数据科学家毕业项目

项目概述

通过星巴克用户推送实验数据，实现用户细分，找到最能够激发顾客消费的推送。目的在于抛弃全量推送的方案，采取个性化推荐的精细化运营提高用户复购率和留存。

问题描述

从用户日志中挖掘对于不同推送的反馈，构建 3 类特征，用户基础特征（性别，收入等），购物特征（消费频次，客单价），推送反应特征（推送打开率，补贴力度，核销率），对行为特征进行特征工程，再使用 kmeans 进行分层打标。最后使用基础特征交叉验证建立预测模型，用于后续用户分群使用

评价指标

- F1score
- Accuracy=预测准确量/总样本量

探索性数据分析

一共有三个数据文件：

- portfolio.json – 包括推送的 id 和每个推送的元数据（持续时间、种类等等）
- profile.json – 每个顾客的人口统计数据
- transcript.json – 交易、收到的推送、查看的推送和完成的推送的记录

	age	became_member_on	income
count	17000.000000	1.700000e+04	14825.000000
mean	62.531412	2.016703e+07	65404.991568
std	26.738580	1.167750e+04	21598.299410
min	18.000000	2.013073e+07	30000.000000
25%	45.000000	2.016053e+07	49000.000000
50%	58.000000	2.017080e+07	64000.000000
75%	73.000000	2.017123e+07	80000.000000
max	118.000000	2.018073e+07	120000.000000

年纪为 118 的老年人达到了 2175 名且大多没有收入数据，对于这部分年纪过大没有收入的人群做剔除。

	age	became_member_on	income
count	14825.000000	1.482500e+04	14825.000000
mean	54.393524	2.016689e+07	65404.991568
std	17.383705	1.188565e+04	21598.299410
min	18.000000	2.013073e+07	30000.000000
25%	42.000000	2.016052e+07	49000.000000
50%	55.000000	2.017080e+07	64000.000000
75%	66.000000	2.017123e+07	80000.000000
max	101.000000	2.018073e+07	120000.000000

经过数据清洗，总用户数 14825 人平均年龄 54.4 岁，中位数 55 岁，样本群体以中老年为主，平均年收入 65404 美金。

Transcript 表格中的推送 id 包裹在 json 字段中，需要做进一步数据处理才能做分析

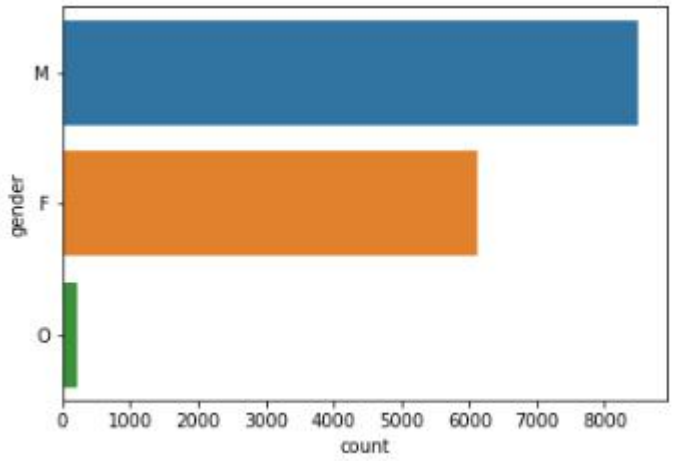
transaction	138953
offer received	76277
offer viewed	57725
offer completed	33579

总交易 138953 单，推送 76277 次，用户打开推送 57725 次，使用优惠 33579 次
推送类型分为三种，在刺激力度和持续时间上有差异化：

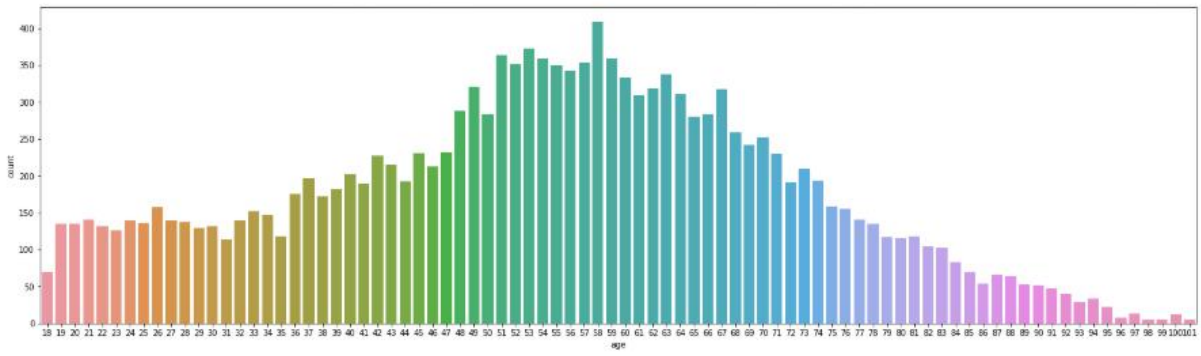
1. 买一送一：满 10 送 10，满 5 赠 5，各分 7 天和 5 天。
2. 打折：满 20 减 5（20%折扣 ）10 天，满 7 减 3 （50%折扣）7 天，满 10 减 2（20%折扣）分 10 天/7 天
3. 信息推送

