Starbucks_Capstone_notebook-zh

2020 年 4 月 2 日

1 优达学城的数据科学家毕业项目-星巴克促销效果预测

1.0.1 项目基本情况

这是我在优达学城学习数据科学家课程的毕业项目。数据科学家课程总体感觉不错，涉及了数据科学的各个方面，对新手老手，会有不同的收益，课程地址是：https://www.udacity.com/course/data-scientist-nanodegree–nd025。个人比较推荐：-）这个项目提供了星巴克的促销活动相关的真实数据，主要是三个数据文件，在 data 目录可以看到：

- 第一个文件（portfolio.json）是关于促销活动的，主要描述了一个促销活动的门槛，奖励标准等
- 第二个文件（profile.json）是客户的基本属性信息，描述了他们的性别，收入，会员创建日期等
- 第三个文件（transcript.json）是客户和星巴克的促销活动推送的互动记录

1.0.2 问题描述

这里我选择切入的一个角度是，建立一个模型，可以用来评估客户是否会对推送做出回应，哪些因子会对成功回应起关键作用。数据处理，探索和建模过程，我放在了一个 notebook 文件里，文件名为 Starbucks_Capstone_notebook-zh.ipynb。我基本上是分了三个大的步骤：

- 第一步：数据整理，主要是对三个文件的数据做些标准化，对某些类别字段做独热编码，最后并合并成一个大的文件，方便进行数据探索和建模
- 第二步：数据探索，这一步主要是探索数据，做些可视化的展现，得到一些显性的结论
- 第三步：数据建模和评估，我根据 Scikit Learn 官方的 Machine Learning Map，找出适合我们这个案例的三个分类模型，做了初步比较，然后再用网格搜索对潜力最大的一个模型进行参数遍历，验证准确率和 F beta 分数值，最后展示因子重要程度

经过三步之后，我给出结论和总结

1.0.3 衡量指标

对于分类模型，有几个重要的指标，说明如下：

准确率：这个指标评估一个模型在所有数据上的表现。在及其特殊的情况下，准确率高，不代表模型实用。以信用卡欺诈识别为例，信用卡欺诈本身是一个样本集里头非常少出现的情况，一个模型，只要对所有的样本都预测不是一个欺诈，就可以达到很高的准确率。但是这个模型并没有什么用。我们的数据没有那么特殊，准确率是一个的评估指标

精确率：这个指标主要衡量预测结果为 1 的样本，有多大的比例真实是 1，这个指标对于垃圾邮件识别很重要，这时候我们需要确保不能把正常标志为垃圾邮件，而是宁愿有垃圾邮件没有识别出来。对于促销类的推送，精确率有意义，因为我们希望模型帮我门找出真正感兴趣的消费者，以免对不感兴趣的消费者产生困扰，或者浪费促销成本（当然现在的成本在降低）

召回率：这个指标主要衡量样本中为 1 的，有多大比例被模型准确预测。在特定的场合，比如目前的疾病检测，要尽可能多的预测到正样本，哪怕有些负样本被预测为正样本，可以后续进一步甄别。这个指标对我们的例子也很有用，因为获取消费者数据不容易，要从现有的消费者数据里头尽量找到合适的消费者

F Beta Score：是精确率和召回率的平衡，是综合考虑精确率和召回率的指标，以下从维基百科中下载的图片能解释这个指标的作用。将 Beta 值设为 1（就是所谓的 F1 Score），其实就能同时反映精确率和召回率的影响，这个指标比较符合我们的需要。

1.0.4 数据集

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

以下是文件中每个变量的类型和解释：

portfolio.json * id (string) – 推送的 id * offer_type (string) – 推送的种类，例如 BOGO、打折（discount）、信息（informational）* difficulty (int) – 满足推送的要求所需的最少花费 * reward (int) – 满足推送的要求后给与的优惠 * duration (int) – 推送持续的时间，单位是天 * channels (字符串列表)

profile.json * age (int) – 顾客的年龄 * became_member_on (int) – 该顾客第一次注册 app 的时间 * gender (str) – 顾客的性别（注意除了表示男性的 M 和表示女性的 F 之外，还有表示其他的 O）* id (str) – 顾客 id * income (float) – 顾客的收入

transcript.json * event (str) – 记录的描述（比如交易记录、推送已收到、推送已阅）* person (str) – 顾客 id * time (int) – 单位是小时，测试开始时计时。该数据从时间点 t=0 开始 * value - (dict of strings) – 推送的 id 或者交易的数额

[1]:
```
# 基本的库导入
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image

# 以下是 Scikit-Learn 的库导入
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.
 ↪html#sklearn.svm.LinearSVC
from sklearn.svm import LinearSVC

# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.
 ↪KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

# https://scikit-learn.org/stable/modules/ensemble.
 ↪html#forests-of-randomized-trees
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import fbeta_score, make_scorer

# 用于 SQLite 处理
from sqlalchemy import create_engine

%matplotlib inline
```

[2]:
```
# 文件读取
portfolio = pd.read_json("data/portfolio.json", orient="records", lines=True)
profile = pd.read_json("data/profile.json", orient="records", lines=True)
transcript = pd.read_json("data/transcript.json", orient="records", lines=True)
```

1.0.5 第一步，数据整理

1.1 先看看三个文件的数据结构及其字段内容

[3]:
```
# 三个数据集的大小先有个认识
portfolio.shape, profile.shape, transcript.shape
```

[3]:
```
((10, 6), (17000, 5), (306534, 4))
```

[4]:
```
portfolio.head()
```

[4]:

	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	[web, email]	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5

[5]:
```
# channels，是个类别数据，需要 one-hot 处理
print(portfolio["channels"].value_counts())
print("\n-------我是一条完美的分割线-------\n")
# offer_type 是个类别数据，需要 one-hot 处理
print(portfolio["offer_type"].value_counts())
```

```
[web, email, mobile, social]    4
[web, email, mobile]            3
[email, mobile, social]         2
[web, email]                    1
```

```
Name: channels, dtype: int64

-------我是一条完美的分割线-------

bogo             4
discount         4
informational    2
Name: offer_type, dtype: int64
```

[6]:
```
profile.head()
```

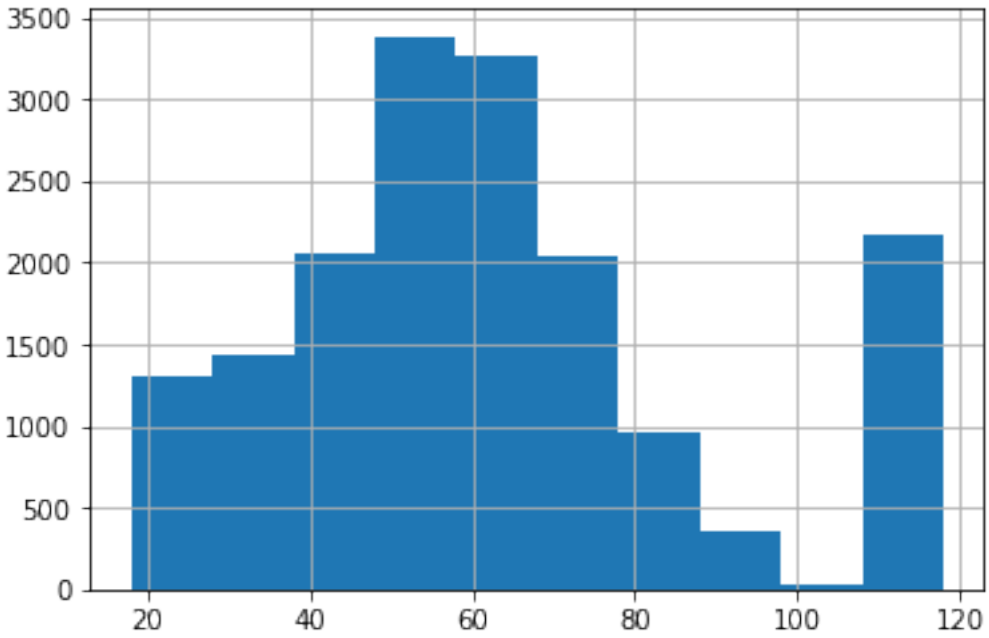
[6]:

	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN

[7]:
```
# 100 岁以上的消费者竟然 有 2180 个，有点怪怪的
print(profile[profile["age"] > 100].shape, profile.shape)
profile["age"].hist()
```

```
(2180, 5) (17000, 5)
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29a57335438>
```



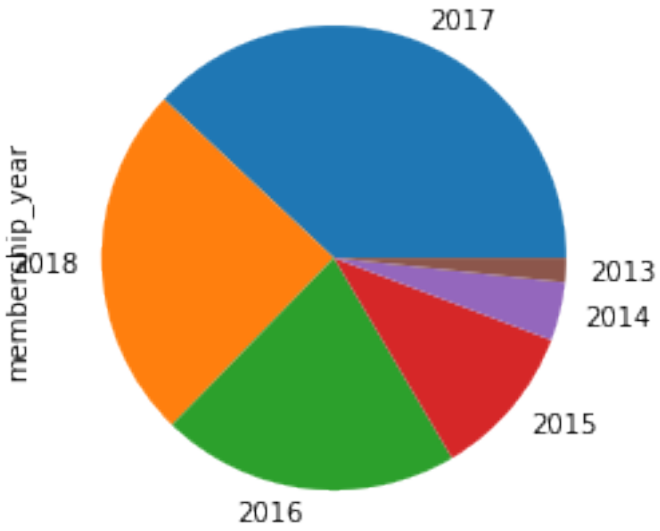
[8]:
```
# 不过也不要紧，由于性别和收入字段为空，只有 5 位 100 岁以上的消费者会保留
profile[profile["age"] > 100].sort_values(["gender", "income"]).head(10)
```

[8]:

	age	became_member_on	gender	id	income
1556	101	20171004	F	4d2ccfcbbebf4bd9baf4b7e433d0e288	43000.0
14846	101	20171109	F	e0ea90ddd2f147e082d21e97f51ec1b1	56000.0
15800	101	20170309	F	047ad0135cfe4c0ea5ba019da4de9c52	59000.0
16864	101	20171127	F	1593d617fac246ef8e50dbb0ffd77f5f	82000.0
4100	101	20150526	F	d2fdc2be8ab64e4ba04830d441e53fd5	99000.0
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN

[9]:
```
# became_member_on 是个字符串，把年份取出来看看
# 新会员数量比老会员多很多
profile["membership_year"] = pd.to_datetime(profile["became_member_on"],␣
 ↪format="%Y%m%d").dt.year
profile["membership_year"].value_counts().plot.pie()
```

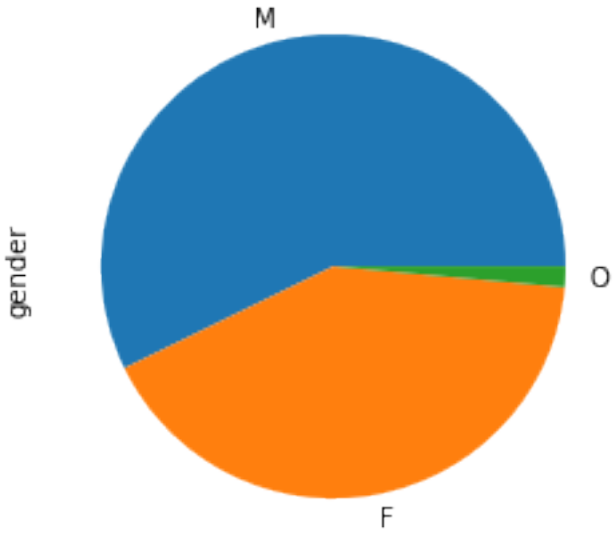
[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29a5736b898>
```



[10]:
```
# 性别有 212 条"O" 的记录，不算特别多，删除，这一列就两个值，做个 one-hot
print(profile["gender"].value_counts())
profile["gender"].value_counts().plot.pie()
```

```
M    8484
F    6129
O     212
Name: gender, dtype: int64
```

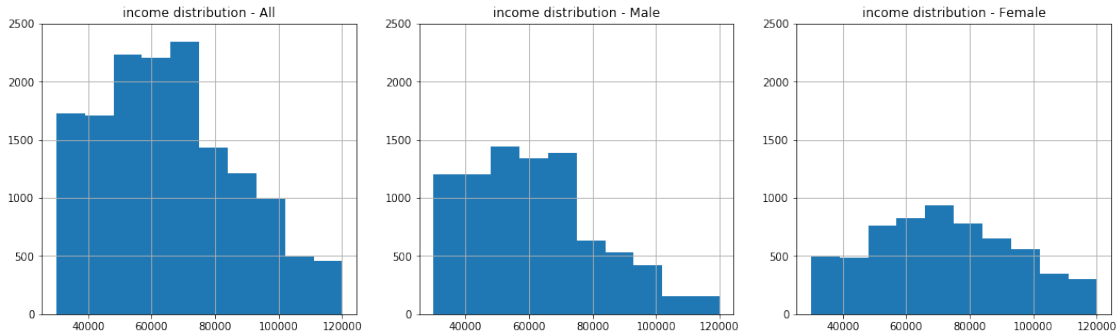
[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29a57391dd8>
```



[11]:
```
# 收入为空的记录有点多.....不过经过之前的分析，发现都是些 100 岁以上没有收入的“老人”
# 其实我觉得是消费者故意留下的脏数据或者是星巴克收集数据的过程中有些小问题
print(profile.shape, profile[profile["income"].isna()].shape)
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# 收入和性别有关系吗？-- 男性高收入的还少点.....
axes[0].set_title("income distribution - All")
axes[0].set_ylim(top=2500)
axes[1].set_title("income distribution - Male")
axes[1].set_ylim(top=2500)
axes[2].set_title("income distribution - Female")
axes[2].set_ylim(top=2500)

profile["income"].hist(ax=axes[0])
profile[profile["gender"]=="M"]["income"].hist(ax=axes[1])
profile[profile["gender"]=="F"]["income"].hist(ax=axes[2])
```

```
(17000, 6) (2175, 6)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29a573e8da0>
```



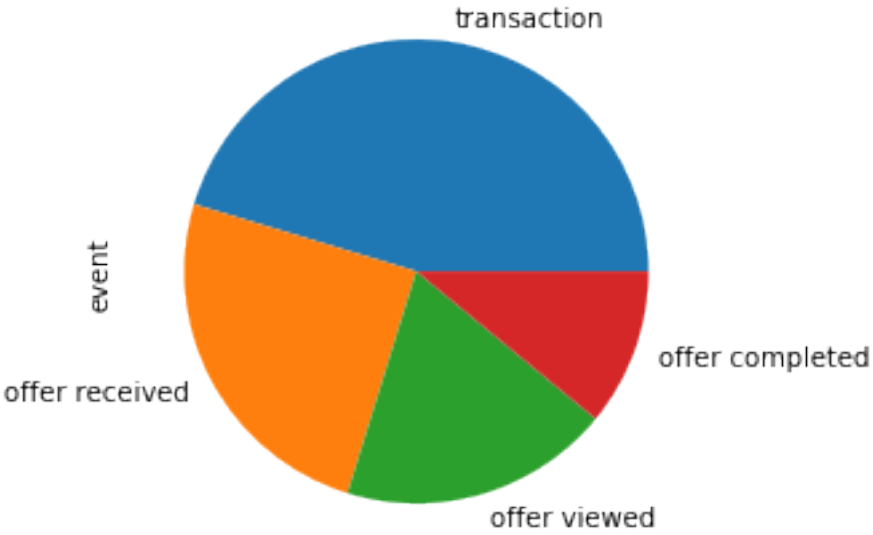
[12]:
```
transcript.head()
```

[12]:

	event	person	time	value
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}

[13]:
```
# Event 列只有四个类别，实际上这个 trascript 数据集，混合了推送的互动信息和消费者的消费记录，应该分开成为四个数据集
transcript["event"].value_counts().plot.pie()
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29a57458c88>
```



[14]:
```
# 对于 transaction 类型记录，value 里头是消费金额
transcript[transcript["event"] == "transaction"].head()
```

[14]:
```
             event                            person  time  \
12654  transaction  02c083884c7d45b39cc68e1314fec56c     0
12657  transaction  9fa9ae8f57894cc9a3b8a9bbe0fc1b2f     0
12659  transaction  54890f68699049c2a04d415abc25e717     0
12670  transaction  b2f1cd155b864803ad8334cdf13c4bd2     0
12671  transaction  fe97aa22dd3e48c8b143116a8403dd52     0

                           value
12654  {'amount': 0.8300000000000001}
12657               {'amount': 34.56}
12659               {'amount': 13.23}
12670               {'amount': 19.51}
12671               {'amount': 18.97}
```

[15]:
```
# 推送的送达记录
transcript[transcript["event"] == "offer received"].head()
```

[15]:
```
             event                            person  time  \
0  offer received  78afa995795e4d85b5d9ceeca43f5fef     0
```

```
1  offer received  a03223e636434f42ac4c3df47e8bac43     0
2  offer received  e2127556f4f64592b11af22de27a7932     0
3  offer received  8ec6ce2a7e7949b1bf142def7d0e0586     0
4  offer received  68617ca6246f4fbc85e91a2a49552598     0

                                                 value
0  {'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2  {'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3  {'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4  {'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}
```

[16]:
```
# 推送的浏览记录
transcript[transcript["event"] == "offer viewed"].head()
```

[16]:
```
              event                            person  time  \
12650  offer viewed  389bc3fa690240e798340f5a15918d5c     0
12651  offer viewed  d1ede868e29245ea91818a903fec04c6     0
12652  offer viewed  102e9454054946fda62242d2e176fdce     0
12653  offer viewed  02c083884c7d45b39cc68e1314fec56c     0
12655  offer viewed  be8a5d1981a2458d90b255ddc7e0d174     0

                                                 value
12650  {'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}
12651  {'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
12652  {'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}
12653  {'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}
12655  {'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
```

[17]:
```
# 推送的完成记录
transcript[transcript["event"] == "offer completed"].head()
```

[17]:
```
                event                            person  time  \
12658  offer completed  9fa9ae8f57894cc9a3b8a9bbe0fc1b2f     0
12672  offer completed  fe97aa22dd3e48c8b143116a8403dd52     0
12679  offer completed  629fc02d56414d91bca360decdfa9288     0
12692  offer completed  676506bad68e4161b9bbaffeb039626b     0
12697  offer completed  8f7dd3b2afe14c078eb4f6e6fe4ba97d     0
```

```
                                                    value
12658  {'offer_id': '2906b810c7d4411798c6938adc9daaa5...
12672  {'offer_id': 'fafdcd668e3743c1bb461111dcafc2a4...
12679  {'offer_id': '9b98b8c7a33c4b65b9aebfe6a799e6d9...
12692  {'offer_id': 'ae264e3637204a6fb9bb56bc8210ddfd...
12697  {'offer_id': '4d5c57ea9a6940dd891ad53e9dbe8da0...
```

总结一下理解到的关键信息

- portfolio 表的 channel 字段，需要根据里头的取值，处理成独立的字段，不然无法分析
- portfolio 表的 id 列，重新命名成 offer_id，方便 table join
- portfolio 表的 offer_type，需要做独热编码
- –
- profile 表的 age 字段，后续不好处理，根据直方图的观察，分成几个类做独热编码：青少年（20 岁以下），青年（20 岁到 40 岁），中年（40 岁到 60 岁），中老年（60 岁到 80 岁）和老年（80 岁以上）
- profile 表的 became_member_on 是个字符串，进行统计还不好搞，做个日期字段，然后年做 one-hot（月份有点多，暂时先不做 one-hot）
- profile 表的性别列，有 None 值，应该是不合理数据，删除掉相关记录，同时做 one-hot
- profile 表的 id 列，重新命名成 consumer_id，方便 table join
- profile 表中 income 列有空值，删除相关记录
- –
- transcript 需要拆分成四个数据集，三个是 offer 互动，一个是 consumer_transaction，用 event 的值区分
- transcript 的 value 列，包含 offer_id 或者消费金额，这个字段需要处理成一个新的列
- transcript 的 person 列，包含的是 profile id，用来连接 profile 的外键，改个列名（consumer_id）方便后续处理

