“用户对品类下店铺的购买预测”报告

没有猴子组

小组成员：
组长:黄裕文 24320162202841

组员：陈昱甫 24320162202815
邱恒哲 24320162202891

小组分工：

lightgbm 模型、lightgbm 部分报告	陈昱甫、邱恒哲
xgboost 模型、xgboost 部分报告撰写	黄裕文

目 录

我们组尝试了 Xgboost 和 Lightgbm 两个算法

Lightgbm：

1. 特征提取：

赛题数据一共五个表，行为数据（jdata_action），评论数据（jdata_comment），商品数据（jdata_product），商家店铺数据（jdata_shop），用户数据（jdata_user），数据从 2018.2.10 到 2018.4.15

算法思路是使用一个时间跨度为 a+7（天）的数据表，使用前 a 天的用户行为数据，预测后 7 天的用户购买行为。根据上述思路，jdata_action 表中的表项应该大部分选定为特征（未选行为唯一标识号 module_id，对预测没有参考价值，未选）。用户的购买行为会受到多种因素影响，比如商品的评论数、好评率，商品对应的店铺评分、粉丝数、员数，用户的基本信息——例如用户的消费等级（体现为会员级别）、用户年龄、用户性别。用户的所在地区没有选定为特征，因为所在地的等级和用户的消费水平之间并不存在明显的对应关系。

综上所述，我们选定了以下数据作为特征值：

1） 行为类型（type）——jdata_action

2） 行为时间（action_time）——jdata_action

以上数据来源 jdata_action 表

3） 评论数（comments）

4） 好评率（通过好评数计算，命名 goodComment_rate）

以上数据来源 jdata_comment 表

5） 粉丝数(fans_num)

6） 会员数(vip_num)

7） 店铺评分(shop_score)

以上数据来源 jdata_shop 表

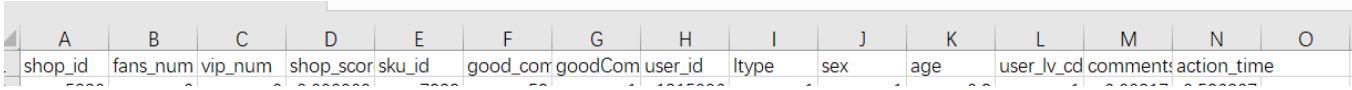
8） 年龄(age)

9） 性别(sex)

10） 会员级别(user_lv_cd)

以上数据来源 jdata_user 表

确定数据项后，读取所有数据，将行为数据表（jdata_action）和商品数据表(jdata_product)根据 sku_id 连接，获得一张包含有 user_id，sku_id，shop_id 的表，之后再根据 id 与剩余 3 个表连接，获得总数据表后剔除无用数据获得特征表。



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
shop_id	fans_num	vip_num	shop_scor	sku_id	good_con	goodCom	user_id	ltype	sex	age	user_lv_cd	comments	action_time	

2. 数据预处理：

首先，设置 a=30，根据时间段的不同，获得 5 张特征表（开始日期分别为 2/10，2/17，2/24，3/3，3/9）和这五张数据表对应的 31-37 天的行为表，根据 id 将特征表和对应的行为表进行连接，如果 type 等于 2，设置为 1，否则设置为 0。得到模型的输入数据集。将特征表的 type 设置为 ltype，行为表的数据设置为 rtype。最终得到 5 组数据。

将各个缺失值填补为均值，然后做 min-max 的归一化处理。

3. 模型训练：

使用了 sklearn 对数据进行处理，将输入数据集分为 2 部分，训练集（0.8）和测试集（0.2）。

提取出特征表中的特征数据集作为 train_set，rtype 作为 valid_set 进行训练并将该学习机保存成文件。因有 5 组数据，读取文件中的弱学习机并学习下一组数据，使学习机变强。