数据可视化

- 性别分布

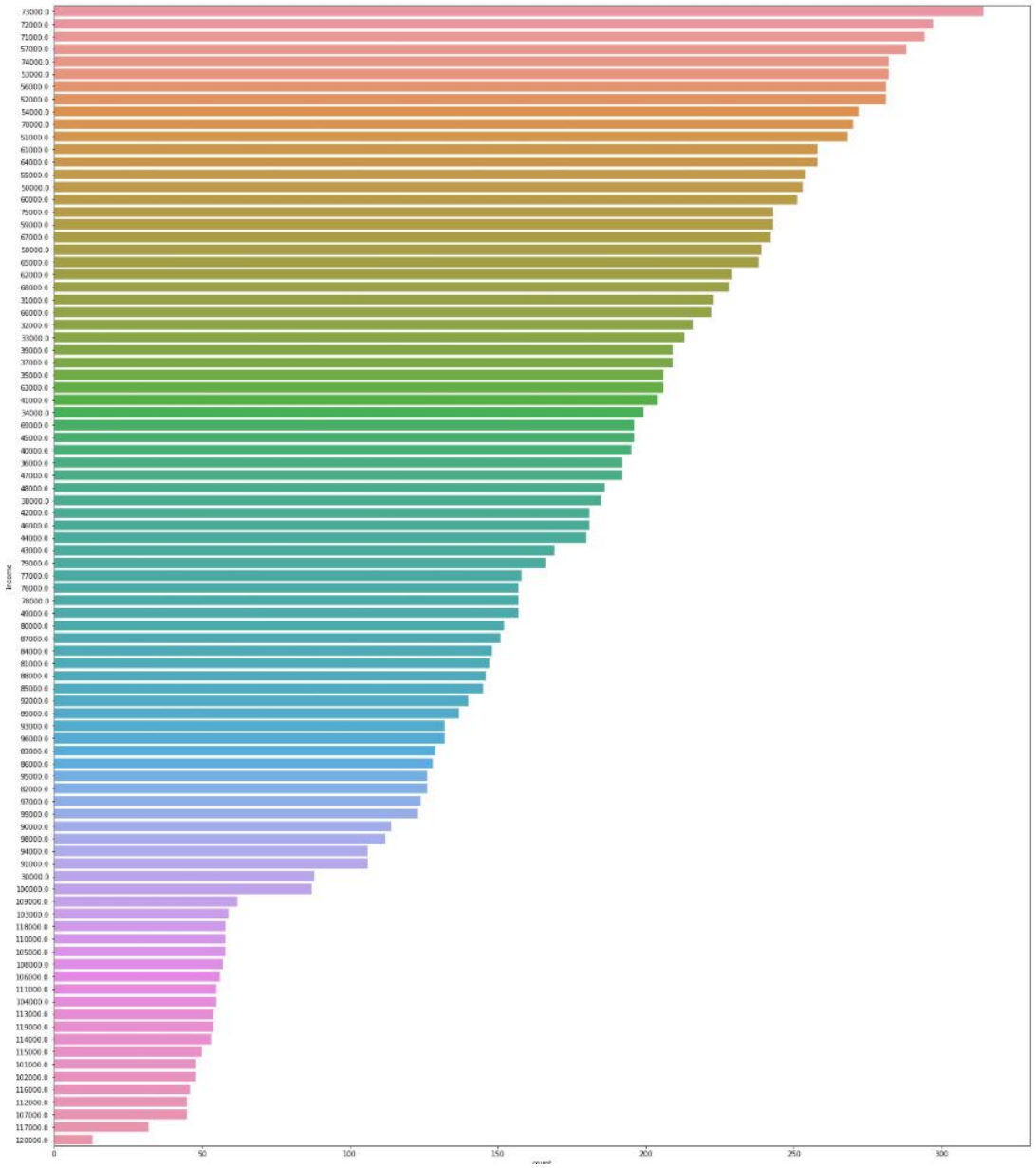


- 用户年龄分布

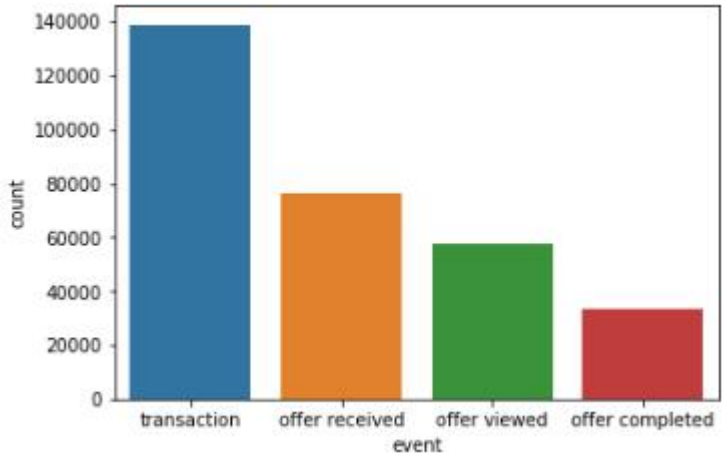


整体分布左偏正态

■ 收入分布



- 推送转化



30%的交易用到了补贴，整体的补贴率和国内外卖市场相比处于相对较低的水平，72%的推送被打开，打开的推送中 60%转化交易，考虑到总推送中 20%属于信息推送，所以实际补贴交易转化在 80%以上。

数据预处理

处理步骤：

- Transcript 表格提取 value 中的推送 id，关联推送信息，计算奖励金和消费金额
- 按用户和推送类型做聚合，计算推送数，打开数，完成单量，消费总额和补贴相关信息
- 对不同推送类型做切分，以用户为关联键做横向关联
- 新增客单价，核销率，补贴率等字段
- 剔除无推送用户
- 填充空值

	客单价	核销率	补贴率	总推送数	完成单_trans
count	16578.000000	16928.000000	16578.000000	17000.000000	17000.000000
mean	13.680266	0.549091	0.106878	4.486882	8.173706
std	16.056763	0.396682	0.109332	1.076165	5.116250
min	0.050000	0.000000	0.000000	0.000000	0.000000
25%	3.181392	0.166667	0.021605	4.000000	4.000000
50%	11.996607	0.600000	0.086294	5.000000	7.000000
75%	20.469643	1.000000	0.152650	5.000000	11.000000
max	451.470000	1.000000	1.470588	6.000000	36.000000

客单价和完成单的标准差很大，波动很明显，需要做标准化处理

代码实现

第一步：数据清洗

```
#提取推送id，奖励金和消费金额
transcript['offer_id']=[transcript.value[i]['offer id'] if 'offer id' in transcript.value[i] else transcript.value[i]['offer_i
transcript['reward']=[transcript.value[i]['reward'] if 'reward' in transcript.value[i]  else 0 for i in range(transcript['val
transcript['amount']=[transcript.value[i]['amount'] if 'amount' in transcript.value[i]  else 0 for i in range(transcript['val

#关联推送信息
new_transcript = pd.merge(transcript.loc[0:,['person','event','time','offer_id','reward','amount']],
                          portfolio,left_on='offer_id',right_on='id', how='left')
new_transcript['offer_type']=new_transcript.offer_type.fillna('trans')
```

```
a=new_transcript.groupby(['event','person','offer_type']).agg({'event':"count",'amount':'sum','reward_x':'mean','reward_y':'sum
a.columns=a.columns.droplevel()

b=a.iloc[:,[0,1,2,3,4,5,9,10,14]]
b.columns=['用户','type','使用优惠数','收到推送数','打开推送数','完成单','gmv','单均补贴','总补贴']
```

```
#重构表格
bogo=b[b.type=='bogo']
discount=b[b.type=='discount']
informational=b[b.type=='informational']
trans=b[b.type=='trans']
```