1.2 完成数据清理和数据合并

- 1.2.1 清理 profolio 数据集

[18]:
```
# 再瞄一眼 portfolio 数据集
portfolio.head()
```

[18]:
```
                      channels  difficulty  duration  \
0       [email, mobile, social]          10         7
1  [web, email, mobile, social]          10         5
2          [web, email, mobile]           0         4
3          [web, email, mobile]           5         7
4                   [web, email]          20        10

                                 id     offer_type  reward
0  ae264e3637204a6fb9bb56bc8210ddfd           bogo      10
1  4d5c57ea9a6940dd891ad53e9dbe8da0           bogo      10
2  3f207df678b143eea3cee63160fa8bed  informational       0
3  9b98b8c7a33c4b65b9aebfe6a799e6d9           bogo       5
4  0b1e1539f2cc45b7b9fa7c272da2e1d7       discount       5
```

[19]:
```
# 处理 channel 字段
mlb = MultiLabelBinarizer()
mlb.fit(portfolio['channels'])
df_channels = pd.DataFrame(data=mlb.transform(portfolio['channels']),␣
 ↪columns=mlb.classes_)
df_channels
```

[19]:
```
   email  mobile  social  web
0      1       1       1    0
1      1       1       1    1
2      1       1       0    1
3      1       1       0    1
4      1       0       0    1
5      1       1       1    1
6      1       1       1    1
7      1       1       1    0
8      1       1       1    1
9      1       1       0    1
```

[20]:
```
# one-hot for offer type
df_offer_type = pd.get_dummies(portfolio["offer_type"])
```

```
df_offer_type
```

[20]:

	bogo	discount	informational
0	1	0	0
1	1	0	0
2	0	0	1
3	1	0	0
4	0	1	0
5	0	1	0
6	0	1	0
7	0	0	1
8	1	0	0
9	0	1	0

[21]:
```
# 完成 portfolio 数据集清理
portfolio = pd.concat([portfolio, df_channels, df_offer_type], axis=1)

# 到后面发现留着原始数据，方便分析
# portfolio.drop(["channels", "offer_type"], inplace=True, axis=1)
new_columns = ["offer_id" if col== "id" else col for col in portfolio.columns]
portfolio.columns = new_columns
portfolio.head()
```

[21]:

	channels	difficulty	duration	\
0	[email, mobile, social]	10	7	
1	[web, email, mobile, social]	10	5	
2	[web, email, mobile]	0	4	
3	[web, email, mobile]	5	7	
4	[web, email]	20	10	

	offer_id	offer_type	reward	email	mobile	\
0	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10	1	1	
1	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10	1	1	
2	3f207df678b143eea3cee63160fa8bed	informational	0	1	1	
3	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5	1	1	
4	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5	1	0	

	social	web	bogo	discount	informational

```
0       1    0     1         0              0
1       1    1     1         0              0
2       0    1     0         0              1
3       0    1     1         0              0
4       0    1     0         1              0
```

- 1.2.2 清理 profile 数据集

[22]:
```
# 再瞄一眼 profile 数据集
profile.head()
```

[22]:
```
   age  became_member_on gender                                id    income  \
0  118          20170212   None  68be06ca386d4c31939f3a4f0e3dd783       NaN
1   55          20170715      F  0610b486422d4921ae7d2bf64640c50b  112000.0
2  118          20180712   None  38fe809add3b4fcf9315a9694bb96ff5       NaN
3   75          20170509      F  78afa995795e4d85b5d9ceeca43f5fef  100000.0
4  118          20170804   None  a03223e636434f42ac4c3df47e8bac43       NaN

   membership_year
0             2017
1             2017
2             2018
3             2017
4             2017
```

[23]:
```
# 删除空值的行和性别等于“O”的
profile = profile[profile["gender"].notna()]
profile = profile[profile["income"].notna()]
profile = profile[profile["gender"]!="O"]
```

[24]:
```
# 处理一下年龄，做个范围的 one-hot
profile["age_range"] = pd.cut(profile["age"], [0, 20, 40, 60, 80,␣
 ↪profile["age"].max()])
df_age = pd.get_dummies(profile["age_range"])
df_age.head()
```

[24]:
```
    (0, 20]  (20, 40]  (40, 60]  (60, 80]  (80, 101]
1         0         0         1         0          0
3         0         0         0         1          0
```

5	0	0	0	1	0
8	0	0	0	1	0
12	0	0	1	0	0

[25]:
```
# 做一个注册年份的 one-hot
df_year = pd.get_dummies(profile["membership_year"])
df_year.head()
```

[25]:

	2013	2014	2015	2016	2017	2018
1	0	0	0	0	1	0
3	0	0	0	0	1	0
5	0	0	0	0	0	1
8	0	0	0	0	0	1
12	0	0	0	0	1	0

[26]:
```
# 性别列做成 one-hot
profile["gender"].replace("M", "Male", inplace=True)
profile["gender"].replace("F", "Female", inplace=True)
df_gender = pd.get_dummies(profile["gender"])
df_gender.head()
```

[26]:

	Female	Male
1	1	0
3	1	0
5	0	1
8	0	1
12	0	1

[27]:
```
# 完成 profile 数据集清理
profile = pd.concat([profile, df_year, df_age, df_gender], axis=1)

# profile.drop(["became_member_on", "gender", "age"], axis=1, inplace=True)
new_columns = ["consumer_id" if col== "id" else col for col in profile.columns]
profile.columns = new_columns
print(profile.shape)
profile.head()
```

```
(14613, 20)
```

[27]:

	age	became_member_on	gender	consumer_id	income	membership_year	age_range	2013	2014	2015	2016	2017	2018	(0, 20]	(20, 40]	(40, 60]	(60, 80]	(80, 101]	Female	Male
1	55	20170715	Female	0610b486422d4921ae7d2bf64640c50b	112000.0	2017	(40, 60]	0	0	0	0	1	0	0	0	1	0	0	1	0
3	75	20170509	Female	78afa995795e4d85b5d9ceeca43f5fef	100000.0	2017	(60, 80]	0	0	0	0	1	0	0	0	0	1	0	1	0
5	68	20180426	Male	e2127556f4f64592b11af22de27a7932	70000.0	2018	(60, 80]	0	0	0	0	0	1	0	0	0	1	0	0	1
8	65	20180209	Male	389bc3fa690240e798340f5a15918d5c	53000.0	2018	(60, 80]	0	0	0	0	0	1	0	0	0	1	0	0	1
12	58	20171111	Male	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	2017	(40, 60]	0	0	0	0	1	0	0	0	1	0	0	0	1

- 1.2.3 清理 transcript 数据集

[28]:

```
# 再瞄一眼 transcript 数据集
transcript.head()
```

[28]:

	event	person	time	value
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}

```
3  {'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4  {'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}
```

[29]:
```
# 列名 person 先改为 consumer_id
new_columns = ["consumer_id" if col== "person" else col for col in transcript.
 ↪columns]
transcript.columns = new_columns

# 因为清理了一些 profile 记录，先把 transcript 里头的也相应清理一下
transcript = transcript[transcript["consumer_id"].isin(profile["consumer_id"])]
```

[30]:
```
# transcript 先分成两个数据集
transaction = transcript[transcript["event"]=="transaction"].copy()
offer = transcript[transcript["event"]!="transaction"].copy()

# 对于 transation 数据集，需要生成金额，用 apply 函数实现
transaction["amount"] = transaction['value'].apply(lambda x: x["amount"])

# 对于 offer 数据集，需要取出"offer id"，"offer id" 中间有空格，总提示 key error，
只能调用 values 函数绕过去
offer["offer_id"] = offer["value"].apply(lambda x: list(x.values())[0])

# 对于 offer 数据集，event 有三个类别，做成独立的表
offer_received = offer[offer["event"] == "offer received"].copy()
offer_viewed = offer[offer["event"] == "offer viewed"].copy()
offer_completed = offer[offer["event"] == "offer completed"].copy()

# 删除不必要字段
transaction.drop(["event", "value"], axis=1, inplace=True)
offer_received.drop(["event", "value"], axis=1, inplace=True)
offer_viewed.drop(["event", "value"], axis=1, inplace=True)
offer_completed.drop(["event", "value"], axis=1, inplace=True)
```

[31]:
```
# 交易信息，由消费者，时间和消费金额信息
# 我的项目不涉及消费金额预测，有时间我在做些各维度的消费者画像
# 这部分数据暂时不用
print(transaction.shape)
```

```
transaction.head()
```

```
(122176, 3)
```

[31]:

	consumer_id	time	amount
12654	02c083884c7d45b39cc68e1314fec56c	0	0.83
12657	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	0	34.56
12659	54890f68699049c2a04d415abc25e717	0	13.23
12670	b2f1cd155b864803ad8334cdf13c4bd2	0	19.51
12671	fe97aa22dd3e48c8b143116a8403dd52	0	18.97

[32]:

```
# 这是推送记录
print(offer_received.shape)
offer_received.head()
```

```
(65585, 3)
```

[32]:

	consumer_id	time	offer_id
0	78afa995795e4d85b5d9ceeca43f5fef	0	9b98b8c7a33c4b65b9aebfe6a799e6d9
2	e2127556f4f64592b11af22de27a7932	0	2906b810c7d4411798c6938adc9daaa5
5	389bc3fa690240e798340f5a15918d5c	0	f19421c1d4aa40978ebb69ca19b0e20d
7	2eeac8d8feae4a8cad5a6af0499a211d	0	3f207df678b143eea3cee63160fa8bed
8	aa4862eba776480b8bb9c68455b8c2e1	0	0b1e1539f2cc45b7b9fa7c272da2e1d7

[33]:

```
# 这是推送的浏览记录
print(offer_viewed.shape)
offer_viewed.head()
```

```
(49087, 3)
```

[33]:

	consumer_id	time	\
12650	389bc3fa690240e798340f5a15918d5c	0	
12652	102e9454054946fda62242d2e176fdce	0	
12653	02c083884c7d45b39cc68e1314fec56c	0	
12655	be8a5d1981a2458d90b255ddc7e0d174	0	
12656	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	0	

	offer_id

```
12650   f19421c1d4aa40978ebb69ca19b0e20d
12652   4d5c57ea9a6940dd891ad53e9dbe8da0
12653   ae264e3637204a6fb9bb56bc8210ddfd
12655   5a8bc65990b245e5a138643cd4eb9837
12656   2906b810c7d4411798c6938adc9daaa5
```

[34]:
```
# 这是推送的完成记录
print(offer_completed.shape)
offer_completed.head()
```

```
(31943, 3)
```

