基于移动推荐算法的用户购买预测

团队成员：2120171046 欧英子

1 简介

该选题是天池新人赛《阿里移动推荐算法》，以阿里巴巴移动电商平台的真实用户-商品行为数据为基础，同时提供移动时代特有的位置信息，而参赛队伍则需要通过大数据和算法构面向建移动电子商务的商品推荐模型，挖掘数据背后丰富的内涵，为移动用户在合适的时间、合适的地点精准推荐合适的内容。

我分别从数据预处理、数据分析与可视化、结果评价和结果展示四个部分完成用户购买预测。代码地址为：https://github.com/oneoyz/DM_User-Purchase-Prediction-Based-on-Mobile-Recommendation-Algorithm

2 问题陈述

在真实的业务场景下，我们往往需要对所有商品的一个子集构建个性化推荐模型，达到用户购买商品的推荐。在完成这件任务的过程中，我们不仅需要利用用户在这个商品子集上的行为数据，往往还需要利用更丰富的用户行为数据。除了基础的真实用户-商品行为数据，移动时代还特有带位置信息的数据，这就需要通过大数据和算法构面向建移动电子商务的商品推荐模型，挖掘数据背后丰富的内涵，为移动用户在合适的时间、合适的地点精准推荐合适的内容。

定义如下符号：

U——用户集合

I——商品全集

P——商品子集，$P \subseteq I$

D——用户对商品全集的行为数据集合

那么我们的目标是利用 D 来构造 U 中用户对 P 中商品的推荐模型。

2.1 数据准备

采用了阿里云天池比赛提供的数据集，包括了 20000 用户的完整行为数据以及百万级的商品信息，数据包含两个部分。 第一部分是用户在商品全集上的移动端行为数据（D），表名为 tianchi_fresh_comp_train_user_2w，包含如下字段：

字段	字段说明	提取说明
user_id	用户标识	抽样&字段脱敏
item_id	商品标识	字段脱敏
behavior_type	用户对商品的行为类型	包括浏览、收藏、加购物车、购买，对应取值分别是1、2、3、4。
user_geohash	用户位置的空间标识，可以为空	由经纬度通过保密的算法生成
item_category	商品分类标识	字段脱敏
time	行为时间	精确到小时级别

图 1：D 表字段

第二个部分是商品子集(P),表名为 tianchi_fresh_comp_train_item_2w, 包含如下字段：

字段	字段说明	提取说明
item_id	商品标识	抽样&字段脱敏
item_ geohash	商品位置的空间标识，可以为空	由经纬度通过保密的算法生成
item_category	商品分类标识	字段脱敏

图 2：P 表字段

训练数据包含了抽样出来的一定量用户在一个月时间（11.18~12.18）之内的移动端行为数据（D)。

2.2 项目评估

采用经典的精确度(precision)、召回率(recall)和 F1 值作为评估指标。具体计算公式如下：

$$\text{Precision} = \frac{|\cap(PredictionSet,\ ReferenceSet)|}{|PredictionSet|}$$

$$\text{Recall} = \frac{|\cap(PredictionSet,\ ReferenceSet)|}{|ReferenceSet|}$$

$$\text{F1} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

其中 PredictionSet 为算法预测的购买数据集合，ReferenceSet 为真实的答案购买数据集合，F1 值作为最终的唯一评测标准，本选题的结果展示为上交至天池平台计算的结果。

3 技术方案

3.1 数据预处理

首先统计数据大致信息，可以观察到大部分缺失的是表示地理位置信息的 geohash，考虑到地点标记可能有用，不剔除这些数据。

```
len is 23291027

user_id          int64
item_id          int64
behavior_type    int64
user_geohash    object
item_category    int64
time            object
dtype: object

           user_id       item_id  behavior_type  item_category
count  2.329103e+07  2.329103e+07   2.329103e+07   2.329103e+07
mean   7.006868e+07  2.023214e+08   1.106268e+00   6.835397e+03
std    4.569072e+07  1.167440e+08   4.599087e-01   3.812873e+03
min    4.920000e+02  3.700000e+01   1.000000e+00   2.000000e+00
25%    3.019541e+07  1.014417e+08   1.000000e+00   3.690000e+03
50%    5.626942e+07  2.022430e+08   1.000000e+00   6.054000e+03
75%    1.166482e+08  3.035325e+08   1.000000e+00   1.027100e+04
max    1.424430e+08  4.045625e+08   4.000000e+00   1.408000e+04

missing count of geohash: 15911010
missing count of time: 0
```

```
len is 620918

item_id          int64
item_geohash    object
item_category    int64
dtype: object

           item_id  item_category
count  6.209180e+05  620918.000000
mean   2.004351e+08    6970.213167
std    1.191648e+08    3479.627372
min    9.580000e+02       2.000000
25%    9.357641e+07    4245.000000
50%    2.053761e+08    6890.000000
75%    3.054015e+08   10120.000000
max    4.045624e+08   14071.000000

missing count of geohash: 417508
```