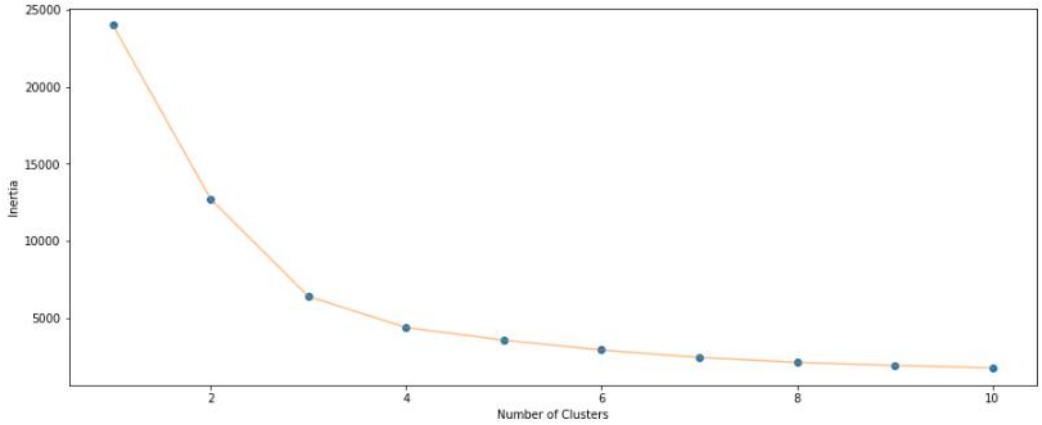
```
def rename_columns(df,prefix=str):
    new_columns=[col+'_'+prefix for col in df.columns]
    return new_columns


##重命名columns
bogo.columns=rename_columns(bogo,prefix='bogo')
discount.columns=rename_columns(discount,prefix='discount')
informational.columns=rename_columns(informational,prefix='informational')
trans.columns=rename_columns(trans,prefix='trans')

user_df = pd.merge(profile.loc[0:,['id']],bogo.loc[0:,['用户_bogo','使用优惠数_bogo','收到推送数_bogo','打开推送数_bogo','单均补
user_df = pd.merge(user_df,discount.loc[0:,['用户_discount','使用优惠数_discount','收到推送数_discount','打开推送数_discount','
user_df = pd.merge(user_df,informational.loc[0:,['用户_informational','收到推送数_informational','打开推送数_informational']],l
user_df = pd.merge(user_df,trans.loc[0:,['用户_trans','完成单_trans','gmv_trans']],left_on='id',right_on='用户_trans', how='out

final_df=user_df.drop(columns=['用户_bogo','用户_discount','用户_trans','用户_informational'])

#新增字段
final_df['客单价']=final_df['gmv_trans']/final_df['完成单_trans']
final_df['核销率']=(final_df['使用优惠数_bogo']+final_df['使用优惠数_discount'])/(final_df['收到推送数_bogo']+final_df['收到推送
final_df['补贴率']=(final_df['总补贴_bogo']+final_df['总补贴_discount'])/(final_df['gmv_trans'])
final_df['总推送数']=(final_df['收到推送数_bogo']+final_df['收到推送数_discount']+final_df['收到推送数_informational'])
final_df['打开率']=(final_df['打开推送数_bogo']+final_df['打开推送数_discount']+final_df['打开推送数_informational'])/final_df[
final_df['bogo打开率']=(final_df['打开推送数_bogo'])/final_df['收到推送数_bogo']
final_df['discount打开率']=(final_df['打开推送数_discount'])/final_df['收到推送数_discount']
final_df['informational打开率']=(final_df['打开推送数_informational'])/final_df['收到推送数_informational']

#剔除0推送人群
final_df=final_df[final_df['总推送数']>0].fillna(0)


#标准化字段用于聚类
final_df['客单价_scaled']=preprocessing.scale(final_df['客单价'])
final_df['gmv_trans_scaled']=preprocessing.scale(final_df['gmv_trans'])
final_df['总推送数_scaled']=preprocessing.scale(final_df['总推送数'])
```

第二步：Kmeans 聚类

聚类特征选取：核销率对应促销敏感度，补贴率对应补贴敏感度，gmv 对应消费能力，打开率对应触达反应。

```
X1 = final_df[['核销率','补贴率','gmv_trans_scaled','打开率']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

```
plt.figure(1 , figsize = (15 ,6))
plt.plot(np.arange(1 , 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```



根据手肘法，选取曲率最高的类别，对于这个数据集的最佳聚类数是 4
手肘法：随着聚类数 k 的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和 SSE 自然会逐渐变小，当 k 小于真实聚类数时，由于 k 的增大会大幅增加每个簇的聚合程度，故 SSE 的下降幅度会很大，而当 k 到达真实聚类数时，再增加 k 所得到的聚合程度回报会迅速变小，所以 SSE 的下降幅度会骤减，然后随着 k 值的继续增大而趋于平缓，也就是说 SSE 和 k 的关系图是一个手肘的形状，而这个肘部对应的 k 值就是数据的真实聚类数。转自 https://www.jianshu.com/p/335b376174d4

第三步：特征筛选
对用户性别特征独热编码，使用 selectKBest 进行特征筛选，此处使用卡方检验用于检验定性自变量对定性因变量的相关性。

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

selector = SelectKBest(chi2, k=5)
X_new=selector.fit_transform(predict_df.loc[0:,['总推送数','完成单_trans','打开推送数_discount','打开率','discount打开率','客单
                                                predict_df.loc[0:,['group']])
```

第四步：train/test split
训练数 80%和测试数量 20%分割做交叉验证

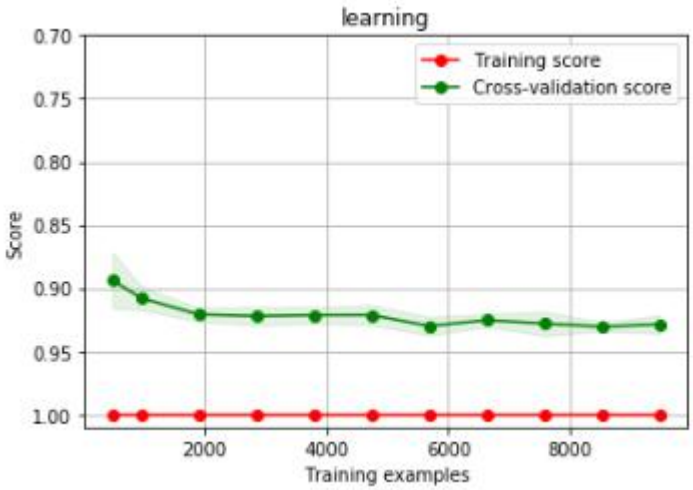
第五步：选取模型进行网格调参，并输出结果

```
class Class_Fit(object):
    def __init__(self, clf, params=None):
        if params:
            self.clf = clf(**params)
        else:
            self.clf = clf()

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def grid_search(self, parameters, Kfold):
        self.grid = GridSearchCV(estimator = self.clf, param_grid = parameters, cv = Kfold)

    def grid_fit(self, X, Y):
        self.grid.fit(X, Y)

    def grid_predict(self, X, Y):
        self.predictions = self.grid.predict(X)
        print("准确率: {:.2f} % ".format(100*metrics.accuracy_score(Y, self.predictions)))
```

```
from sklearn import neighbors, linear_model, svm, tree, ensemble
from sklearn.model_selection import GridSearchCV, learning_curve
tr = Class_Fit(clf = tree.DecisionTreeClassifier)
tr.grid_search(parameters = [{'criterion' : ['entropy', 'gini'], 'max_features' :['sqrt', 'log2']}], Kfold = 5)
tr.grid_fit(X = X_train, Y = Y_train)
tr.grid_predict(X_test, Y_test)
```

```
准确率: 92.07 %
```

第六步：检查是否过拟合



◆ 模型改进

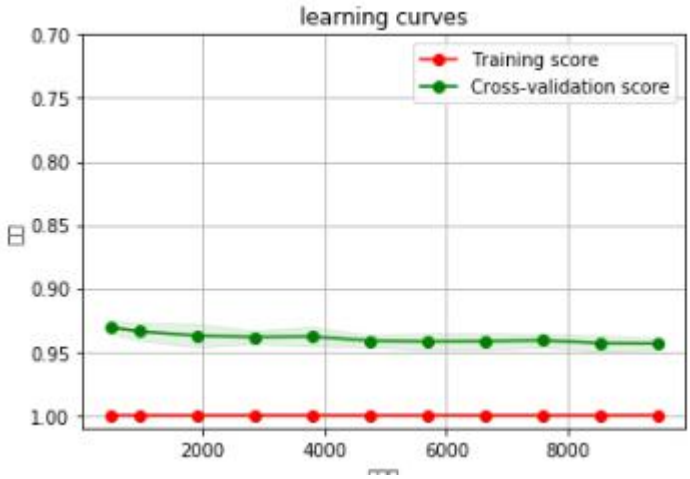
■ 树的准确率相当高，如果用集成模型也许会有更好的结果

```
rf = Class_Fit(clf = ensemble.RandomForestClassifier)
param_grid = {'criterion' : ['entropy', 'gini'], 'n_estimators' : [20, 40, 60, 80, 100],
              'max_features' :['sqrt', 'log2']}
rf.grid_search(parameters = param_grid, Kfold = 5)
rf.grid_fit(X = X_train, Y = Y_train)
rf.grid_predict(X_test, Y_test)
```

```
准确率: 93.89 %
```

集成模型提升了 1.89 个百分点

- 模型评价
  - 模型准确率 93.89%
  - 没有过拟合



- 结果讨论
  - 反思
    - 没有考虑到用户的时间属性，对于生命周期做分析和分箱处理。这个特征应该能贡献很多信息。
  - 改进