4. 模型预测：

使用最后 30 天的数据作为预测数据。

5. 结果获取：

模型预测的是 4/15 日之后一周用户购买行为（用户-商品-商店）发生的概率

lightgbm 模型测试成绩：

XGBOOST：

特征提取：

1、用户行为数据

- 从行为数据表（jdata_action）获取选取时间段内的用户行为数据，其中删除字段 module_id——既不能在预测中作为特征，也不适合作为预测项；字段 type 进行 one_hot 编码处理。
- 将用户行为数据中统计在一个时间段内该用户对除所研究商品外其它商品的浏览和、关注和、下单和等，及该商品除所研究用户外其他用户的浏览和、加购物车和、下单和等，将他们作为用户-商品对的热度，不是直接使用该用户对所有商品的行为特征和所有用户对该商品的行为特征和是为了避免特征冗余。

  this user all product —— 该用户对某商品的行为特征
  this user all product —— 该用户对所有商品的行为特征
  all user this product —— 所有用户对该商品的行为特征
  this user other product —— 该用户对除某商品意外其他商品的行为特征
  other user this product —— 除该用户以外其他用户对某商品的行为特征

```
1. def get_others_action_feat(end_date, days = 30):
2.
3.     actions = get_actions(end_date, days)
4.     actions = actions[['user_id', 'sku_id', 'type']]
5.
6.     feats = ['type_1', 'type_2', 'type_3','type_4']
7.
```

```
    # this user this product
    tutp_feats = ['tutp_%s' % x for x in feats]
    dummies = pd.get_dummies(actions['type'], prefix='tutp_type')
    tutp_df = pd.concat([actions[['user_id', 'sku_id']], dummies], axis=1)
    tutp_df = tutp_df.groupby(['user_id', 'sku_id'], as_index=False).sum()

    # this user all product
    tuap_feats = ['tuap_%s' % x for x in feats]
    dummies = pd.get_dummies(actions['type'], prefix='tuap_type')
    tuap_df = pd.concat([actions[['user_id']], dummies], axis=1)
    tuap_df = tuap_df.groupby(['user_id'], as_index=False).sum()

    # all user this product
    autp_feats = ['autp_%s' % x for x in feats]
    dummies = pd.get_dummies(actions['type'], prefix='autp_type')
    autp_df = pd.concat([actions[['sku_id']], dummies], axis=1)
    autp_df = autp_df.groupby(['sku_id'], as_index=False).sum()

    df = pd.merge(tutp_df, tuap_df, how = 'left', on = ['user_id'])
    df = pd.merge(df, autp_df, how = 'left', on = ['sku_id'])

    # this user other product
    tuop_feats = ['tuop_%s' % x for x in feats]

    # other user this product
    outp_feats = ['outp_%s' % x for x in feats]
    for i in range(len(feats)):
        df[tuop_feats[i]] = df[tuap_feats[i]] - df[tutp_feats[i]]
        df[outp_feats[i]] = df[autp_feats[i]] - df[tutp_feats[i]]
    others_action = df[['user_id', 'sku_id'] + tuop_feats + outp_feats]

    return (others_action)
```

- 获取一段时间内对每个商品的所进行的行为转化为下单行为的比率，并将他们作为“用户-商品”对特征。

```
def get_product_conversion(end_date, days = 30):
    feature = ['sku_id', 'product_type_1_ratio', 'product_type_3_ratio', 'product_type_4_ratio']
    actions = get_actions(end_date, days)

    dummies = pd.get_dummies(actions['type'], prefix='type')
    actions = pd.concat([actions[['sku_id']], dummies], axis=1)
    actions = actions.groupby(['sku_id'], as_index=False).sum()

```

```
    actions['product_type_1_ratio'] = actions['type_2'] / actions['type_1']

    actions['product_type_3_ratio'] = actions['type_2'] / actions['type_3']

    actions['product_type_4_ratio'] = actions['type_2'] / actions['type_4']

    # actions['product_type_5_ratio'] = actions['type_2'] / actions['type_5'
]

    actions = actions[feature]
    actions = actions.fillna(0).replace(np.inf, 0)

    return(actions)
```