[34]:
```
                             consumer_id  time  \
12658  9fa9ae8f57894cc9a3b8a9bbe0fc1b2f     0
12672  fe97aa22dd3e48c8b143116a8403dd52     0
12679  629fc02d56414d91bca360decdfa9288     0
12692  676506bad68e4161b9bbaffeb039626b     0
12697  8f7dd3b2afe14c078eb4f6e6fe4ba97d     0

                                 offer_id
12658  2906b810c7d4411798c6938adc9daaa5
12672  fafdcd668e3743c1bb461111dcafc2a4
12679  9b98b8c7a33c4b65b9aebfe6a799e6d9
12692  ae264e3637204a6fb9bb56bc8210ddfd
12697  4d5c57ea9a6940dd891ad53e9dbe8da0
```

1.2.4 基于 offer_received 表，生成数据宽表，并且生成必要的判断字段

- 一个推送对于不同的消费者，有不同的起止 time 值
- 一个活动还可以触达用户多次
- 起始的 time，是 offer received 里头的 time
- 终止的 time，是起始的 time 加上 offer 主数据里头的 duration （天数 X 24 小时）

这里需要找出成功的推送，所谓成功的推送，是那些：* 推送后，在有效日期内有浏览记录和完成记录 * 完成的时间是不早于浏览时间

[35]:
```
# 对于每次推送，连接到 portfolio 数据集，计算一个截至时间（单位：小时）
offer_received = pd.merge(offer_received, portfolio[["offer_id", "duration"]],␣
↪on="offer_id")
```

```
offer_received["end_time"] = offer_received["time"] +␣
 ↪offer_received["duration"] * 24
```

[36]:
```
# 接下来，由于需要用到“不等于”类型表连接，我借助一下 SQL 来完成，免得写复杂的循环函数，逐条循环处理 offer_received 数据集
engine = create_engine("sqlite:///offer.db" )

offer_received.to_sql("offer_received", engine, index=False,␣
 ↪if_exists="replace")
offer_viewed.to_sql("offer_viewed", engine, index=False, if_exists="replace")
offer_completed.to_sql("offer_completed", engine, index=False,␣
 ↪if_exists="replace")
```

[37]:
```
# 更高效的语句，用 Python 试了写个 supporting functon，感觉挺难维护的，放弃了
# 主表是 offer_received，这里的逻辑是关键，后续数据分析，主要基于这里的匹配逻辑

# 对于每一条推送，查找出有效期内的浏览和完成的记录及其时间
# 如果两个都存在，且完成是在浏览之后，就是一条 valid 的推送
# 65585 个推送中，总共过滤出 22492 个有效的推送

SQL = " \
SELECT r.consumer_id, r.offer_id, r.time, r.end_time, min(v.time) as␣
 ↪viewed_time, max(c.time) as completed_time \
FROM offer_received r \
LEFT JOIN offer_viewed v \
ON (v.consumer_id = r.consumer_id \
    AND v.offer_id = r.offer_id \
    AND v.time >= r.time \
    AND v.time <= r.end_time) \
LEFT JOIN offer_completed c \
ON (c.consumer_id = r.consumer_id \
    AND c.offer_id = r.offer_id \
    AND c.time >= r.time \
    AND c.time <= r.end_time) \
GROUP BY r.consumer_id, r.offer_id, r.time, r.end_time \
"
offer_full = pd.read_sql(SQL, engine)
```

[38]:
```
# 建一个新字段，推送互动成功，值为 1，不成功，值为 0
offer_full["valid_ind"] = 0
offer_full.loc[offer_full[offer_full["completed_time"] >=␣
 ↪offer_full["viewed_time"]].index, "valid_ind"] = 1
print(offer_full.shape, offer_full[offer_full["valid_ind"]==1].shape)
offer_full.sample(10)
```

```
(65585, 7) (22492, 7)
```

[38]:

	consumer_id	offer_id	time	end_time	viewed_time	completed_time	valid_ind
57483	e00e39e41bf64361a81f0c82da3c3ead	2906b810c7d4411798c6938adc9daaa5	168	336	NaN	228.0	0
64415	fb49cd007a49481c939453026dd017e4	fafdcd668e3743c1bb461111dcafc2a4	576	816	582.0	600.0	1
51221	c7456a584a65425188c13311390e53c5	ae264e3637204a6fb9bb56bc8210ddfd	168	336	180.0	NaN	0
47031	b5d0c4015cb44faa823e5402fa7b7560	4d5c57ea9a6940dd891ad53e9dbe8da0	0	120	0.0	NaN	0
31556	7b83fb0794de40ea850c6c6653fcf244	3f207df678b143eea3cee63160fa8bed	0	96	NaN	NaN	0
5032	141e082d59ef48759823ab1ddcfe1f7a	fafdcd668e3743c1bb461111dcafc2a4	504	744	552.0	558.0	1
21355	533c24cdc0904ea1955909565ae507a9	3f207df678b143eea3cee63160fa8bed	408	504	NaN	NaN	0
27157	6a4066d776b64dde8c0444d2f1c14f5f	2298d6c36e964ae4a3e7e9706d1fb8c2	168	336	198.0	264.0	1
27830	6cdaca9000bc4408921453abfafcc6d2	5a8bc65990b245e5a138643cd4eb9837	408	480	408.0	NaN	0
43656	a8fcc392843343428db13c0814f7d5b3	3f207df678b143eea3cee63160fa8bed	408	504	NaN	NaN	0

[39]:
```
# 完成最后的表连接
df_clean = pd.merge(offer_full, profile, on="consumer_id")
df_clean = pd.merge(df_clean, portfolio, on="offer_id")
print(df_clean.shape)
```

```
df_clean.sample(10)
```

```
(65585, 38)
```

[39]:

	consumer_id	offer_id	\
59065	043bcfeacb874bbc837300701ce25870	0b1e1539f2cc45b7b9fa7c272da2e1d7	
62247	7ca8a3cfaa7a4628a9523502c8af804f	0b1e1539f2cc45b7b9fa7c272da2e1d7	
19706	02dd040b77914163b1fd02efc3976d55	f19421c1d4aa40978ebb69ca19b0e20d	
17586	b0628933a9154b31ad6293100c717127	5a8bc65990b245e5a138643cd4eb9837	
60636	402643d64e3941c6acd9b00798a8c07d	0b1e1539f2cc45b7b9fa7c272da2e1d7	
46910	2d4bd960ee8045a7b79327ba3225b395	2298d6c36e964ae4a3e7e9706d1fb8c2	
7125	18929cd7445c4604bb987f4b0cbb036c	3f207df678b143eea3cee63160fa8bed	
32046	e675e1fc77b04d839bf84487fef0bcbb	fafdcd668e3743c1bb461111dcafc2a4	
54512	53e0f764b9454fa2bf382703341848ba	9b98b8c7a33c4b65b9aebfe6a799e6d9	
20107	1261cf51692541a5bdf39f0aa36b65f7	f19421c1d4aa40978ebb69ca19b0e20d	

	time	end_time	viewed_time	completed_time	valid_ind	age	\
59065	336	576	NaN	336.0	0	70	
62247	336	576	NaN	414.0	0	78	
19706	0	120	0.0	6.0	1	37	
17586	408	480	414.0	NaN	0	69	
60636	0	240	NaN	NaN	0	50	
46910	504	672	522.0	528.0	1	60	
7125	336	432	342.0	NaN	0	66	
32046	576	816	594.0	612.0	1	56	
54512	576	744	588.0	588.0	1	37	
20107	408	528	408.0	414.0	1	54	

	became_member_on	gender	...	duration	offer_type	reward	email	\
59065	20171129	Female	...	10	discount	5	1	
62247	20170805	Female	...	10	discount	5	1	
19706	20171122	Male	...	5	bogo	5	1	
17586	20180429	Male	...	3	informational	0	1	
60636	20160327	Male	...	10	discount	5	1	
46910	20150208	Male	...	7	discount	3	1	
7125	20160718	Female	...	4	informational	0	1	
32046	20171213	Male	...	10	discount	2	1	

```
54512          20170816  Female  ...         7          bogo      5      1
20107          20160707    Male  ...         5          bogo      5      1

       mobile  social  web  bogo  discount  informational
59065       0       0    1     0         1              0
62247       0       0    1     0         1              0
19706       1       1    1     1         0              0
17586       1       1    0     0         0              1
60636       0       0    1     0         1              0
46910       1       1    1     0         1              0
7125        1       0    1     0         0              1
32046       1       1    1     0         1              0
54512       1       0    1     1         0              0
20107       1       1    1     1         0              0

[10 rows x 38 columns]
```

[40]:
```
# 测试代码，用来抽查 SQL 返回的数据匹配情况
#␣
 ↪offer_received[offer_received["consumer_id"]=="d7d5dc9730ff479c840e10eaa190e8bc"].
 ↪sort_values("time")
# offer_viewed[offer_viewed["consumer_id"]=="d7d5dc9730ff479c840e10eaa190e8bc"].
 ↪sort_values("time")
#␣
 ↪offer_completed[offer_completed["consumer_id"]=="d7d5dc9730ff479c840e10eaa190e8bc"].
 ↪sort_values("time")
```

1.0.6 第二步，数据探索

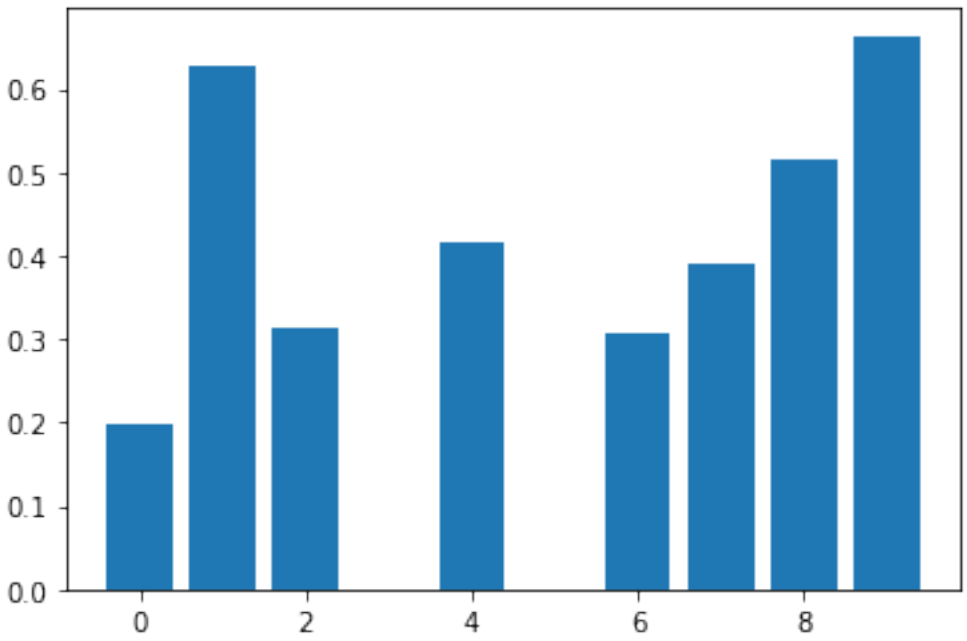
现在我们有了大量的数据，可以从不同的维度取观察一个 offer 成功的影响因素，看哪些因素，对影响推送的成功率，尤其重点应该关注消费者的属性，比如收入，年龄，性别和入会时间长短。这些都是常用到的分析维度

