趣。

七、分工

林宪亮： 特征工程，xgb 模型调优，GradientBoostingClassifer，LGBMClassifier，CatBoostClassifier，ExtraTreesClassifier，神经网络算法的尝试，报告撰写，ppt 制作，汇报展示。

吴嘉楷：对比线性模型、决策树模型、随机森林、LightGBM、xgboost 模型并从中进行选择，xgboost 模型训练，使用网格调参法进行 xgb 参数调优，报告撰写，ppt 制作，QA 答辩。

朱粤锋：数据预处理，异常值、重复值、缺失值处理，数据可视化，xgboost 参数调优，报告撰写，ppt 制作。