➢ 计算用户每个行为转化为下单行为的比率

```
def get_user_order_ratio(end_date, days):
    feature = ['user_id', 'user_type_1_ratio', 'user_type_3_ratio', 'user_ty
pe_4_ratio']
    actions = get_actions(end_date, days)

    dummies = pd.get_dummies(actions['type'], prefix='type')
    actions = pd.concat([actions[['user_id']], dummies], axis=1)
    actions = actions.groupby(['user_id'], as_index=False).sum()

    actions['user_type_1_ratio'] = actions['type_2'] / actions['type_1']
    actions['user_type_3_ratio'] = actions['type_2'] / actions['type_3']
    actions['user_type_4_ratio'] = actions['type_2'] / actions['type_4']
    # actions['user_type_5_ratio'] = actions['type_2'] / actions['type_5']

    actions = actions[feature]
    actions = actions.fillna(0).replace(np.inf, 0)
    return (actions)
```

2、评论数据

统计一段时间内某个商品的总评论数、总好评数、总差评数，并由此获得商品的好评率。由于已有好评率和总评论数，再保留好评数（good_comments）和差评数（bad_comments）将导致特征冗余，将他们舍去。

```
def get_basic_comment_feat(end_date, days):
    comments = pd.read_csv(comment_path)
    start_date = datetime.strptime(end_date, '%Y-%m-%d') - dt.timedelta(days
= days)
    start_date = start_date.strftime('%Y-%m-%d')
```

```
    comments = comments[(comments.dt >= start_date)&(comments.dt < end_date)]
    comments = comments.groupby(['sku_id'], as_index=False).sum()
    comments['bad_comment_ratio'] = comments.bad_comments / comments.comments
    # comments['good_comment_ratio'] = comments.good_comments / comments.comments
    comments['comments']=comments['comments'].astype(int)
    del comments['bad_comments']
    del comments['good_comments']

    return (comments)
```

3、商品数据

保留商品数据（jdata_product）所有字段，其中将 market_time 转化为时间戳并将该列数据进行最大最小标准化（上市时间相对长短对用户购买意向有影响）。

时间转化函数：

```
def transfer_time(data,atrribute):
    timeSeries=data[atrribute]
    timeList=list(timeSeries)
    len(timeList)

    times=[]
    for i in timeList:
        sturct_time=time.strptime(i,'%Y-%m-%d %X.0')
        timeStamp = int(time.mktime(sturct_time))
        times.append(timeStamp)
    times_array=np.array(times)
    print(times_array)
    min_time=min(times_array)
    max_time=max(times_array)
    print(min_time)
    times_array=(times_array-min_time)/(max_time-min_time)
    print(times_array)
    data[atrribute]=pd.Series(times_array)
    return data
```

4、商家店铺数据

删除商家店铺数据（jdata_shop）中商家 id（vender_id）字段（对表的连接没有作用且不考虑作为训练特征）以及开店时间（shop_reg_tm）字段（认为用户的行为是需求驱动的，首先用户考虑的是这家店有没有目标商品，其次才考虑开店时间长信誉度等问题，所以

在这里认为开店时间对结果影响较小，舍去），并将主营类目 cate 改名为 main_cate（考虑到之后表连接会与商品数据中的 cate 冲突）。

```
# 获取店铺的特征
def get_basic_shop_feat():
    shop = pd.read_csv(shop_path)

    shop['main_cate']=shop['cate']
    del shop['vender_id']
    del shop['shop_reg_tm']

    return (shop)
```