[41]:
```
# 我们首先看看整体 9 个 offer 的成功率的情况
g_obj = df_clean.groupby(by=["offer_id"])
df_info = g_obj["valid_ind"].agg(['count', 'sum']).reset_index()
df_info["success_rate"] = df_info["sum"]/df_info["count"]
```

```
# 明显有两个 offer 有问题，difficulty 是 0，也没有成功的参与，感觉是没有开放出来的 offer
plt.bar(df_info.index, df_info["success_rate"])
df_info
```

[41]:

	offer_id	count	sum	success_rate
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	6627	1321	0.199336
1	2298d6c36e964ae4a3e7e9706d1fb8c2	6563	4112	0.626543
2	2906b810c7d4411798c6938adc9daaa5	6543	2049	0.313159
3	3f207df678b143eea3cee63160fa8bed	6561	0	0.000000
4	4d5c57ea9a6940dd891ad53e9dbe8da0	6521	2719	0.416961
5	5a8bc65990b245e5a138643cd4eb9837	6544	0	0.000000
6	9b98b8c7a33c4b65b9aebfe6a799e6d9	6584	2019	0.306652
7	ae264e3637204a6fb9bb56bc8210ddfd	6590	2579	0.391351
8	f19421c1d4aa40978ebb69ca19b0e20d	6488	3336	0.514180
9	fafdcd668e3743c1bb461111dcafc2a4	6564	4357	0.663772



[42]:

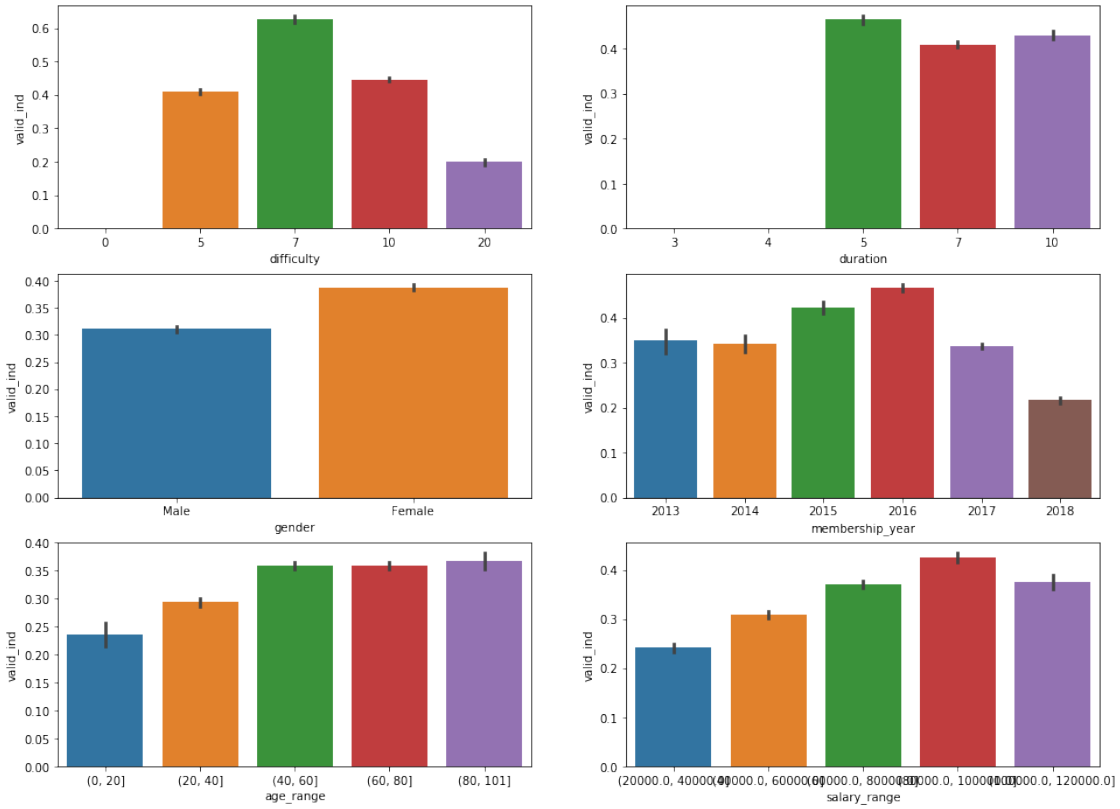
```
# 可视化函数，方便反复调用
def visualize_output(offer_id):
    """
```

```
    PURPOSE: 对于输入的推送 ID，对数据从六个维度分组，输出子图
    INPUT:
        - offer_id: 推送 ID

    OUTPUT:
        - 图形输出，无返回值

    """
    if offer_id.strip()!= "":
        print("正在查看 offer_id 为: {}的数据集。".format(offer_id))
        df_subset = df_clean[df_clean["offer_id"]==offer_id].copy()
    else:
        print("正在查看整个数据集。")
        df_subset = df_clean

    # 我们着重从推送设置和消费者属性上去观察活动的情况
    fig, axes = plt.subplots(3, 2, figsize=(16, 12))

    # 推送的设置本身也有可能对最终结果有影响
    sns.barplot(x="difficulty", y="valid_ind", data=df_subset, ax=axes[0][0])
    sns.barplot(x="duration", y="valid_ind", data=df_subset, ax=axes[0][1])

    # 女性对优惠更敏感
    sns.barplot(x="gender", y="valid_ind", data=df_subset, ax=axes[1][0],␣
↪order=["Male", "Female"])

    # 新会员，参与度稍低
    sns.barplot(x="membership_year", y="valid_ind", data=df_subset,␣
↪ax=axes[1][1])

    # 年轻群体， 对优惠不太敏感
    sns.barplot(x="age_range", y="valid_ind", data=df_subset, ax=axes[2][0])

    # 高收入群体的反馈更积极
    df_subset["salary_range"] = pd.cut(df_subset["income"], [20000, 40000,␣
↪60000, 80000, 100000, df_subset["income"].max()])
```

```
    sns.barplot(x="salary_range", y="valid_ind", data=df_subset, ax=axes[2][1])
    plt.show()

    return
```

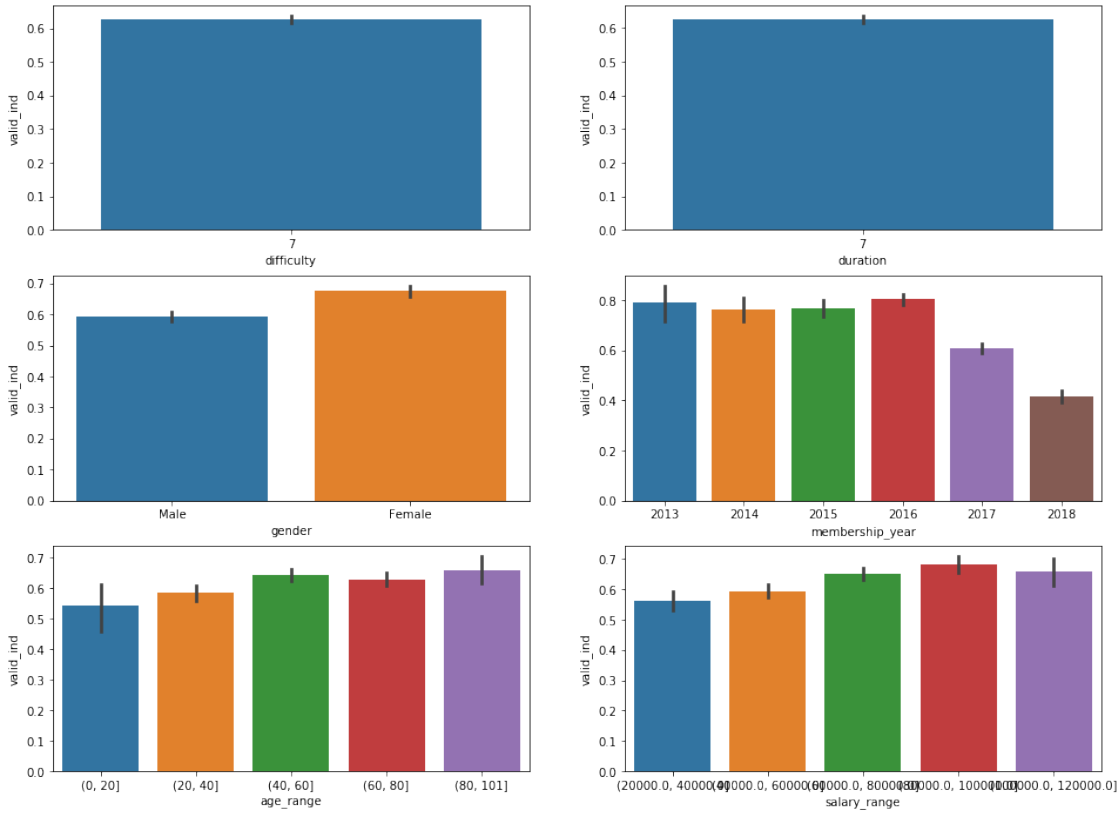
[43]:
```
# 查看整个数据集的情况
visualize_output("")
```