3.2 数据分析和可视化

首先对 11.18-12.16 的用户操作做统计，可以发现 12.11 和 12.12 两日用户操作次数远高于其他天数，为了消除“兴奋日”对建模的影响，将这两天的数据剔除。

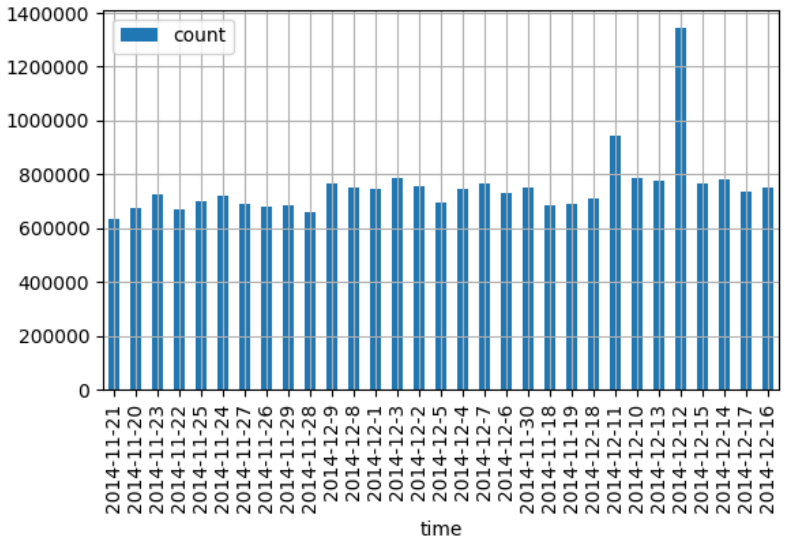


图 3：11.18-12.16 用户操作总统计

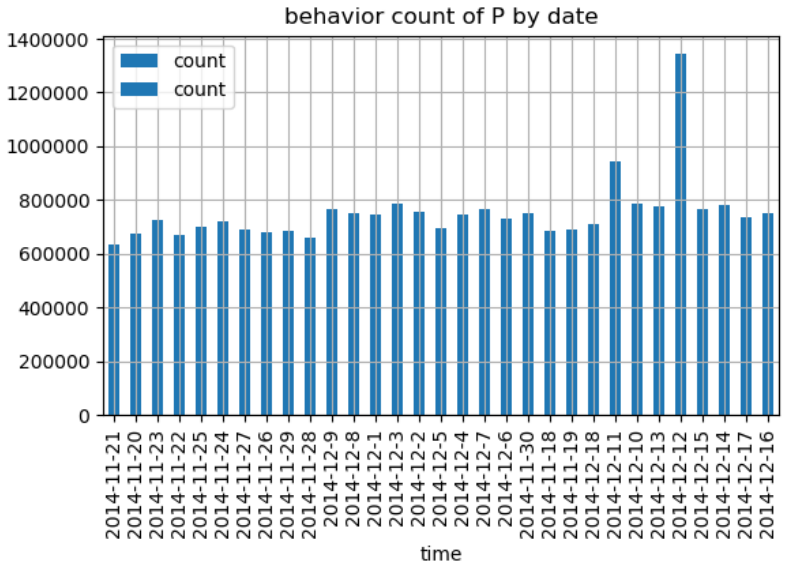


图 4：11.18-12.16 商品子集用户操作总统计

上面两张图可以反映除了双 12 期间的操作记录猛增，购买总体都是比较稳定的。为了

挖掘不同操作的具体反映，选取了 12.17 和 12.18 两日 48 小时的操作进行呈现。

0-3 分别对应浏览、收藏、加购物车、购买。浏览的操作次数要远远多于其他，这符合常理，同时也反映出一些符合实际的规律，比如操作的高峰在较为空闲的晚上，凌晨的时候操作次数最少。单独看购买的操作，10:00-15:00 和 20:00-22:00 是两个操作高峰时段。