5、用户数据

对年龄 age 和性别 sex 以及用户会员等级 uer_lv_cd 进行了 one_hot 编码，用-1 填充年龄的空值，用 2 填充性别的空值，删除注册时间，注册时间长短对用户选择商品的影响较小，删除省市县数据，虽然有某地区对某种商品需求较大的情况，不考虑。

```
# 所有用户特征
# 注册时间，认为对结果影响较小？毕竟购买行为是需求驱动的，有需求不管注册多久还是会购买
def get_basic_uesr_feat():
    user = pd.read_csv(user_path)
    user['age'] = user['age'].fillna(-1)
    user['sex'] = user['sex'].fillna(2)

    age_dummies = pd.get_dummies(user['age'], prefix='age')
    sex_dummies = pd.get_dummies(user['sex'], prefix='sex')
    user_lv_dummies = pd.get_dummies(user['user_lv_cd'], prefix='user_lv_cd')
    user = pd.concat([user[['user_id','city_level']], age_dummies, sex_dummies, user_lv_dummies], axis=1)

    return (user)
```

将所有特征合并：

获取 end_date 之前按时间窗口 days_window 划分的时间区间内的行为特征，考虑到离当前时间 end_date 越远时间区间内用户的行为特征对接下来几天的行为影响程度越小，将时间窗口设为 days_window = (1, 3, 7, 14, 30)，32G 内存只能允许处理前 30 天数据，如果条件允许，可以把前两个月（60 天）的数据也加进来。

- 获取特征子表

```
user = gf.get_basic_uesr_feat()
```

```
    product = gf.get_basic_product_feat()
    comment = gf.get_basic_comment_feat(end_date,days)
    others_action = gf.get_others_action_feat(end_date,days)
    user_order = gf.get_user_order_ratio(end_date,days)
    product_conv = gf.get_product_conversion(end_date,days)
    shop = gf.get_basic_shop_feat()
```

- 以用户行为表为中心根据用户 id 或商品 id 或商店 id 等关键值连接其他子表（尝试了很多次，以最大的用户行为表去合并是可行的，其他方案都会出现内存不足的情况），按照用户表 -> 商品表 -> 店铺表的顺序合并。

```
actions = gf.get_action_feat(end_date, days_window)
    actions = pd.merge(actions, others_action, how='left', on=['user_id', 'sku_id'])
    actions = pd.merge(actions, user, how='left', on='user_id')
    actions = pd.merge(actions, user_order, how='left', on='user_id')
    actions = pd.merge(actions, product, how='left', on='sku_id')
```

- 统计“用户-商店”对出现的次数，评估商店热度

```
# 增加（用户，商店）对出现次数的特征
    shop_time = actions.groupby(['user_id','shop_id']).size().reset_index(name='user_shop_time')
    actions = pd.merge(actions, shop_time, how='left',on=['user_id','shop_id'])
```

- 最后合并商店数据和评论数据，对合并过程中产生的 na 值填充 0

```
actions = pd.merge(actions, product_conv, how='left', on='sku_id')
   actions = pd.merge(actions, shop,how='left',on='shop_id')
   print('最后一个表合并之前:',end='\t')
   print(actions.shape)
   actions = pd.merge(actions, comment, how='left', on='sku_id')
   print('最后一个表合并之后:',end='\t')
   print(actions.shape)

   actions = actions.fillna(0)
```

正负样本平衡：

将所有特征合并之后，发现样本集中正负样本相差实在太大（比如正样本：25000，负样本 18000000），考虑稍微把正负样本的相差比例缩小，实践中先随机删除一定数量的负样本（比如 400000 行，删除太多影响模型训练），再将 5 个正样本集分成 20 个子正样本集随机插入样本集增加正样本集。

- 随机删除负样本集

```
def sub_neg_sample(label,train_data):
    # 负样本索引
    label_neg_index=label[label==0].index.tolist()
    drop_num=400000
    neg_index_drop=random.sample(label_neg_index,drop_num)

    label.drop(neg_index_drop,inplace=True)
    train_data.drop(neg_index_drop,inplace=True)

    return (label,train_data)
```