正在查看整个数据集。



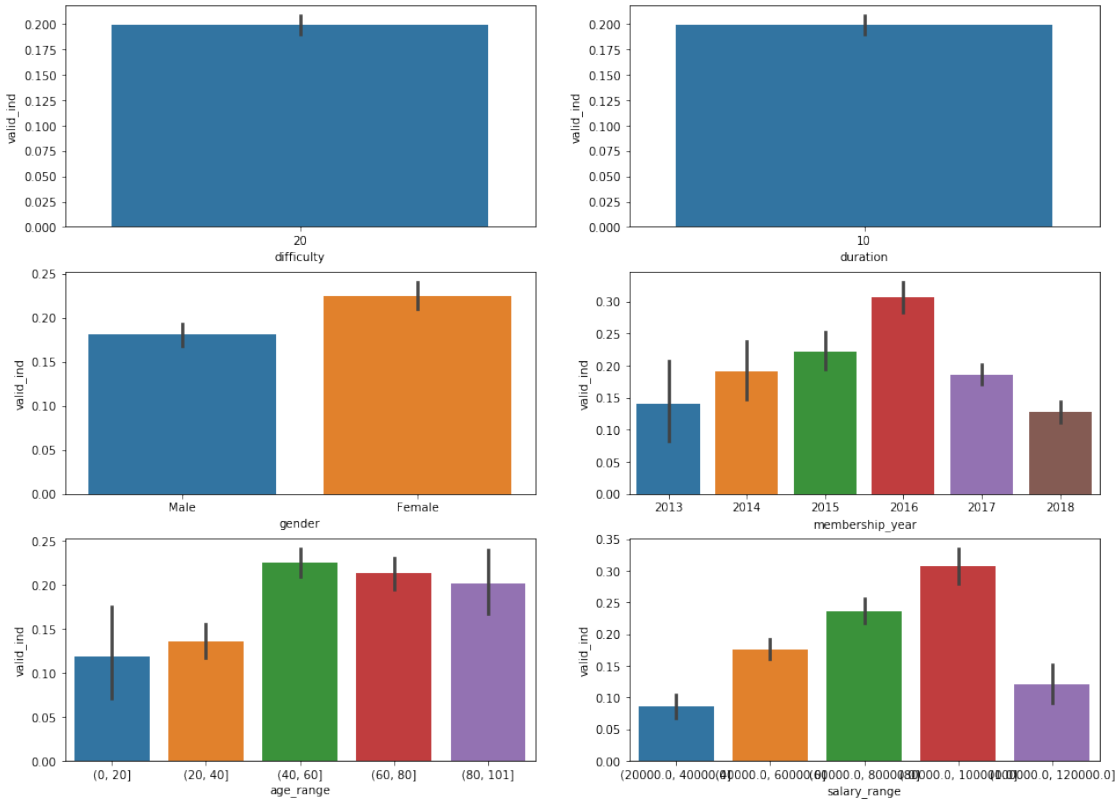
[44]:
```
# 查看完成率高的数据集的情况
visualize_output("2298d6c36e964ae4a3e7e9706d1fb8c2")
```

正在查看 offer_id 为: 2298d6c36e964ae4a3e7e9706d1fb8c2 的数据集。



[45]:
```
# 查看完成率较低的数据集的情况
visualize_output("0b1e1539f2cc45b7b9fa7c272da2e1d7")
```

正在查看 offer_id 为: 0b1e1539f2cc45b7b9fa7c272da2e1d7 的数据集。



小结

- 活动门槛高，完成率会相应低
- 推送时间短，完成率也相应比较低
- 女性对优惠更敏感
- 年轻群体，对优惠普遍不太敏感
- 收入对参与度的影响，视 offer，有不同
- 新会员，参与度在不同的活动类别中有分化

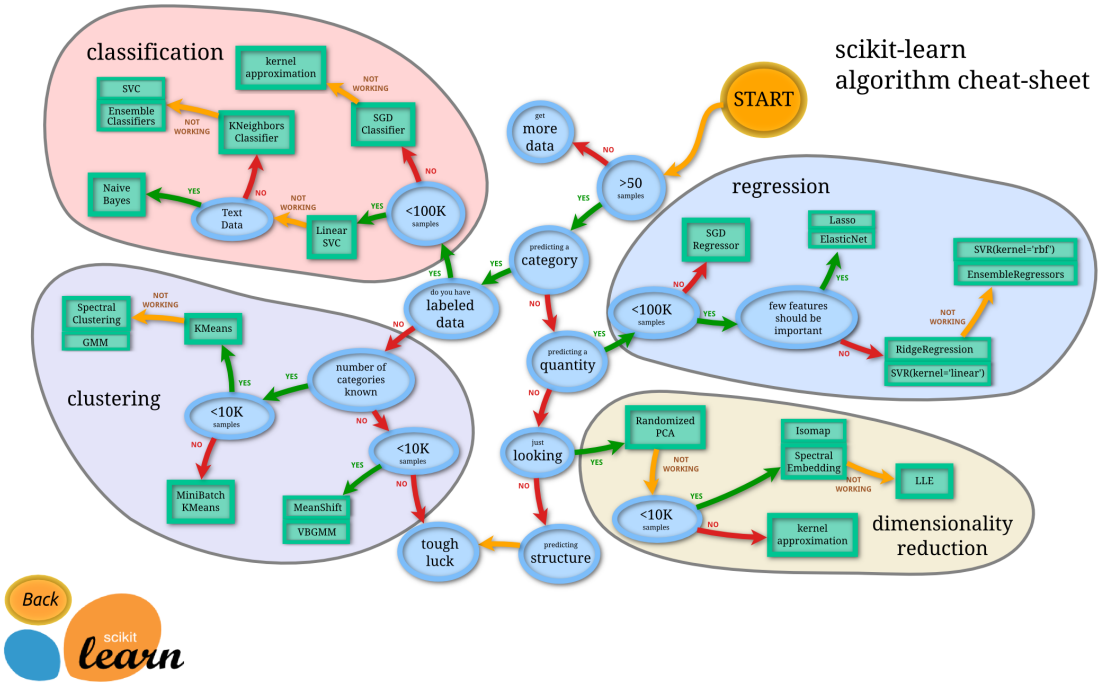
1.1 第三步，数据建模分析

目标：建立一个有效的预测模型，同时解释推送成功的各个影响因素的重要程度，看看有没有那些因素从直观的分析中没有能注意到

模型选择 我自己发现一个最有用的指引是 sklearn 官方教程里头的 machine learning map: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

[46]:
```
# 如何为自己的业务问题选择一个合适的机器学习模型
Image(filename="./ml_map.png")
```

[46]:



根据模型选择指引，我的选择路径是这样的：

- 数据量是 65585 个 sample
- 模型目标是预测分类
- 我们是有标签的数据
- 我们的数据小于 10 万行
- 选择 Linear SVC 试试
- 我们的数据不是文本类型
- 选择 KNN 分类试试
- 选择一个 ensemble 库里头的分类模型（随机森林分类器）