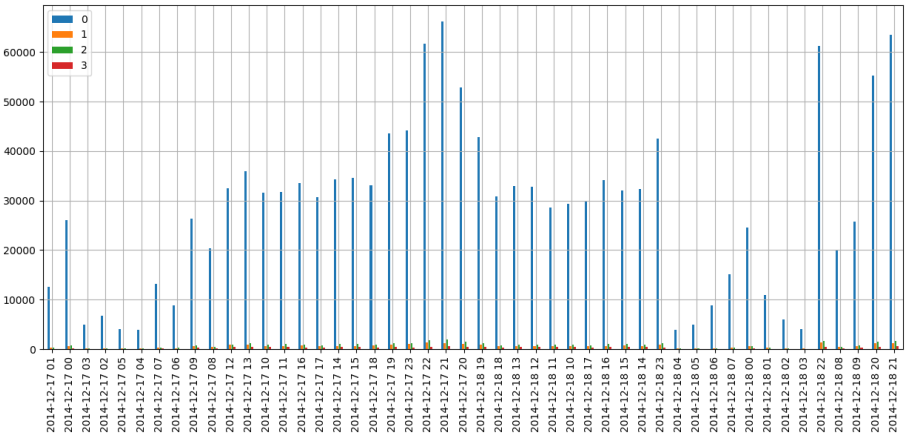


图 5：12.17-12.18 用户各操作统计

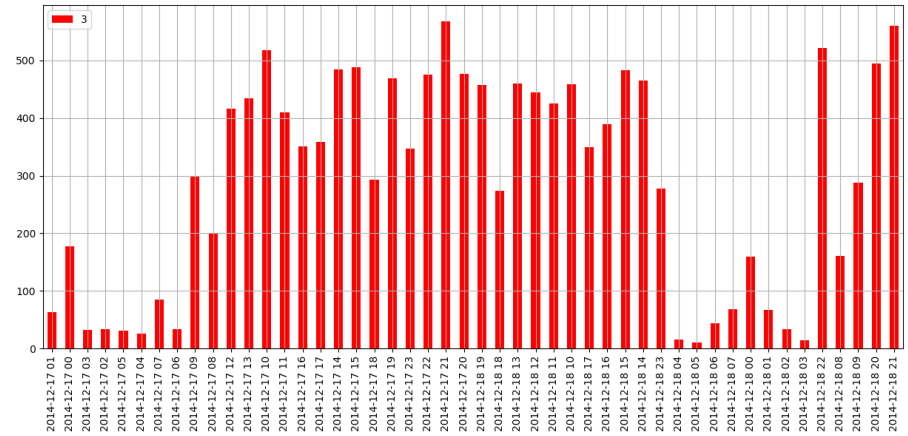


图 6：12.17-12.18 用户购买操作统计

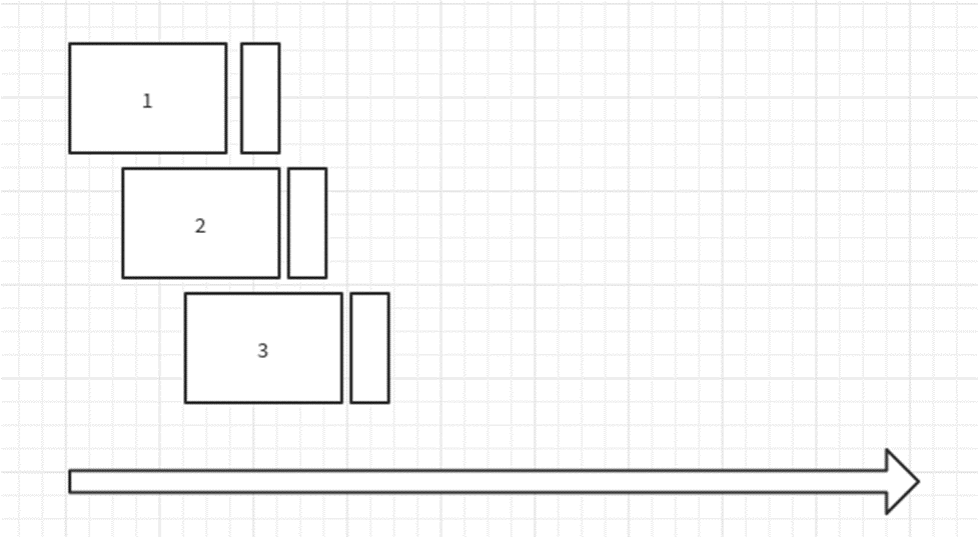
	0	1	2	3	CTR
10001082	207	0	0	4	0.019324
111345634	6505	45	260	81	0.012452
107369933	743	1	33	8	0.010767
10827687	2826	11	111	21	0.007431
111699844	2126	54	13	9	0.004233
10496835	1340	2	30	5	0.003731
110507614	11590	353	426	24	0.002071
108461135	18023	439	244	24	0.001332
108266048	5421	42	3	1	0.000184
110939584	202	2	6	0	0

图 7：部分的用户展示操作统计

上图是选取了部分的用户展示，CTR 是购买转换率 浏览/购买。可以看到用户 10001082 的 CTR 为 0.019，且没有收藏和加购物车的行为，说明该用户行为决断。用户 108461135 操作次数非常多，但购买率不怎么高。

3.3 数据集处理

我们总共获得了 1000 万用户在一个月的行为数据 (11.18~12.18)，需要预测的是在 12 月 19 日用户的购买情况，因而我们采用滑窗的形式来构造训练集和预测集。



将数据打好标签，某一天作为验证或测试集，当天根据用户操作进行打标<用户-商品-种类-标签>，1 为购买，0 为未购买。

```
def Get_train(train_user,end_time):
    # 取出label day 前一天的记录作为打标记录
    data_train = train_user[(train_user['daystime'] == (end_time-datetime.timedelta(days=1)))]
    # 训练样本中，删除重复的样本
    data_train = data_train.drop_duplicates(['user_id', 'item_id'])
    data_train_ui = data_train['user_id'] / data_train['item_id']
#   print(len(data_train))

    # 使用label day 的实际购买情况进行打标
    data_label = train_user[train_user['daystime'] == end_time]
    data_label_buy = data_label[data_label['behavior_type'] == 4]
    data_label_buy_ui = data_label_buy['user_id'] / data_label_buy['item_id']

    # 对前一天的交互记录进行打标
    data_train_labeled = data_train_ui.isin(data_label_buy_ui)
    dict = {True: 1, False: 0}
    data_train_labeled = data_train_labeled.map(dict)

    data_train['label'] = data_train_labeled
    return data_train[['user_id', 'item_id','item_category', 'label']]
```

再加入数据负采样，根据打标签划分之前的数据，分为已购买的对和未购买的对，比例为 1:0.9。

```
train_set_1 = train_set[train_set['label']==1]
train_set_0 = train_set[train_set['label']==0]
new_train_set_0 = train_set_0.sample(len(train_set_1)*90)
train_set = pd.concat([train_set_1,new_train_set_0],axis=0)
```

3.4 特征选择

关于特征的构建，我们必须要明确一点：我们需要预测什么？只有清楚的明白预测的主体，我们才能去构建合适的特征。官方需要我们给出的是所有<用户，子集商品>的购买情况，我们可以从任意一天的<用户，子集商品>对的购买情况看出，其中绝大部分购买是用户当天交互当天购买（一般有 2/3 左右），这些用户商品对没有任何历史情况。对于这部分购买，我们实际上直接选择了放弃预测，即只考虑预测这一个月中有历史记录的用户商品对，这主要是基于以下两个原因：

（1）不能确定用户是否会购买该商品

（2）不能确定用户会在什么时候购买

我们没法通过学习历史数据去评估这两个因素的影响，因而直接凭空预测的代价太高，会严重影响整体预测的准确度，这也决定了我们召回率的极限。所以，我们的预测主体是有交互历史用户商品对，预测的内容是用户是否会购买该商品、用户是否会在预测日当天购买。因此，我将特征归为以下四类[1][1]：

1. 用户特征

[1] https://blog.csdn.net/u014374284/article/details/49933487

用户特征就只是针对用户来说的，反映的是用户整个购物习惯与购物规律，而与具体哪件商品无关，比如用户是不是喜欢浏览购物网站、用户的购物频率等。

```
def user_click(beforesomeday):
    user_act_count = pd.croosstab([beforesomeday.user_id, beforesomeday.behavior_type], beforesomeday.hours, dropna = False)
    user_act_count = user_act_count.unstack(fill_value = 0)
    return user_act_count

def user_activeday(train_user_window1):
    user_act_day = train_user_window1.groupby(by = ['user_id', 'behavior_type']).agg({"daytime": lambda x:x.nunique()})
    user_act_day = user_live.unstack(fill_value = 0)
    return user_act_day
```

2. 商品特征

商品特征反映的是商品本身的品质或者受欢迎程度如何，而与具体哪一个用户没有关系，同时商品特征也体现了商品的活动规律，即被用户购买的频率、最后有人购买的时间等。

```
def user_item_click(beforesomeday):
    user_item_act_count = pd.croosstab([beforesomeday.user_id, beforesomeday.item_id, beforesomeday.behavior_type], beforesomeday.hours)
    user_item_act_count = user_item_act_count.unstack(fill_value = 0)
    return user_item_act_count

def user_cate_click(beforesomeday):
    user_cate_act_count = pd.croosstab([beforesomeday.user_id, beforesomeday.item_category, beforesomeday.behavior_type], beforesomeday.hours)
    user_cate_act_count = user_cate_act_count.unstack(fill_value = 0)
    return user_cate_act_count

def user_item_long_touch(train_user_window1):
    _active = train_user_window1.groupby(by = ['user_id', 'item_id']).agg({"daytime": lambda x:(x.max() - x.min()).days})
    return _active

def user_cate_lone_touch(train_user_window1):
    _active = train_user_window1.groupby(by = ['user_id', 'item_category']).agg({"datetime": lambda x:(x.max() - x.min()).days})
    return _active
```

3. 协同特征

协同特征则是以<用户，商品>作为统计对象，是用来表现某个用户对某件商品的喜爱程度或是购买的可能性，这一部分特征直接与测试集数据对接，对预测结果起着决定性的作用。

4. 类别特征

类别特征对预测的影响主要体现在两个方面 :第一, 通过对用户对某一类商品的浏览、收藏、加购、购买情况可以看出用户是否有可能购买该商品（不是误点），以及用户对这一类商品的偏爱程度；第二，主要是体现在物品竞争上，当我们需要判断用户是否会购买某个商品，可以看用户在看该商品是还关注了多少同类商品，并借此来评判用户购买该商品的可能性。

3.5 利用 xgboost 工具构造提升树

```
params={
'booster':'gbtree',
'objective': 'multi:softmax', #多分类的问题
'num_class':10, # 类别数，与 multisoftmax 并用
'gamma':0.1,  # 用于控制是否后剪枝的参数,越大越保守，一般0.1、0.2这样子。
'max_depth':12, # 构建树的深度，越大越容易过拟合
'lambda':2,  # 控制模型复杂度的权重值的L2正则化项参数，参数越大，模型越不容易过拟合。
'subsample':0.7, # 随机采样训练样本
'colsample_bytree':0.7, # 生成树时进行的列采样
'min_child_weight':3,
# 这个参数默认是 1，是每个叶子里面 h 的和至少是多少，对正负样本不均衡时的 0-1 分类而言
#，假设 h 在 0.01 附近，min_child_weight 为 1 意味着叶子节点中最少需要包含 100 个样本。
#这个参数非常影响结果，控制叶子节点中二阶导的和的最小值，该参数值越小，越容易 overfitting。
'silent':0 ,#设置成1则没有运行信息输出，最好是设置为0.
'eta': 0.007, # 如同学习率
'seed':1000,
'nthread':7,# cpu 线程数
#'eval_metric': 'auc'
}
plst = list(params.items())
num_rounds = 5000 # 迭代次数
watchlist = [(xgb_train, 'train'),(xgb_val, 'val')]

#训练模型并保存
# early_stopping_rounds 当设置的迭代次数较大时，early_stopping_rounds 可在一定的迭代次数内准确率没有
model = xgb.train(plst, xgb_train, num_rounds, watchlist,early_stopping_rounds=100)
model.save_model('./model/xgb.model') # 用于存储训练出的模型
print "best best_ntree_limit",model.best_ntree_limit
```

```
###############
train_y = train_set['label'].values
train_x = train_set.drop(['user_id', 'item_id','item_category', 'label'], axis=1).values
test_x = test.drop(['user_id', 'item_id','item_category'], axis=1).values
num_round = 900
params = {'max_depth': 4, 'colsample_bytree': 0.8, 'subsample': 0.8, 'eta': 0.02, 'silent': 1,
          'objective': 'binary:logistic','eval_metric ':'error', 'min_child_weight': 2.5,#'max_delta_step':10,'gamma':0.1,'scale_pos_weight':230/1,
          'seed': 10}  #
plst = list(params.items())
dtrain = xgb.DMatrix(train_x, label=train_y)
dtest = xgb.DMatrix(test_x)
bst = xgb.train(plst, dtrain, num_round)
predicted_proba = bst.predict(dtest)
```

4 实现与结果展示

代码利用 python 实现，工程代码为 preprocess 和 feature 文件夹，分别进行数据处理、特征提取的工作，相关的图片和处理后的数据存放在 output 和 dataset 文件夹中。截止至 2018.5.11 的新人赛第一赛季最优排名/成绩为 38 / 10.02% / 0.09。

时间	F1评分	准确率	当天排名
2018-04-16 15:38:20	10.02465078%	0.08714286	1