- 随机插入子正样本集

```
def add_pos_sample(label,train_data):
    # 获得正样本索引
    label_pos_index=label.index[label['label']==1].tolist()
    label_pos_select=label.loc[label_pos_index]
    train_data_selct=train_data.loc[label_pos_index]
    repeat_time=5
    slices=20
    label_list=split_dataframe(label_pos_select,slices)
    train_data_list=split_dataframe(train_data_selct,slices)
    for j in np.arange(repeat_time):
        print('第%d 次插入正样本集。'%(j))
        for (sub_label,sub_train_data,i) in zip(label_list,train_data_list,np.arange(slices+1)):
            print('随机插入了第%d 个子正样本集'%(i+1))
            # 随机选择一个要插入的位置
            index=random.randint(1,label.shape[0])
            # 分成上下两部分
            label_above=label[:index]
            label_below=label[index+1:]
            train_data_above=train_data[:index]
            train_data_below=train_data[index+1:]
            # 合并成新数据帧
            label=pd.concat([label_above,sub_label,label_below],ignore_index=True)
            train_data=pd.concat([train_data_above,sub_train_data,train_data_below],ignore_index=True)

    return (label,train_data)
```

模型训练：

train_end_date 之前一个月的数据进行特征提取，test_end_date 之前一周的数据进行用户行为判断标记（train_end_date 和 test_end_date 刚好间隔一周），这部分之前做过滑动时间窗口的尝试，滑动长度一个月（没有包括春节数据）步长为 1 天，效果跟只取 4 月 16 日之前一个月数据效果差不多，并且滑动窗口运行时间太长，时间成本较大，均衡下来还是考虑只取前一个月的数据。

相关参数设置：

```
X_train, X_test, y_train, y_test = train_test_split(train_data,
                                                    label,
                                                    test_size=0.2,
                                                    random_state=0)
    dtrain = xgb.DMatrix(X_train, label = y_train)
    dtest = xgb.DMatrix(X_test, label = y_test)
    param = {'learning_rate': 0.1, 'n_estimators': 1000, 'max_depth': 3,  # 此处用函数使用过 max_depth 的取值，为 3 时得分最高
             'min_child_weight': 5, 'gamma': 0, 'subsample': 1.0, 'colsample_bytree': 0.8,
             'scale_pos_weight': 1, 'eta': 0.05, 'silent': 1, 'objective': 'binary:logistic'}
    num_round = 400
    param['nthread'] = 30
    param['eval_metric'] = 'auc'
    param_list = list(param.items())
    param_list += [('eval_metric', 'logloss')]
    evallist = [(dtest, 'eval'), (dtrain, 'train')]
    bst = xgb.train(param_list, dtrain, num_round, evallist, verbose_eval=49)
```

预测：

sub_end_date 之前一个月的数据进行特征提取。

结果获取：

模型预测的是 sub_end_date 之后一周（用户-商品-商店）对发生的概率。将获得的接受概率下的“用户-商品-商店”对（user_id, sku_id, shop_id）与 jdata_product 数据表连接，查找出商品对应的品类。

```
def get_pred_result(pred,acceptance):
    pred_accept=pred[pred['label']>=acceptance]
    pred_accept.to_csv('result_%s.csv'%(acceptance),index=False)
    pred_accept=pred_accept[['user_id','sku_id','shop_id']]
    product=pd.read_csv('jdata_product.csv')
    pred_result=pd.merge(pred_accept,product,on=['sku_id','shop_id'])
```

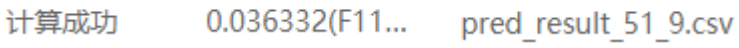
```
    pred_result=pred_result[['user_id','cate','shop_id']]

    pred_result=pred_result.astype(int)
    pred_result=pred_result.drop_duplicates()

    pred_result.to_csv('pred_result.csv',index=False)
```

模型成绩：

xgboost 模型参数调优之前成绩：



xgboost 模型参数调优之后最好成绩：