在这个路线上经过三个模型，我先用默认参数比较，随机森林分类器明显好，我继续参考官方文档，进一步调优随机森林分类器

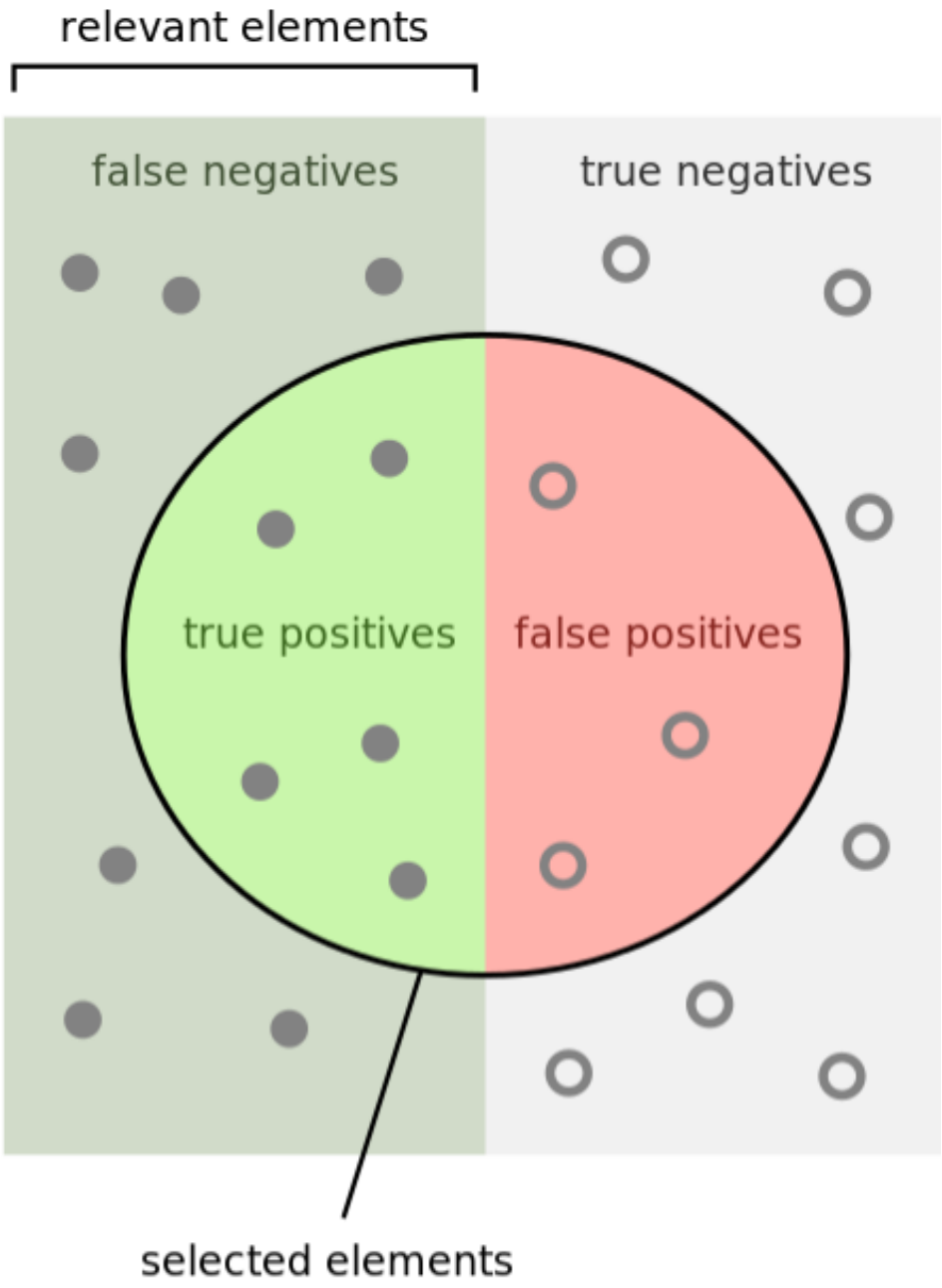
模型评估说明 对于分类模型，有几个重要的指标，说明如下：

- 准确率：这个指标评估一个模型在所有数据上的表现。在及其特殊的情况下，准确率高，不代表模型实用。以信用卡欺诈识别为例，信用卡欺诈本身是一个样本集里头非常少出现的情况，一个模型，只要对所有的样本都预测不是一个欺诈，就可以达到很高的准确率。但是这个模型并没有什么用。我们的数据没有那么特殊，准确率是一个的评估指标
- 精确率：这个指标主要衡量预测结果为 1 的样本，有多大的比例真实是 1，这个指标对于促销类的推送有意义，因为我们希望模型帮我门找出真正感兴趣的消费者，以免对不感兴趣的消费者产生困扰，或者浪费促销成本（当然现在的成本在降低）
- 召回率：这个指标主要衡量样本中为 1 的，有多大比例被模型准确预测。在特定的场合，比如目前的疾病检测，要尽可能多的预测到正样本，哪怕有些负样本被预测为正样本，也不要紧。这个指标对我们的例子也很有用，因为获取消费者数据不容易，要从现有的消费者数据里头尽量找到合适的消费者
- F Beta Score：是精确率和召回率的平衡，是综合考虑精确率和召回率的指标，以下从维基百科中下载的图片能解释这个指标的作用。将 Beta 值设为 1（就是所谓的 F1 Score），其实就能同时反映精确率和召回率的影响。

经过分析，我决定应用准确率和 F1 Score 两个指标来评估模型的表现

[47]:
```
# 精确率和召回率的图示
Image(filename="./precision_recall.png")
```

[47]:



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

3.1 数据集拆分和归一化

[48]:
```
# 生成建模数据集，挑出评有用的字段，去除掉那些只是分析用的，或者已经经过 one-hot 编码的字段
df_clean = df_clean.drop([
    "consumer_id", "offer_id", "time", "end_time", "viewed_time",
    "completed_time", "age", "became_member_on", "gender",
    "membership_year", "age_range", "channels", "offer_type", "salary_range"],␣
 ↪axis=1)
```

[49]:
```
# 分类字段是 valid_ind，1 代表推送互动成功，0 代表推送互动失败
label_column_name = "valid_ind"
features = df_clean.drop(label_column_name, axis=1)
label = df_clean[label_column_name]

# 这样是为了拆分可以复现
state = 123
X_train, X_test, y_train, y_test = train_test_split(features.values, label.
 ↪values, test_size=0.2, random_state=state)
```

[50]:
```
# 几个连续数值字段，进行归一化：income(0), difficulty(14), durationdays(15),␣
 ↪reward(16)
# 第 0 列，income
mms = MinMaxScaler()
mms.fit(X_train[:,0].reshape(-1, 1))
X_train[:,0] = mms.transform(X_train[:,0].reshape(-1, 1)).reshape(1, -1)
X_test[:,0] = mms.transform(X_test[:,0].reshape(-1, 1)).reshape(1, -1)

# 第 14 列，difficulty
mms.fit(X_train[:,14].reshape(-1, 1))
X_train[:,14] = mms.transform(X_train[:,14].reshape(-1, 1)).reshape(1, -1)
X_test[:,14] = mms.transform(X_test[:,14].reshape(-1, 1)).reshape(1, -1)

# 第 15 列，durationdays
mms.fit(X_train[:,15].reshape(-1, 1))
```

```
X_train[:,15] = mms.transform(X_train[:,15].reshape(-1, 1)).reshape(1, -1)
X_test[:,15] = mms.transform(X_test[:,15].reshape(-1, 1)).reshape(1, -1)

# 第 16 列，rewards
mms.fit(X_train[:,16].reshape(-1, 1))
X_train[:,16] = mms.transform(X_train[:,16].reshape(-1, 1)).reshape(1, -1)
X_test[:,16] = mms.transform(X_test[:,16].reshape(-1, 1)).reshape(1, -1)
```

[51]:
```
# 测试脚本，检查一下转换效果
# (X_train[3][16] - np.min(X_train[:, 16])) / (np.max(X_train[:, 16]) - np.
 ↪min(X_train[:, 16]))
```

3.2.1 尝试一下 LinearSVC

[52]:
```
# Follow 官方 machine learning map，第一步尝试一下 LinearSVC
clf = LinearSVC(random_state=state)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_train)

clf_accuracy = accuracy_score(y_train, y_pred)
clf_f1_score = f1_score(y_train, y_pred)

print("LinearSVC 分类模型在训练集上的预测准确率为：{:.3f}".format(clf_accuracy))
print("LinearSVC 分类模型在训练集上的 F1 分数为：{:.3f}".format(clf_f1_score))

print("\n----------------------分割线--------------------\n")

y_pred = clf.predict(X_test)

clf_accuracy = accuracy_score(y_test, y_pred)
clf_f1_score = f1_score(y_test, y_pred)

print("LinearSVC 分类模型在测试集上的预测准确率为：{:.3f}".format(clf_accuracy))
print("LinearSVC 分类模型在测试集上的 F1 分数为：{:.3f}".format(clf_f1_score))
```

```
LinearSVC 分类模型在训练集上的预测准确率为：0.746
LinearSVC 分类模型在训练集上的 F1 分数为：0.599
```

```
----------------------分割线--------------------
```

```
LinearSVC 分类模型在测试集上的预测准确率为：0.747
LinearSVC 分类模型在测试集上的 F1 分数为：0.593
```

3.2.2 初步尝试以下 KNN 分类器

[53]:
```
# Follow 官方 machine learning map, 第二步尝试一下 KNN 分类模型
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)

y_pred = clf.predict(X_train)

clf_accuracy = accuracy_score(y_train, y_pred)
clf_f1_score = f1_score(y_train, y_pred)

print("KNN 分类模型在训练集上的预测准确率为：{:.3f}".format(clf_accuracy))
print("KNN 分类模型在训练集上的 F1 分数为：{:.3f}".format(clf_f1_score))

print("\n----------------------分割线--------------------\n")

y_pred = clf.predict(X_test)

clf_accuracy = accuracy_score(y_test, y_pred)
clf_f1_score = f1_score(y_test, y_pred)

print("KNN 分类模型在测试集上的预测准确率为：{:.3f}".format(clf_accuracy))
print("KNN 分类模型在测试集上的 F1 分数为：{:.3f}".format(clf_f1_score))
```

```
KNN 分类模型在训练集上的预测准确率为：0.803
KNN 分类模型在训练集上的 F1 分数为：0.705

----------------------分割线--------------------

KNN 分类模型在测试集上的预测准确率为：0.726
KNN 分类模型在测试集上的 F1 分数为：0.582
```

3.2.3 初步尝试一下随机森林分类器

[54]:
```
# Follow 官方 machine learning map, 第三步尝试一下集成模型里头随机森林分类模型
clf = RandomForestClassifier(random_state=state, n_estimators=10)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_train)

clf_accuracy = accuracy_score(y_train, y_pred)
clf_f1_score = f1_score(y_train, y_pred)

print("随机森林分类模型在训练集上的预测准确率为: {:.3f}".format(clf_accuracy))
print("随机森林分类模型在训练集上的 F1 分数为: {:.3f}".format(clf_f1_score))

print("\n----------------------分割线--------------------\n")

y_pred = clf.predict(X_test)

clf_accuracy = accuracy_score(y_test, y_pred)
clf_f1_score = f1_score(y_test, y_pred)

print("随机森林分类模型在测试集上的预测准确率为: {:.3f}".format(clf_accuracy))
print("随机森林分类模型在测试集上的 F1 分数为: {:.3f}".format(clf_f1_score))
```

```
随机森林分类模型在训练集上的预测准确率为: 0.838
随机森林分类模型在训练集上的 F1 分数为: 0.757

----------------------分割线--------------------

随机森林分类模型在测试集上的预测准确率为: 0.713
随机森林分类模型在测试集上的 F1 分数为: 0.563
```

- 三个模型，随机森林准确率明显高些，可以进一步调优
- 下一步我查找官方文档，看看还有什么调整空间
- 但是我也留意到，LinearSVC 在训练测试集上，表现更一致些

3.2.4 优化随机森林分类器 没有调参之前，发现随机森林在训练集上表现更好，下一步我们对随机森林分类器进行参数搜索和优化，我参考的主要是官方文档中的指引，在这个指引中提到，随机森

林主要是调整 n_estimator，其它因子，试了多次，感觉改善很小，所以我用 GridSearchCV 来做参数的筛选测试。

[55]:
```
# 随机森林分类器文档里头的参数调优建议
Image(filename="./parameters.png")
```

[55]:

1.11.2.3. Parameters

The main parameters to adjust when using these methods is `n_estimators` and `max_features`. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are `max_features=None` (always considering all features instead of a random subset) for regression problems, and `max_features="sqrt"` (using a random subset of size `sqrt(n_features)`) for classification tasks (where `n_features` is the number of features in the data). Good results are often achieved when setting `max_depth=None` in combination with `min_samples_split=2` (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (`bootstrap=True`) while the default strategy for extra-trees is to use the whole dataset (`bootstrap=False`). When using bootstrap sampling the generalization accuracy can be estimated on the left out or out-of-bag samples. This can be enabled by setting `oob_score=True`.

[56]:
```
# 分类模型建立
clf = RandomForestClassifier(random_state=state)

# 随机森林模型的重要参数
n_estimators = [100, 400, 800, 1200] # 调参建议里头说，越大越好，但是越大，模型的性能越差，多次实验，800 是个比较合适的数值
max_features = ["sqrt", None] # 调参建议里头，分类问题，选择"sqrt"，回归问题选择"none"，我两个都搜索
max_depth = [None] # 推荐参数
min_samples_split = [2] # 推荐参数

# 参数值范围限定
param_grid = {
    "n_estimators": n_estimators,
    "max_features": max_features,
    "max_depth": max_depth,
    "min_samples_split": min_samples_split
}

# beta 值为 1，recall 和 precision 等权重，就是 F1 score 了
```

```
# 我的立足点是希望能调和精确率和召回率两个指标，因为我感觉两个都重要
scorer = make_scorer(fbeta_score, beta=1)

# 网格搜索，遍历参数空间，需要运行 5 分钟左右
rf_grid = GridSearchCV(estimator = clf,
                       param_grid =  param_grid,
                       scoring=scorer,
                       verbose=2,
                       cv=3,
                       n_jobs = 3)

rf_grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits

[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done  24 out of  24 | elapsed: 11.4min finished
```

[56]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False, random_state=123,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=3,
             param_grid={'max_depth': [None], 'max_features': ['sqrt', None],
                         'min_samples_split': [2],
                         'n_estimators': [100, 400, 800, 1200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=1), verbose=2)
```

[57]:
```
# 查看遍历后得到的最佳参数
rf_grid.best_params_
```

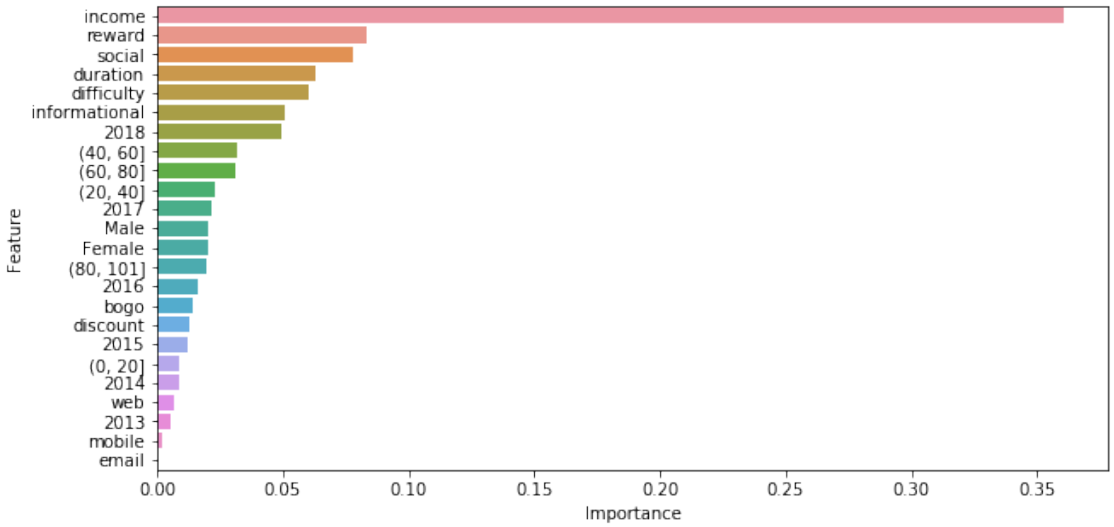
[57]:
```
{'max_depth': None,
 'max_features': None,
 'min_samples_split': 2,
 'n_estimators': 800}
```

[58]:
```
# 调优后的模型，表现都有细微提升
# 模型在训练集上的表现，在真实商业应用中，这样的性能不错
y_pred = rf_grid.predict(X_train)

clf_accuracy = accuracy_score(y_train, y_pred)
clf_f1_score = f1_score(y_train, y_pred)

print("模型在训练集上的准确率为: {:.3f}".format(clf_accuracy))
print("模型在训练集上的 F1 分数为: {:.3f}".format(clf_f1_score))

print("----------------------分割线----------------------")

# 模型在测试集上的表现和训练集上表现稍微差点，但是也比没有调整前好
y_pred = rf_grid.predict(X_test)

clf_accuracy = accuracy_score(y_test, y_pred)
clf_f1_score = f1_score(y_test, y_pred)

print("模型在测试集上的准确率为: {:.3f}".format(clf_accuracy))
print("模型在测试集上的 F1 分数为: {:.3f}".format(clf_f1_score))
```

```
模型在训练集上的准确率为: 0.845
模型在训练集上的 F1 分数为: 0.769
----------------------分割线----------------------
模型在测试集上的准确率为: 0.725
模型在测试集上的 F1 分数为: 0.586
```

[59]:
```
# 因子重要程度展示
importance = pd.DataFrame(list(zip(list(features.columns.values), rf_grid.
 ↪best_estimator_.feature_importances_)),
```

```
                              columns=["Feature", "Importance"])
importance = importance.sort_values("Importance", ascending=False)
plt.figure(figsize=(10, 5))
sns.barplot(x="Importance", y="Feature", data=importance)
plt.show()
```



1.1.1 3.3 结果解读

这里的因子的重要程度，对推送的优化非常有价值，能提供重要的 insight:

- 1，会员收入对推送效果，作用最明显
- 2，优惠的力度，决定了对消费者的吸引力，这个是很容易理解的
- 3，渠道上来说，社交渠道现在的重要性在上升
- 4，推送的有效期也是影响推送效果的重要因素
- 5，优惠的门槛，影响推送效果
- 6，email 在数据中几乎没有，所以重要性低是可以理解的

1.2 结论

拿到这个项目的时候，我通过观察数据，结合自己在工作中接触过的数据分析的例子，我选了一个我感兴趣的切入点：活动设置和消费者属性如何影响推送效果。我的步骤是这样的：

1.2.1 数据探索

在这一步，我导入数据，观察数据的字段的值，结合项目给到的元数据信息，充分理解数据集的业务意义。在这个过程当中，我也开始思考，原始数据需要经过哪些转换才能变成对最终建模有用的数据。

1.2.2 数据整理

这一步非常有挑战。除了充分应用 sklearn 中的 MultiLabelBinarizer 和 MinMaxScaler 等数据预处理工具以及 pandas.cut, pandas.get_dummies 等实用函数，我花了不少时间在理解 transcript 数据集的处理上。这个数据集其实包含了四种类型的数据，后来我把它细分成四个数据集方便处理。分完后，我觉得评估推送效果，需要对每次推送，都要找 viewed 和 compeleted，用一定的逻辑来标识一个推送的互动是否是真的有效完成。

1.2.3 初步的数据分析

在没有建模之前，我也做些直观的数据分析，可以分析的维度很多，包括活动属性的维度，消费者属性的维度，感觉不容易直观地把握哪个特征比较值得分析。这时候更说明了，我通过建模，找到模型和影响因子地重要程度是具有业务现实意义的

1.2.4 数据建模和评估

为了完成这个项目，我再次到 sklearn 网站回顾了教程地内容，根据 machine learning map，考虑我的数据集特征和业务性质，在路线图上找到三个可以测试的模型（LinearSVC，KNN 分类器，集成模型中挑一个随机森林分类器）。然后我选定用准确率和 F1 分数值来初步筛选模型。对于选出来的模型（随机森林分类器），我选择 GridSearchCV 搜索最佳的参数，返回最佳模型参数。

1.2.5 回顾与总结

1. 通过这个项目，我的 Pandas Dataframe 应用水平得到了实战锻炼。之前我喜欢对照着 Excel 去学 Python，总感觉没有真正体会到它的好处。现在通过项目，发现 Pandas 在性能上，在数据处理量级上，都比其它用户类工具好很多。

2. 项目最有挑战的地方，对我来说是模型选择。网上我看了不少文章，走了点弯路。后来我静下心来，回到最基本的东西，再仔细看了 SKLearn 中的官方文档，发现其实官方就提供了一个“Machine Learning Map”，它实质上告诉用户，根据数据集的大小，问题的类别，选择模型是有个最佳路径的，我在具体步骤中有详细说明我是怎么三个备选模型来分析和优化的。从这以后，我对 SKLearn 模型的使用，将会更加的心应手。

3. 在对分类模型的学习和实践中，我更深入认识到了各种模型的特点和调优过程。模型和问题其实是有“缘分”的，需要根据模型的特点，问题的类型，数据集大小，因子数目等等进行挑选和尝试。

1.2.6 代码实现的改进

1. 我经过课程软件工程部分的学习后，体会到两个很关键的点：减少代码重复（之前读设计模式之类的书籍，没能体会到其实设计模式的最终目的，就是增加代码复用减少代码重复）和对最佳性能的不断探索。尤其在最佳性能探索方面，在校验推送是否有效时，我第一版是用 Python 循环实现的，对 offer_received 表每一条循环，然后在循环里，匹配 offer_viewed 和 offer_completed，然后对比时间字段值，看是否在有效期范围。运行一次下来，需要 10 分钟左右。后来我想到选修课程里头的 SQL，这样的数据处理，对 SQL 语法就是小菜一碟，所以我在这个项目里头突然使用到了一条 SQL 语句，就是用来代替我的循环设计的，这个 SQL，用 offer_received 表，left join 两个表, 几秒钟出结果，性能提高了 60 倍。这点很有启发意义。代码优化需要蛮多时间和经验，后续我会不断总结 best practice，我觉得如果再经过几个真实项目，我再做回这个项目，我的代码质量和性能还会更上一个台阶。

2. 另外，我在模型调优部分，刚开始是所有参数都一把抓，尤其分类模型的 n_esimator，从 100，到 10 万, 结果发现耗时，根本无法运行出结果。于是我参考了文档，找出关键的四个参数，再进行不断调优，基本参数的大致范围就被我限定住了，能提升的空间也有了感觉。机器学习真是一个需要耐心的去摸索的事物，但是也值得我深入再课后好好研究。我发现 ensemble 库里还有好几个分类模型，如果能花大量时间，仔细都学习一遍，用真实数据来研究应用，我的预测准确率应该还有很大的提升空间。

1.2.7 参考链接

我整个做的过程当中，参考优达学成的视频课程和 SK Learn 的官网为主，也在网上做了些学习和拓展，以下是实验进行中的参考文献和链接：

- 【1】https://retailanalysis.igd.com/ （一个很好的网站，提供很多关于零售业数据洞察的最新进展）
- 【2】https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html （Scikit-Learn 官方一个关于预测模型选择的实用指引）
- 【3】https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees （官方文档有参数调整的建议）
- 【4】https://scikit-learn.org/stable/modules/grid_search.html#（关于模型参数网格遍历的参考链接）
- 【5】https://www.jianshu.com/p/1afbda3a04ab（关于模型指标的解释，这里有篇很明细的文章）

- 【6】https://www.w3school.com.cn/sql/index.asp （这里有一个 SQL 语法的速查链接，我常用）

[ ]: