高潜用户购买画像

一、课前准备

1.1 熟悉Python的数据分析库numpy、pandas和scikit算法库

1. 2 熟悉特征工程和XGB算法

- 以京东商城真实的用户、商品和行为数据（脱敏后）为基础，通过数据挖掘的技术和机器学习的算法，构建用户购买商品的预测模型，输出高潜用户和目标商品的匹配结果，为精准营销提供高质量的目标群体。目标：使用电商多个品类下商品的历史销售数据，构建算法模型，预测用户在未来5天内，对某个目标品类下商品的购买意向。
- 数据集：
- 这里涉及到的数据集是最新的数据集：
- Data_User.csv 用户数据集 105,321个用户
- Data_Comment.csv 商品评论 558,552条记录
- Data_Product.csv 预测商品集合 24,187条记录
- Data_Action_201602.csv 2月份行为交互记录 11,485,424条记录
- Data_Action_201603.csv 3月份行为交互记录 25,916,378条记录
- Data_Action_201604.csv 4月份行为交互记录 13,199,934条记录

Data_User表说明

列名称	列名称解释	列名称说明
user_id	用户ID	脱敏
age	年龄段	-1表示未知
sex	性别	0表示男，1表示女，2表示保密
user_lv_cd	用户等级	有顺序的级别枚举，越高级别数字越大
user_reg_tm	用户注册日期	用天来表示

Data_Comment表说明

列名称	列名称解释	列名称说明
sku_id	商品编号	脱敏
a1	属性1	枚举，-1表示未知
a2	属性2	枚举，-1表示未知
a3	属性3	枚举，-1表示未知
cate	品类ID	脱敏
brand	品牌ID	脱敏

Data_Product表说明

列名称	列名称解释	列名称说明
dt	截止到时间	粒度到天
sku_id	商品编号	脱敏
comment_num	累计评论数分段	0表示无评论，1表示有1条评论， 2表示有2-10条评论， 3表示有11-50条评论 4表示大于50条评论
has_bad_comment	是否有差评	0表示无，1表示有
bad_comment_rate	差评率	差评数占总评论数的比重

Data_Action表说明

列名称	列名称解释	列名称说明
user_id	用户编号	脱敏
sku_id	商品编号	脱敏
time	行为时间	秒
model_id	点击模块编号，如果是点击	脱敏
type	行为类型	1:浏览 2:加入购物车 3:购物车删除 4:下单 5:关注 6:点击
cate	品类ID	脱敏
brand	品牌ID	脱敏

三、课堂目标

（一）.数据清洗与格式转换
1. 数据集完整性验证
2. 数据集中是否存在缺失值
3. 数据集中各特征数值应该如何处理
4. 哪些数据是我们想要的，哪些是可以过滤掉的
5. 将有价值数据信息做成新的数据源
6. 去除无行为交互的商品和用户
7. 去掉浏览量很大而购买量很少的用户(惰性用户或爬虫用户)

（二）.EDA/探索性数据分析
1. 掌握各个特征的含义
2. 观察数据有哪些特点，是否可利用来建模
3. 可视化展示便于分析
4. 用户的购买意向是否随着时间等因素变化

（三）.特征提取
1. 基于清洗后的数据集哪些特征是有价值
2. 分别对用户与商品以及其之间构成的行为进行特征提取
3. 行为因素中哪些是核心？如何提取？
4. 瞬时行为特征or累计行为特征

（四）.模型建立与预测
1. 使用机器学习算法进行预测
2. 参数设置与调节
3. 数据集切分

四、知识要点

4.1 数据集验证

4.1.1 检查Data_User中的用户和Data_Action中的用户是否一致

```
%matplotlib inline
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore') #忽视
```

```
# test sample
df1 = pd.DataFrame({'Sku':['a','a','e','c'],'Action':[1,1,2,3]})
df2 = pd.DataFrame({'Sku':['a','b','f']})
df3 = pd.DataFrame({'Sku':['a','b','d']})
df4 = pd.DataFrame({'Sku':['a','b','c','d']})
print (pd.merge(df2,df1))
#print (pd.merge(df1,df2))
#print (pd.merge(df3,df1))
#print (pd.merge(df4,df1))
#print (pd.merge(df1,df3))
```

```
  Sku  Action
0   a       1
1   a       1
```

```
def user_action_check():
    df_user = pd.read_csv('data/Data_User.csv',encoding='gbk') # 读入数据
    df_sku = df_user.loc[:,'user_id'].to_frame() # series将数组转换为DataFrame格式

    df_month2 = pd.read_csv('data/Data_Action_201602.csv',encoding='gbk')
    print ('Is action of Feb. from User file? ', len(df_month2) == len(pd.merge(df_sku,df_month2)))




    df_month3 = pd.read_csv('data/Data_Action_201603.csv',encoding='gbk')
    print ('Is action of Mar. from User file? ', len(df_month3) == len(pd.merge(df_sku,df_month3)))
    df_month4 = pd.read_csv('data/Data_Action_201604.csv',encoding='gbk')
    print ('Is action of Apr. from User file? ', len(df_month4) == len(pd.merge(df_sku,df_month4)))

user_action_check()

# 2、3、4月份的数据是否来自User文件
```

```
Is action of Feb. from User file?  True
Is action of Mar. from User file?  True
Is action of Apr. from User file?  True
```

4.1.2 检查是否有重复记录

- 查看各个数据文件中完全重复的记录,可能解释是重复数据是有意义的，比如用户同时购买多件商品，同时添加多个数量的商品到购物车等

```
def deduplicate(filepath, filename, newpath):
    df_file = pd.read_csv(filepath,encoding='gbk')  # 读入数据
    before = df_file.shape[0] # 样本的行号/长度
    df_file.drop_duplicates(inplace=True) # 去重复值
    after = df_file.shape[0] # 再查看有多少样本数/长度
    n_dup = before-after # 前后样本数的差值
    print ('No. of duplicate records for ' + filename + ' is: ' + str(n_dup))
    if n_dup != 0:
        df_file.to_csv(newpath, index=None)
    else:
        print ('no duplicate records in ' + filename)
```

```
# deduplicate('data/Data_Action_201602.csv', 'Feb. action', 'data/Data_Action_201602_dedup.csv')
deduplicate('data/Data_Action_201603.csv', 'Mar. action', 'data/Data_Action_201603_dedup.csv')
deduplicate('data/Data_Action_201604.csv', 'Feb. action', 'data/Data_Action_201604_dedup.csv')
deduplicate('data/Data_Comment.csv', 'Comment', 'data/Data_Comment_dedup.csv')
deduplicate('data/Data_Product.csv', 'Product', 'data/Data_Product_dedup.csv')
deduplicate('data/Data_User.csv', 'User', 'data/Data_User_dedup.csv')

# 第一行重复数据有7085038, 说明同一个商品买了多个
# 第二行重复数据有3672710
# 第三行重复数据为0
```

```
No. of duplicate records for Mar. action is: 7085038
No. of duplicate records for Feb. action is: 3672710
No. of duplicate records for Comment is: 0
no duplicate records in Comment
No. of duplicate records for Product is: 0
no duplicate records in Product
No. of duplicate records for User is: 0
no duplicate records in User
```

```
df_month2 = pd.read_csv('data/Data_Action_201602.csv',encoding='gbk')
IsDuplicated = df_month2.duplicated()  # 检查重复值
df_d=df_month2[IsDuplicated]
df_d.groupby('type').count()   #发现重复数据大多数都是由于浏览 (1), 或者点击(6)产生
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time	model_id	cate	brand
type						
1	2176378	2176378	2176378	0	2176378	2176378
2	636	636	636	0	636	636
3	1464	1464	1464	0	1464	1464
4	37	37	37	0	37	37
5	1981	1981	1981	0	1981	1981
6	575597	575597	575597	545054	575597	575597

4.1.3 检查是否存在注册时间在2016年-4月-15号之后的用户

```
import pandas as pd
df_user = pd.read_csv('data/Data_User.csv',encoding='gbk')
df_user['user_reg_tm']=pd.to_datetime(df_user['user_reg_tm']) # 找到用户注册时间这一列
df_user.loc[df_user.user_reg_tm  >= '2016-4-15']
#由于注册时间是系统错误造成，如果行为数据中没有在4月15号之后的数据的话，那么说明这些用户还是正常用户，并不需要删除。
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age	sex	user_lv_cd	user_reg_tm
7457	207458	-1.0	2.0	1	2016-04-15
7463	207464	3.0	2.0	2	2016-04-15
7467	207468	4.0	2.0	3	2016-04-15
7472	207473	-1.0	2.0	1	2016-04-15
7482	207483	3.0	2.0	3	2016-04-15
7492	207493	2.0	2.0	3	2016-04-15
7493	207494	2.0	2.0	3	2016-04-15
7503	207504	2.0	2.0	4	2016-04-15
7510	207511	5.0	2.0	5	2016-04-15
7512	207513	-1.0	2.0	1	2016-04-15
7518	207519	3.0	2.0	2	2016-04-15
7521	207522	3.0	0.0	3	2016-04-15
7525	207526	-1.0	2.0	3	2016-04-15
7533	207534	-1.0	2.0	1	2016-04-15
7543	207544	3.0	2.0	3	2016-04-15
7544	207545	-1.0	2.0	1	2016-04-15
7551	207552	3.0	2.0	3	2016-04-15
7553	207554	2.0	2.0	4	2016-04-15
8545	208546	2.0	0.0	2	2016-04-29
9394	209395	2.0	1.0	2	2016-05-11
10362	210363	6.0	2.0	2	2016-05-24
10367	210368	-1.0	2.0	1	2016-05-24
11019	211020	4.0	2.0	3	2016-06-06
12014	212015	4.0	2.0	2	2016-07-05
13850	213851	3.0	2.0	3	2016-09-11
14542	214543	-1.0	2.0	1	2016-10-05
16746	216747	2.0	2.0	1	2016-11-25

```
df_month = pd.read_csv('data/Data_Action_201604.csv')
df_month['time'] = pd.to_datetime(df_month['time'])
df_month.loc[df_month.time >= '2016-4-16']
# 结论：说明用户没有异常操作数据，所以这一批用户不删除
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time	model_id	type	cate	brand

4.1.4 行为数据中的user_id为浮点型，进行INT类型转换

- 因为2、3、4月份的数据集中用USERID，因此要转换为INT类型

```
import pandas as pd
df_month = pd.read_csv('data/Data_Action_201602.csv',encoding='gbk')
df_month['user_id'] = df_month['user_id'].apply(lambda x:int(x))
print (df_month['user_id'].dtype)
df_month.to_csv('data/Data_Action_201602.csv',index=None)
df_month = pd.read_csv('data/Data_Action_201603.csv',encoding='gbk')
df_month['user_id'] = df_month['user_id'].apply(lambda x:int(x))
print (df_month['user_id'].dtype)
df_month.to_csv('data/Data_Action_201603.csv',index=None)
df_month = pd.read_csv('data/Data_Action_201604.csv',encoding='gbk')
df_month['user_id'] = df_month['user_id'].apply(lambda x:int(x))
print (df_month['user_id'].dtype)
df_month.to_csv('data/Data_Action_201604.csv',index=None)
```

```
int64
int64
int64
```

4.1.5 年龄区间的处理

- 把年龄映射成值

```
import pandas as pd
df_user = pd.read_csv('data/Data_User.csv',encoding='gbk')

def tranAge(x):
    if x == u'15岁以下':
        x='1'
    elif x==u'16-25岁':
        x='2'
    elif x==u'26-35岁':
        x='3'
    elif x==u'36-45岁':
        x='4'
    elif x==u'46-55岁':
        x='5'
    elif x==u'56岁以上':
        x='6'
    return x
df_user['age']=df_user['age'].apply(tranAge)
print (df_user.groupby(df_user['age']).count()) # 有14412个没有透露性别，在年龄值为3时候最多，属于"26-35岁"
df_user.to_csv('data/Data_User.csv',index=None)
```

```
      user_id    sex  user_lv_cd  user_reg_tm
age
-1.0    14412  14412       14412        14412
 1.0        7      7           7            7
 2.0     8797   8797        8797         8797
 3.0    46570  46570       46570        46570
 4.0    30336  30336       30336        30336
 5.0     3325   3325        3325         3325
 6.0     1871   1871        1871         1871
```

user_table

- user_table特征包括:
- user_id(用户id),age(年龄),sex(性别),
- user_lv_cd(用户级别),browse_num(浏览数),
- addcart_num(加购数),delcart_num(删购数),
- buy_num(购买数),favor_num(收藏数),
- click_num(点击数),buy_addcart_ratio(购买转化率),
- buy_browse_ratio(购买浏览转化率),
- buy_click_ratio(购买点击转化率),
- buy_favor_ratio(购买收藏转化率)

item_table特征包括:

- sku_id(商品id),attr1,attr2,
- attr3,cate,brand,browse_num,
- addcart_num,delcart_num,
- buy_num,favor_num,click_num,
- buy_addcart_ratio,buy_browse_ratio,
- buy_click_ratio,buy_favor_ratio,
- comment_num(评论数),

- has_bad_comment(是否有差评),
- bad_comment_rate(差评率)

4.1.6 构建User_table

```
#重定义文件名
ACTION_201602_FILE = "data/Data_Action_201602.csv"
ACTION_201603_FILE = "data/Data_Action_201603.csv"
ACTION_201604_FILE = "data/Data_Action_201604.csv"
COMMENT_FILE = "data/Data_Comment.csv"
PRODUCT_FILE = "data/Data_Product.csv"
USER_FILE = "data/Data_User.csv"
USER_TABLE_FILE = "data/User_table.csv"
ITEM_TABLE_FILE = "data/Item_table.csv"
```

```
# 导入相关包
import pandas as pd
import numpy as np
from collections import Counter
```

```
# 功能函数: 对每一个user分组的数据进行统计
def add_type_count(group):
    behavior_type = group.type.astype(int)
    # 统计用户行为类别
    type_cnt = Counter(behavior_type)
    # 1: 浏览 2: 加购 3: 删除
    # 4: 购买 5: 收藏 6: 点击
    group['browse_num'] = type_cnt[1]
    group['addcart_num'] = type_cnt[2]
    group['delcart_num'] = type_cnt[3]
    group['buy_num'] = type_cnt[4]
    group['favor_num'] = type_cnt[5]
    group['click_num'] = type_cnt[6]

    return group[['user_id', 'browse_num', 'addcart_num',
                  'delcart_num', 'buy_num', 'favor_num',
                  'click_num']]
```

```
#对action数据进行统计
#因为由于用户行为数据量较大,一次性读入可能造成内存错误(Memory Error)
#因而使用pandas的分块(chunk)读取.根据自己调节chunk_size大小

def get_from_action_data(fname, chunk_size=50000):
    reader = pd.read_csv(fname, header=0, iterator=True,encoding='gbk')
    chunks = []
    loop = True
    while loop:
        try:
            # 只读取user_id和type两个字段
            chunk = reader.get_chunk(chunk_size)[["user_id", "type"]]
            chunks.append(chunk)
        except StopIteration: # 读完了就停止
            loop = False
            print("Iteration is stopped")
    # 将块拼接为pandas dataframe格式
    df_ac = pd.concat(chunks, ignore_index=True)
    # 按user_id分组, 对每一组进行统计, as_index 表示无索引形式返回数据
    df_ac = df_ac.groupby(['user_id'], as_index=False).apply(add_type_count)
    # 将重复的行丢弃
    df_ac = df_ac.drop_duplicates('user_id')

    return df_ac
```

```
# 将各个action数据的统计量进行聚合
def merge_action_data():
    df_ac = []
    df_ac.append(get_from_action_data(fname=ACTION_201602_FILE))
    df_ac.append(get_from_action_data(fname=ACTION_201603_FILE))
    df_ac.append(get_from_action_data(fname=ACTION_201604_FILE))

    df_ac = pd.concat(df_ac, ignore_index=True)
    # 用户在不同action表中统计量求和
    df_ac = df_ac.groupby(['user_id'], as_index=False).sum()
    #  构造转化率字段
    df_ac['buy_addcart_ratio'] = df_ac['buy_num'] / df_ac['addcart_num'] # 加了多少次购物车才买, 购买率
    df_ac['buy_browse_ratio'] = df_ac['buy_num'] / df_ac['browse_num'] # 浏览了多少次才买
```

```python
    df_ac['buy_click_ratio'] = df_ac['buy_num'] / df_ac['click_num'] # 点击了多少次才买
    df_ac['buy_favor_ratio'] = df_ac['buy_num'] / df_ac['favor_num'] # 喜欢了多少个才买

    # 将大于 1 的转化率字段置为 1 (100%)，确保数据没有问题
    df_ac.ix[df_ac['buy_addcart_ratio'] > 1., 'buy_addcart_ratio'] = 1.
    df_ac.ix[df_ac['buy_browse_ratio'] > 1., 'buy_browse_ratio'] = 1.
    df_ac.ix[df_ac['buy_click_ratio'] > 1., 'buy_click_ratio'] = 1.
    df_ac.ix[df_ac['buy_favor_ratio'] > 1., 'buy_favor_ratio'] = 1.

    return df_ac
```

```python
#  从Data_User表中抽取需要的字段
def get_from_jdata_user():
    df_usr = pd.read_csv(USER_FILE, header=0)
    df_usr = df_usr[["user_id", "age", "sex", "user_lv_cd"]]
    return df_usr
```

```python
# 执行目的是得到大表
user_base = get_from_jdata_user()
user_behavior = merge_action_data()
```

```
Iteration is stopped
Iteration is stopped
Iteration is stopped
```

```python
# 连接成一张表，类似于SQL的左连接(left join)
user_behavior = pd.merge(user_base, user_behavior, on=['user_id'], how='left')
# 保存中间结果为user_table.csv
user_behavior.to_csv(USER_TABLE_FILE, index=False)
```

```python
user_table = pd.read_csv(USER_TABLE_FILE)
user_table.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age	sex	user_lv_cd	browse_num	addcart_num	delcart_num	buy_num	favor_num	click_num	buy_addcart_ratio	buy_brov
0	200001	6.0	2.0	5	212.0	22.0	13.0	1.0	0.0	414.0	0.045455	0.004717
1	200002	-1.0	0.0	1	238.0	1.0	0.0	0.0	0.0	484.0	0.000000	0.000000
2	200003	4.0	1.0	4	221.0	4.0	1.0	0.0	1.0	420.0	0.000000	0.000000
3	200004	-1.0	2.0	1	52.0	0.0	0.0	0.0	0.0	61.0	NaN	0.000000
4	200005	2.0	0.0	4	106.0	2.0	3.0	1.0	2.0	161.0	0.500000	0.009434

4.1.7 构建Item_table

- 跟上面一样

```python
#定义文件名
ACTION_201602_FILE = "data/Data_Action_201602.csv"
ACTION_201603_FILE = "data/Data_Action_201603.csv"
ACTION_201604_FILE = "data/Data_Action_201604.csv"
COMMENT_FILE = "data/Data_Comment.csv"
PRODUCT_FILE = "data/Data_Product.csv"
USER_FILE = "data/Data_User.csv"
USER_TABLE_FILE = "data/User_table.csv"
ITEM_TABLE_FILE = "data/Item_table.csv"
```

```python
# 导入相关包
import pandas as pd
import numpy as np
from collections import Counter
```

```python
# 读取Product中商品
def get_from_jdata_product():
    df_item = pd.read_csv(PRODUCT_FILE, header=0,encoding='gbk')
    return df_item
```

```python
# 对每一个商品分组进行统计
def add_type_count(group):
    behavior_type = group.type.astype(int)
    type_cnt = Counter(behavior_type)

    group['browse_num'] = type_cnt[1]
    group['addcart_num'] = type_cnt[2]
    group['delcart_num'] = type_cnt[3]
    group['buy_num'] = type_cnt[4]
    group['favor_num'] = type_cnt[5]
    group['click_num'] = type_cnt[6]

    return group[['sku_id', 'browse_num', 'addcart_num',
                  'delcart_num', 'buy_num', 'favor_num',
                  'click_num']]
```

```python
#对action中的数据进行统计
def get_from_action_data(fname, chunk_size=50000):
    reader = pd.read_csv(fname, header=0, iterator=True)
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(chunk_size)[["sku_id", "type"]]
            chunks.append(chunk)
        except StopIteration:
            loop = False
            print("Iteration is stopped")

    df_ac = pd.concat(chunks, ignore_index=True)

    df_ac = df_ac.groupby(['sku_id'], as_index=False).apply(add_type_count)
    # Select unique row
    df_ac = df_ac.drop_duplicates('sku_id')

    return df_ac
```

```python
# 获取评论中的商品数据,如果存在某一个商品有两个日期的评论,我们取最晚的那一个
def get_from_jdata_comment():
    df_cmt = pd.read_csv(COMMENT_FILE, header=0)
    df_cmt['dt'] = pd.to_datetime(df_cmt['dt'])
    # find latest comment index
    idx = df_cmt.groupby(['sku_id'])['dt'].transform(max) == df_cmt['dt']
    df_cmt = df_cmt[idx]

    return df_cmt[['sku_id', 'comment_num',
                   'has_bad_comment', 'bad_comment_rate']]
```

```python
def merge_action_data():
    df_ac = []
    df_ac.append(get_from_action_data(fname=ACTION_201602_FILE))
    df_ac.append(get_from_action_data(fname=ACTION_201603_FILE))
    df_ac.append(get_from_action_data(fname=ACTION_201604_FILE))

    df_ac = pd.concat(df_ac, ignore_index=True)
    df_ac = df_ac.groupby(['sku_id'], as_index=False).sum()

    df_ac['buy_addcart_ratio'] = df_ac['buy_num'] / df_ac['addcart_num']
    df_ac['buy_browse_ratio'] = df_ac['buy_num'] / df_ac['browse_num']
    df_ac['buy_click_ratio'] = df_ac['buy_num'] / df_ac['click_num']
    df_ac['buy_favor_ratio'] = df_ac['buy_num'] / df_ac['favor_num']

    df_ac.ix[df_ac['buy_addcart_ratio'] > 1., 'buy_addcart_ratio'] = 1.
```

```
    df_ac.ix[df_ac['buy_browse_ratio'] > 1., 'buy_browse_ratio'] = 1.
    df_ac.ix[df_ac['buy_click_ratio'] > 1., 'buy_click_ratio'] = 1.
    df_ac.ix[df_ac['buy_favor_ratio'] > 1., 'buy_favor_ratio'] = 1.

    return df_ac
```

```
item_base = get_from_jdata_product()
item_behavior = merge_action_data()
item_comment = get_from_jdata_comment()

# SQL: left join
item_behavior = pd.merge(
    item_base, item_behavior, on=['sku_id'], how='left')
item_behavior = pd.merge(
    item_behavior, item_comment, on=['sku_id'], how='left')

item_behavior.to_csv(ITEM_TABLE_FILE, index=False)
```

```
Iteration is stopped
Iteration is stopped
Iteration is stopped
```

```
item_table = pd.read_csv(ITEM_TABLE_FILE)
item_table.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sku_id	a1	a2	a3	cate	brand	browse_num	addcart_num	delcart_num	buy_num	favor_num	click_num	buy_addcart_ratio
0	10	3	1	1	8	489	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	100002	3	2	2	8	489	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	100003	1	-1	-1	8	30	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	100006	1	2	1	8	545	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	10001	-1	1	2	8	244	NaN	NaN	NaN	NaN	NaN	NaN	NaN

4.1.8 用户清洗

```
import pandas as pd
df_user = pd.read_csv('data/User_table.csv',header=0)
pd.options.display.float_format = '{:,.3f}'.format  #输出格式设置，保留三位小数
df_user.describe()

#第一行中根据User_id统计发现有105321个用户，发现有几个用户没有age,sex字段，
#而且根据浏览、加购、删购、购买等记录却只有105180条记录，说明存在用户无任何交互记录，因此可以删除上述用户。
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age	sex	user_lv_cd	browse_num	addcart_num	delcart_num	buy_num	favor_num	click_num	b
count	105,321.000	105,318.000	105,318.000	105,321.000	105,180.000	105,180.000	105,180.000	105,180.000	105,180.000	105,180.000	7
mean	252,661.000	2.773	1.113	3.850	180.466	5.471	2.434	0.459	1.045	291.222	0
std	30,403.698	1.672	0.956	1.072	273.437	10.618	5.600	1.048	3.442	460.031	0
min	200,001.000	-1.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0
25%	226,331.000	3.000	0.000	3.000	40.000	0.000	0.000	0.000	0.000	59.000	0
50%	252,661.000	3.000	2.000	4.000	94.000	2.000	0.000	0.000	0.000	148.000	0
75%	278,991.000	4.000	2.000	5.000	212.000	6.000	3.000	1.000	0.000	342.000	0
max	305,321.000	6.000	2.000	5.000	7,605.000	369.000	231.000	50.000	99.000	15,302.000	1

```
#删除少数的3行的年龄
df_user[df_user['age'].isnull()]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age	sex	user_lv_cd	browse_num	addcart_num	delcart_num	buy_num	favor_num	click_num	buy_addcart_ratio	buy
34072	234073	nan	nan	1	32.000	6.000	4.000	1.000	0.000	41.000	0.167	0.03
38905	238906	nan	nan	1	171.000	3.000	2.000	2.000	3.000	464.000	0.667	0.01
67704	267705	nan	nan	1	342.000	18.000	8.000	0.000	0.000	743.000	0.000	0.00

```
#删除无交互记录的用户
df_naction = df_user[(df_user['browse_num'].isnull()) & (df_user['addcart_num'].isnull()) & (df_user['delcart_num'].isnull()) &
(df_user['buy_num'].isnull()) & (df_user['favor_num'].isnull()) & (df_user['click_num'].isnull())]
df_user.drop(df_naction.index,axis=0,inplace=True)
print (len(df_user))
```

```
105180
```

```
#统计无购买记录的用户
df_bzero = df_user[df_user['buy_num']==0]
#输出购买数为0的总记录数
print (len(df_bzero))
```

```
75695
```

```
#删除无购买记录的用户
df_user = df_user[df_user['buy_num']!=0]
```

```
#浏览购买转换比和点击购买转换比小于0.0005的用户为惰性用户
# 删除爬虫及惰性用户
bindex = df_user[df_user['buy_browse_ratio']<0.0005].index
print (len(bindex))
df_user.drop(bindex,axis=0,inplace=True)
```

```
90
```

点击购买转换比和点击购买转换比小于0.0005的用户为惰性用户

```
# 点击购买转换比和点击购买转换比小于0.0005的用户为惰性用户
# 删除爬虫及惰性用户
cindex = df_user[df_user['buy_click_ratio']<0.0005].index
print (len(cindex))
df_user.drop(cindex,axis=0,inplace=True)
```

```
323
```

```
df_user.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age	sex	user_lv_cd	browse_num	addcart_num	delcart_num	buy_num	favor_num	click_num	buy_ad
count	29,072.000	29,070.000	29,070.000	29,072.000	29,072.000	29,072.000	29,072.000	29,072.000	29,072.000	29,072.000	29,072.
mean	250,766.117	2.910	1.028	4.268	280.248	10.144	4.457	1.644	1.589	447.099	0.364
std	29,998.079	1.492	0.959	0.810	325.122	13.443	6.998	1.419	4.293	530.981	0.320
min	200,001.000	-1.000	0.000	1.000	1.000	0.000	0.000	1.000	0.000	0.000	0.004
25%	225,038.000	3.000	0.000	4.000	75.000	3.000	0.000	1.000	0.000	114.000	0.125
50%	249,199.500	3.000	1.000	4.000	174.000	6.000	2.000	1.000	0.000	275.000	0.250
75%	276,282.000	4.000	2.000	5.000	366.000	13.000	6.000	2.000	1.000	585.000	0.500
max	305,318.000	6.000	2.000	5.000	5,007.000	288.000	158.000	50.000	69.000	8,156.000	1.000

4.2 EDA/探索性数据分析

4.2.1 周一到周日每天购买情况

```
# 导入相关包
%matplotlib inline
# 绘图包
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
#定义文件名
ACTION_201602_FILE = "data/Data_Action_201602.csv"
ACTION_201603_FILE = "data/Data_Action_201603.csv"
ACTION_201604_FILE = "data/Data_Action_201604.csv"
COMMENT_FILE = "data/Data_Comment.csv"
PRODUCT_FILE = "data/Data_Product.csv"
USER_FILE = "data/Data_User.csv"
USER_TABLE_FILE = "data/User_table.csv"
ITEM_TABLE_FILE = "data/Item_table.csv"
```

```
# 提取购买(type=4)的下单行为数据
def get_from_action_data(fname, chunk_size=50000):
    reader = pd.read_csv(fname, header=0, iterator=True)
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(chunk_size)[
                ["user_id", "sku_id", "type", "time"]]
            chunks.append(chunk)
        except StopIteration:
            loop = False
```

```
            print("Iteration is stopped")

    df_ac = pd.concat(chunks, ignore_index=True)
    # type=4,为购买/下单
    df_ac = df_ac[df_ac['type'] == 4]

    return df_ac[["user_id", "sku_id", "time"]]
```

```
df_ac = []
df_ac.append(get_from_action_data(fname=ACTION_201602_FILE))
df_ac.append(get_from_action_data(fname=ACTION_201603_FILE))
df_ac.append(get_from_action_data(fname=ACTION_201604_FILE))
df_ac = pd.concat(df_ac, ignore_index=True)
```

```
Iteration is stopped
Iteration is stopped
Iteration is stopped
```

```
print(df_ac.dtypes) # 将time字段转换为datetime类型
```

```
user_id     int64
sku_id      int64
time       object
dtype: object
```

```
# 将time字段转换为datetime类型
df_ac['time'] = pd.to_datetime(df_ac['time'])

# 使用lambda匿名函数将时间time转换为星期(周一为1, 周日为７)
df_ac['time'] = df_ac['time'].apply(lambda x: x.weekday() + 1)
```

```
df_ac.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time
0	269365	166345	1
1	235443	36692	1
2	247689	9112	1
3	273959	102034	1
4	226791	163550	1

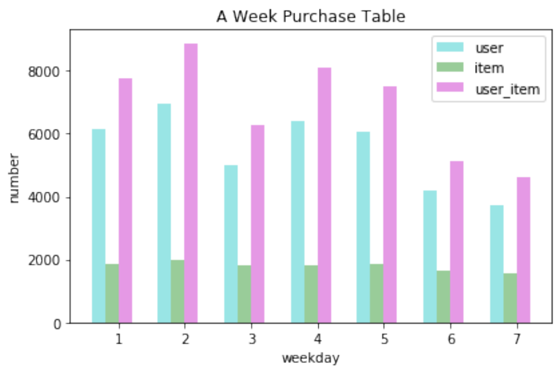
```
# 周一到周日每天购买用户个数
df_user = df_ac.groupby('time')['user_id'].nunique()
df_user = df_user.to_frame().reset_index() # DataFrame可以通过set_index方法，可以设置索引
df_user.columns = ['weekday', 'user_num']
```

```
# 周一到周日每天购买商品个数
df_item = df_ac.groupby('time')['sku_id'].nunique()
df_item = df_item.to_frame().reset_index()
df_item.columns = ['weekday', 'item_num']
```

```python
# 周一到周日每天购买记录个数
df_ui = df_ac.groupby('time', as_index=False).size()
df_ui = df_ui.to_frame().reset_index()
df_ui.columns = ['weekday', 'user_item_num']
```

```python
# 条形宽度
bar_width = 0.2
# 透明度
opacity = 0.4

plt.bar(df_user['weekday'], df_user['user_num'], bar_width,
        alpha=opacity, color='c', label='user')
plt.bar(df_item['weekday']+bar_width, df_item['item_num'],
        bar_width, alpha=opacity, color='g', label='item')
plt.bar(df_ui['weekday']+bar_width*2, df_ui['user_item_num'],
        bar_width, alpha=opacity, color='m', label='user_item')

plt.xlabel('weekday')
plt.ylabel('number')
plt.title('A Week Purchase Table')
plt.xticks(df_user['weekday'] + bar_width * 3 / 2., (1,2,3,4,5,6,7))
plt.tight_layout()
plt.legend(prop={'size':10})

# 分析：周六，周日购买量较少，配送问题
```

```
<matplotlib.legend.Legend at 0x12fc35b90>
```



4.2.2 2016年2月中各天购买量

```python
df_ac = get_from_action_data(fname=ACTION_201602_FILE)

# 将time字段转换为datetime类型并使用lambda匿名函数将时间time转换为天
df_ac['time'] = pd.to_datetime(df_ac['time']).apply(lambda x: x.day)
```

```
Iteration is stopped
```

```python
df_ac.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time
351	269365	166345	1
649	235443	36692	1
980	247689	9112	1
1719	273959	102034	1
2153	226791	163550	1

```
df_ac.tail()
```

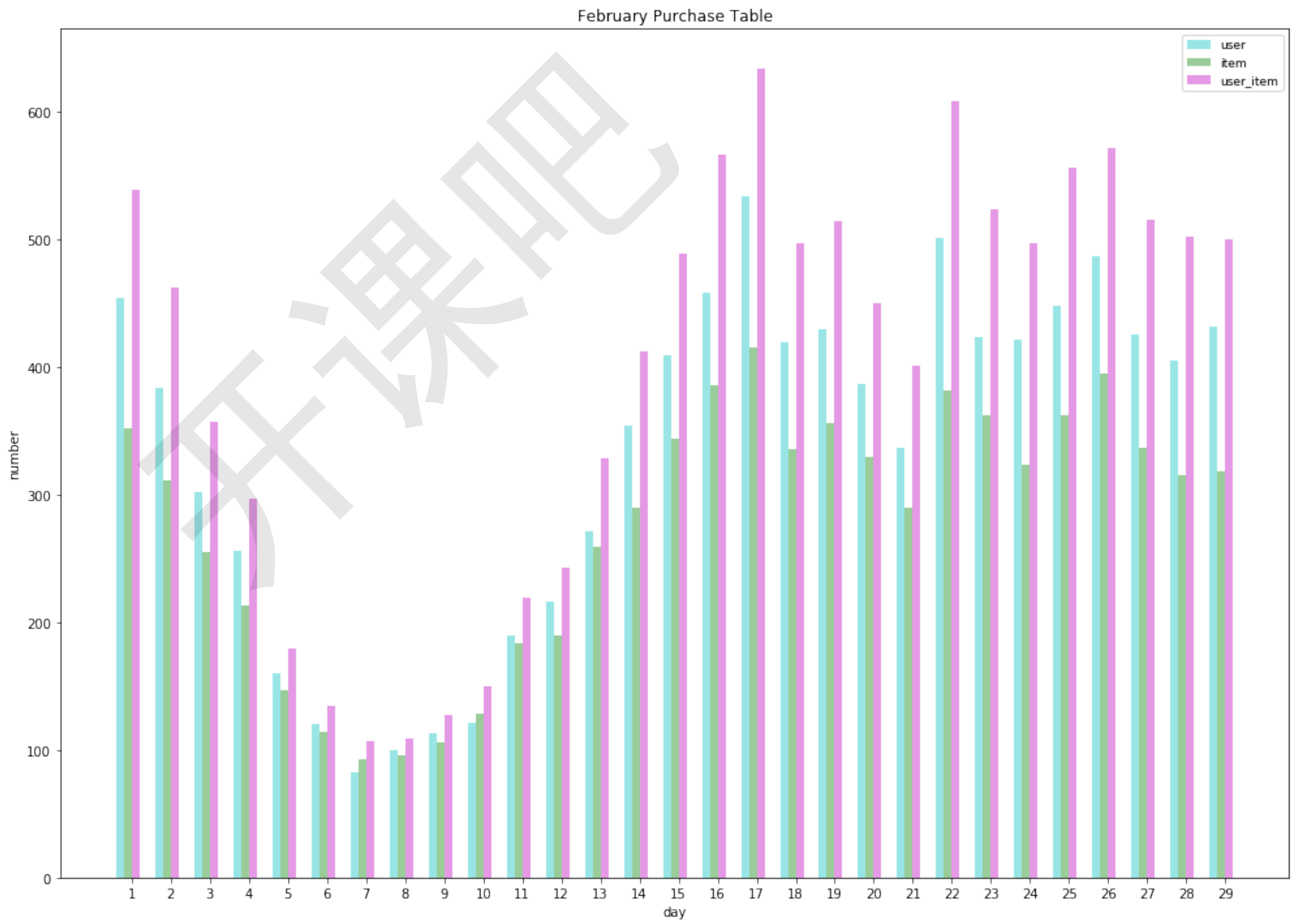
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time
11464511	256461	126092	29
11470852	224347	137636	29
11478541	300214	102335	29
11480871	213442	48000	29
11483928	228994	165190	29

```
df_user = df_ac.groupby('time')['user_id'].nunique()
df_user = df_user.to_frame().reset_index()
df_user.columns = ['day', 'user_num']

df_item = df_ac.groupby('time')['sku_id'].nunique()
df_item = df_item.to_frame().reset_index()
df_item.columns = ['day', 'item_num']

df_ui = df_ac.groupby('time', as_index=False).size()
df_ui = df_ui.to_frame().reset_index()
df_ui.columns = ['day', 'user_item_num']
```

```
# 条形宽度
bar_width = 0.2
# 透明度
opacity = 0.4
# 天数
day_range = range(1,len(df_user['day']) + 1, 1)
# 设置图片大小
plt.figure(figsize=(14,10))

plt.bar(df_user['day'], df_user['user_num'], bar_width,
        alpha=opacity, color='c', label='user')
plt.bar(df_item['day']+bar_width, df_item['item_num'],
        bar_width, alpha=opacity, color='g', label='item')
plt.bar(df_ui['day']+bar_width*2, df_ui['user_item_num'],
        bar_width, alpha=opacity, color='m', label='user_item')

plt.xlabel('day')
plt.ylabel('number')
plt.title('February Purchase Table')
plt.xticks(df_user['day'] + bar_width * 3 / 2., day_range)
# plt.ylim(0, 80)
plt.tight_layout()
plt.legend(prop={'size':9})
```

```
<matplotlib.legend.Legend at 0x15a590550>
```



- 分析：2月份5,6,7,8,9,10 这几天购买量非常少，原因可能是中国农历春节，快递不营业

4.2.3 2016年3月中各天购买量

```
df_ac = get_from_action_data(fname=ACTION_201603_FILE)

# 将time字段转换为datetime类型并使用lambda匿名函数将时间time转换为天
df_ac['time'] = pd.to_datetime(df_ac['time']).apply(lambda x: x.day)
```

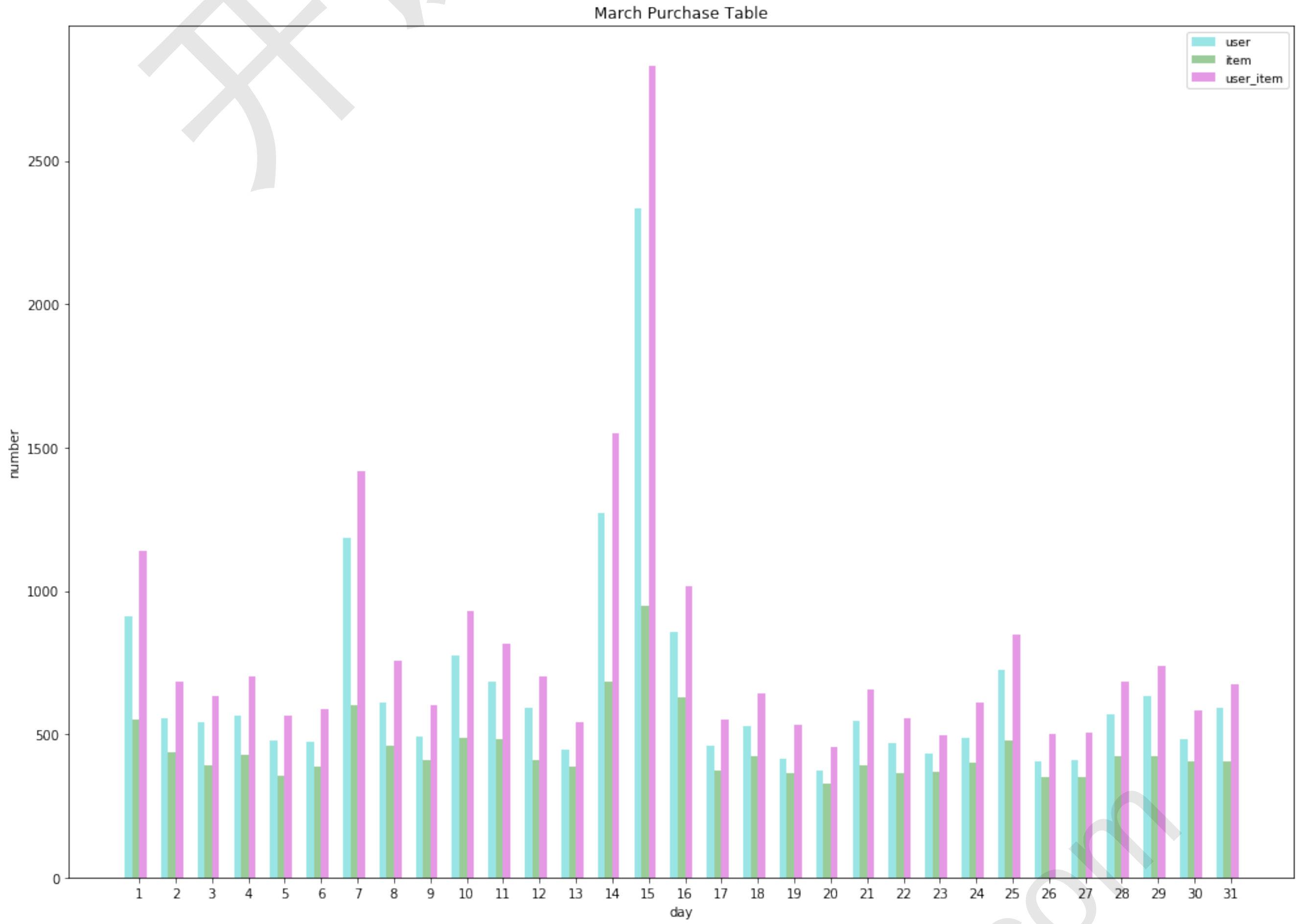
```
Iteration is stopped
```

```
df_user = df_ac.groupby('time')['user_id'].nunique()
df_user = df_user.to_frame().reset_index()
df_user.columns = ['day', 'user_num']

df_item = df_ac.groupby('time')['sku_id'].nunique()
df_item = df_item.to_frame().reset_index()
df_item.columns = ['day', 'item_num']

df_ui = df_ac.groupby('time', as_index=False).size()
df_ui = df_ui.to_frame().reset_index()
df_ui.columns = ['day', 'user_item_num']
```

```
# 条形宽度
bar_width = 0.2
# 透明度
opacity = 0.4
# 天数
day_range = range(1,len(df_user['day']) + 1, 1)
# 设置图片大小
plt.figure(figsize=(14,10))

plt.bar(df_user['day'], df_user['user_num'], bar_width,
```

```python
        alpha=opacity, color='c', label='user')
plt.bar(df_item['day']+bar_width, df_item['item_num'],
        bar_width, alpha=opacity, color='g', label='item')
plt.bar(df_ui['day']+bar_width*2, df_ui['user_item_num'],
        bar_width, alpha=opacity, color='m', label='user_item')

plt.xlabel('day')
plt.ylabel('number')
plt.title('March Purchase Table')
plt.xticks(df_user['day'] + bar_width * 3 / 2., day_range)
# plt.ylim(0, 80)
plt.tight_layout()
plt.legend(prop={'size':9})
```

```
<matplotlib.legend.Legend at 0x175b74f50>
```



4.2.4 2016年4月中各天购买量

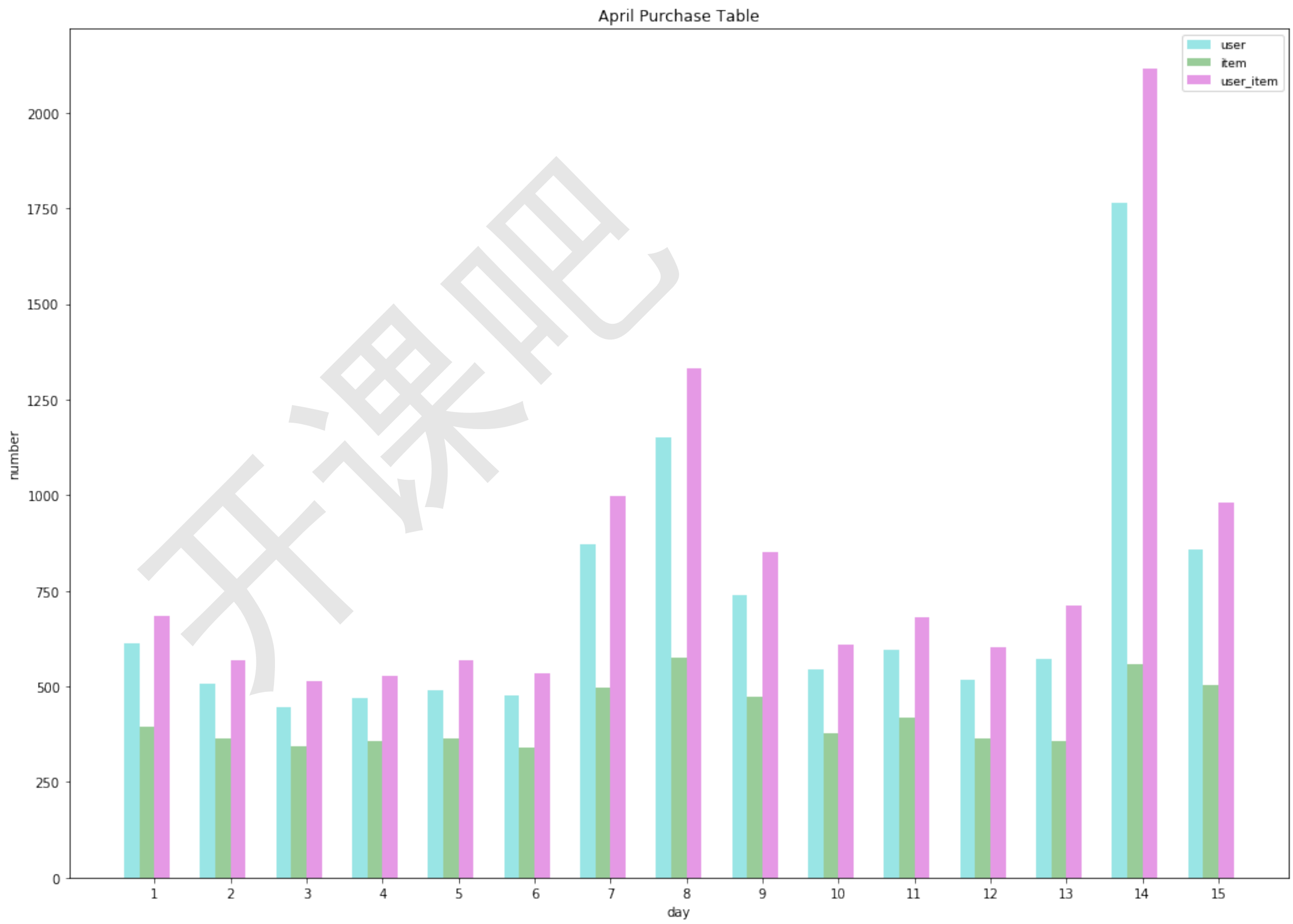
```python
df_ac = get_from_action_data(fname=ACTION_201604_FILE)

# 将time字段转换为datetime类型并使用lambda匿名函数将时间time转换为天
df_ac['time'] = pd.to_datetime(df_ac['time']).apply(lambda x: x.day)
```

```
Iteration is stopped
```

```
df_user = df_ac.groupby('time')['user_id'].nunique()
df_user = df_user.to_frame().reset_index()
df_user.columns = ['day', 'user_num']

df_item = df_ac.groupby('time')['sku_id'].nunique()
df_item = df_item.to_frame().reset_index()
df_item.columns = ['day', 'item_num']

df_ui = df_ac.groupby('time', as_index=False).size()
df_ui = df_ui.to_frame().reset_index()
df_ui.columns = ['day', 'user_item_num']
```

```
# 条形宽度
bar_width = 0.2
# 透明度
opacity = 0.4
# 天数
day_range = range(1,len(df_user['day']) + 1, 1)
# 设置图片大小
plt.figure(figsize=(14,10))

plt.bar(df_user['day'], df_user['user_num'], bar_width,
        alpha=opacity, color='c', label='user')
plt.bar(df_item['day']+bar_width, df_item['item_num'],
        bar_width, alpha=opacity, color='g', label='item')
plt.bar(df_ui['day']+bar_width*2, df_ui['user_item_num'],
        bar_width, alpha=opacity, color='m', label='user_item')

plt.xlabel('day')
plt.ylabel('number')
plt.title('April Purchase Table')
plt.xticks(df_user['day'] + bar_width * 3 / 2., day_range)
# plt.ylim(0, 80)
plt.tight_layout()
plt.legend(prop={'size':9})
```

```
<matplotlib.legend.Legend at 0x138b4db50>
```



4.2.5 商品类别销售统计

- 周一到周日各商品类别销售情况

```
# 从行为记录中提取商品类别数据
def get_from_action_data(fname, chunk_size=50000):
    reader = pd.read_csv(fname, header=0, iterator=True)
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(chunk_size)[
                ["cate", "brand", "type", "time"]]
            chunks.append(chunk)
        except StopIteration:
            loop = False
            print("Iteration is stopped")

    df_ac = pd.concat(chunks, ignore_index=True)
    # type=4,为购买
    df_ac = df_ac[df_ac['type'] == 4]

    return df_ac[["cate", "brand", "type", "time"]]
```

```
df_ac = []
df_ac.append(get_from_action_data(fname=ACTION_201602_FILE))
df_ac.append(get_from_action_data(fname=ACTION_201603_FILE))
df_ac.append(get_from_action_data(fname=ACTION_201604_FILE))
df_ac = pd.concat(df_ac, ignore_index=True)
```

```
Iteration is stopped
Iteration is stopped
Iteration is stopped
```

```python
# 将time字段转换为datetime类型
df_ac['time'] = pd.to_datetime(df_ac['time'])

# 使用lambda匿名函数将时间time转换为星期(周一为1, 周日为７)
df_ac['time'] = df_ac['time'].apply(lambda x: x.weekday() + 1)
```

```python
df_ac.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	brand	type	time
0	9	306	4	1
1	4	174	4	1
2	5	78	4	1
3	5	78	4	1
4	4	306	4	1

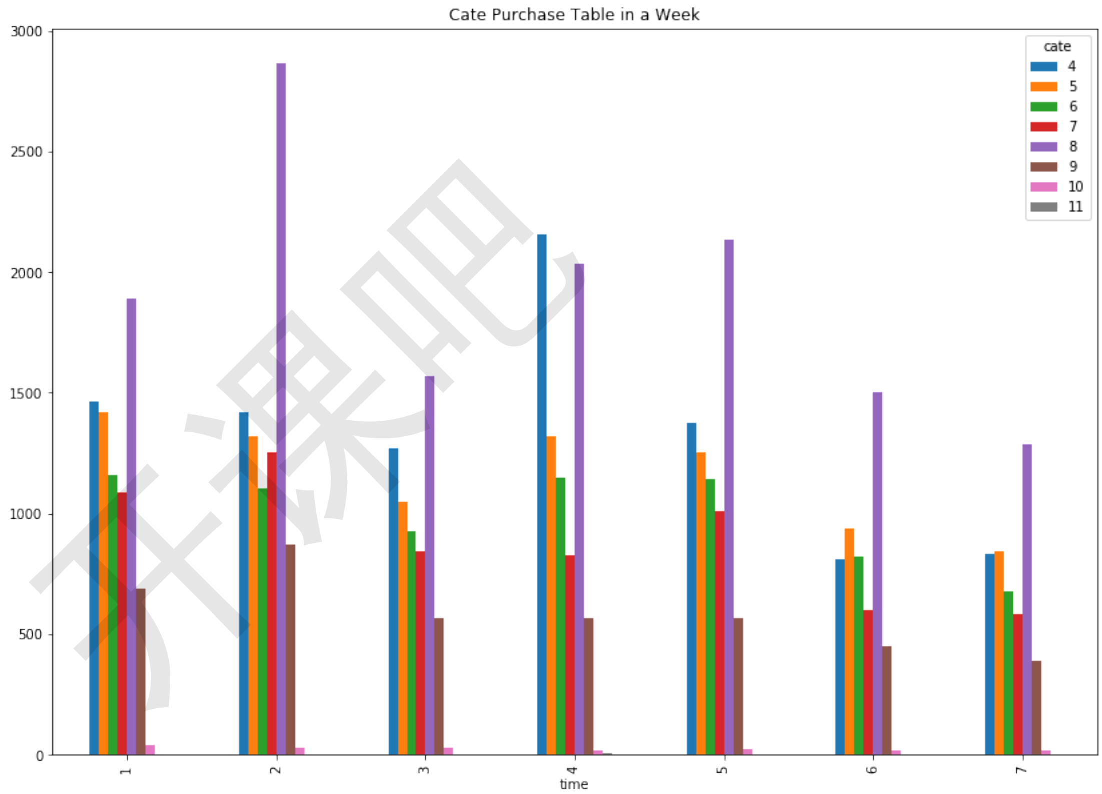
```python
# 观察有几个类别商品
df_ac.groupby(df_ac['cate']).count()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	brand	type	time
cate				
4	9326	9326	9326	9326
5	8138	8138	8138	8138
6	6982	6982	6982	6982
7	6214	6214	6214	6214
8	13281	13281	13281	13281
9	4104	4104	4104	4104
10	189	189	189	189
11	18	18	18	18

```python
# 周一到周日每天购买商品类别数量统计
df_product = df_ac['brand'].groupby([df_ac['time'],df_ac['cate']]).count()
df_product=df_product.unstack()
df_product.plot(kind='bar',title='Cate Purchase Table in a Week',figsize=(14,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x187094350>
```



- 分析：星期二买类别8的最多，星期天最少。

4.2.6 每月各类商品销售情况

```
df_ac2 = get_from_action_data(fname=ACTION_201602_FILE)

# 将time字段转换为datetime类型并使用lambda匿名函数将时间time转换为天
df_ac2['time'] = pd.to_datetime(df_ac2['time']).apply(lambda x: x.day)
df_ac3 = get_from_action_data(fname=ACTION_201603_FILE)

# 将time字段转换为datetime类型并使用lambda匿名函数将时间time转换为天
df_ac3['time'] = pd.to_datetime(df_ac3['time']).apply(lambda x: x.day)
df_ac4 = get_from_action_data(fname=ACTION_201604_FILE)

# 将time字段转换为datetime类型并使用lambda匿名函数将时间time转换为天
df_ac4['time'] = pd.to_datetime(df_ac4['time']).apply(lambda x: x.day)
```

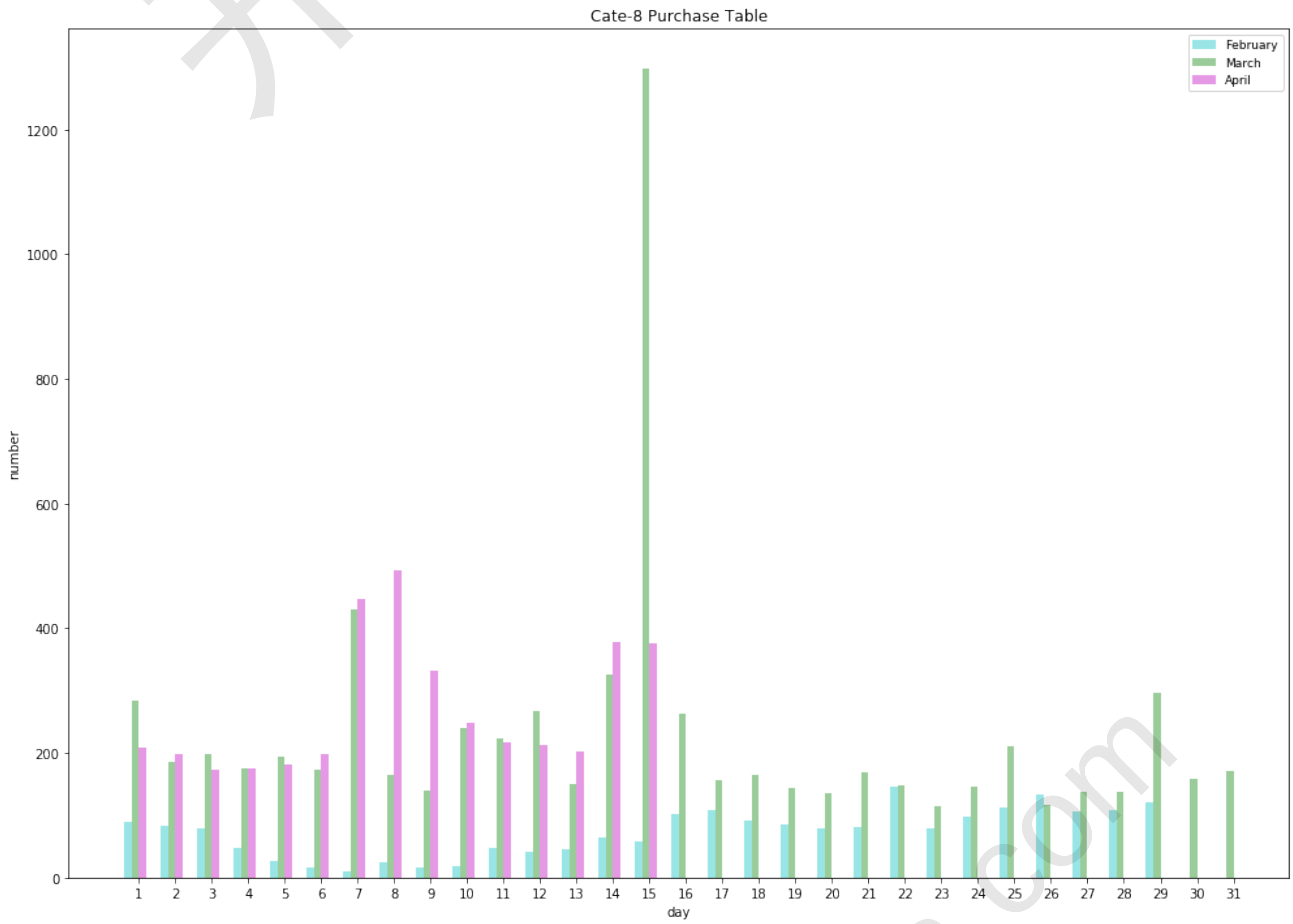
```
Iteration is stopped
Iteration is stopped
Iteration is stopped
```

```
dc_cate2 = df_ac2[df_ac2['cate']==8]
dc_cate2 = dc_cate2['brand'].groupby(dc_cate2['time']).count()
dc_cate2 = dc_cate2.to_frame().reset_index()
dc_cate2.columns = ['day', 'product_num']

dc_cate3 = df_ac3[df_ac3['cate']==8]
dc_cate3 = dc_cate3['brand'].groupby(dc_cate3['time']).count()
dc_cate3 = dc_cate3.to_frame().reset_index()
dc_cate3.columns = ['day', 'product_num']

dc_cate4 = df_ac4[df_ac4['cate']==8]
dc_cate4 = dc_cate4['brand'].groupby(dc_cate4['time']).count()
dc_cate4 = dc_cate4.to_frame().reset_index()
dc_cate4.columns = ['day', 'product_num']
```

```
# 条形宽度
bar_width = 0.2
# 透明度
opacity = 0.4
# 天数
day_range = range(1,len(dc_cate3['day']) + 1, 1)
# 设置图片大小
```

```
plt.figure(figsize=(14,10))

plt.bar(dc_cate2['day'], dc_cate2['product_num'], bar_width,
        alpha=opacity, color='c', label='February')
plt.bar(dc_cate3['day']+bar_width, dc_cate3['product_num'],
        bar_width, alpha=opacity, color='g', label='March')
plt.bar(dc_cate4['day']+bar_width*2, dc_cate4['product_num'],
        bar_width, alpha=opacity, color='m', label='April')

plt.xlabel('day')
plt.ylabel('number')
plt.title('Cate-8 Purchase Table')
plt.xticks(dc_cate3['day'] + bar_width * 3 / 2., day_range)
# plt.ylim(0, 80)
plt.tight_layout()
plt.legend(prop={'size':9})
```

```
<matplotlib.legend.Legend at 0x1725cf790>
```



- 分析：2月份对类别8商品的购买普遍偏低，3，4月份普遍偏高，3月15日购买极其多！可以对比3月份的销售记录，发现类别8将近占了3月15日总销售的一半！同时发现，3,4月份类别8销售记录在前半个月特别相似，除了4月8号，9号和3月15号。

4.2.7 查看特定用户对特定商品的的轨迹

```
def spec_ui_action_data(fname, user_id, item_id, chunk_size=100000):
    reader = pd.read_csv(fname, header=0, iterator=True)
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(chunk_size)[
                ["user_id", "sku_id", "type", "time"]]
            chunks.append(chunk)
        except StopIteration:
            loop = False
            print("Iteration is stopped")
```

```
    df_ac = pd.concat(chunks, ignore_index=True)
    df_ac = df_ac[(df_ac['user_id'] == user_id) & (df_ac['sku_id'] == item_id)]

    return df_ac
```

```
def explore_user_item_via_time():
    user_id = 266079
    item_id = 138778
    df_ac = []
    df_ac.append(spec_ui_action_data(ACTION_201602_FILE, user_id, item_id))
    df_ac.append(spec_ui_action_data(ACTION_201603_FILE, user_id, item_id))
    df_ac.append(spec_ui_action_data(ACTION_201604_FILE, user_id, item_id))
    df_ac = pd.concat(df_ac, ignore_index=False)
    print(df_ac.sort_values(by='time'))
```

```
explore_user_item_via_time()
```

```
Iteration is stopped
Iteration is stopped
Iteration is stopped
    user_id  sku_id  type                 time
0    266079  138778     1  2016-01-31 23:59:02
1    266079  138778     6  2016-01-31 23:59:03
15   266079  138778     6  2016-01-31 23:59:40
```

4.3 特征工程

```
import time
from datetime import datetime
from datetime import timedelta
import pandas as pd
import pickle
import os
import math
import numpy as np
```

```
test = pd.read_csv('data/Data_Action_201602.csv')
test[['user_id','sku_id','model_id','type','cate','brand']] = test[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')
test.dtypes
test.info() # 目的是float32位代替float64，节约内存
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11485424 entries, 0 to 11485423
Data columns (total 7 columns):
user_id     float32
sku_id      float32
time        object
model_id    float32
type        float32
cate        float32
brand       float32
dtypes: float32(6), object(1)
memory usage: 350.5+ MB
```

```
test = pd.read_csv('data/Data_Action_201602.csv')
test.dtypes
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11485424 entries, 0 to 11485423
Data columns (total 7 columns):
user_id     int64
sku_id      int64
time        object
model_id    float64
type        int64
cate        int64
brand       int64
dtypes: float64(1), int64(5), object(1)
memory usage: 613.4+ MB
```

```python
action_1_path = 'data/Data_Action_201602.csv'
action_2_path = 'data/Data_Action_201603.csv'
action_3_path = 'data/Data_Action_201604.csv'

comment_path = 'data/Data_Comment.csv'
product_path = 'data/Data_Product.csv'
user_path = 'data/Data_User.csv'


comment_date = [
    "2016-02-01", "2016-02-08", "2016-02-15", "2016-02-22", "2016-02-29",
    "2016-03-07", "2016-03-14", "2016-03-21", "2016-03-28", "2016-04-04",
    "2016-04-11", "2016-04-15"
]
```

```python
# 判断读入数据，哪种节约内存，速度快
def get_actions_0():
    action = pd.read_csv(action_1_path)
    return action
def get_actions_1():
    action = pd.read_csv(action_1_path)
    action[['user_id','sku_id','model_id','type','cate','brand']] = action[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')
    return action
def get_actions_2():
    action = pd.read_csv(action_1_path)
    action[['user_id','sku_id','model_id','type','cate','brand']] = action[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')

    return action
def get_actions_3():
    action = pd.read_csv(action_1_path)
    action[['user_id','sku_id','model_id','type','cate','brand']] = action[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')

    return action

def get_actions_10():

    reader = pd.read_csv(action_1_path, iterator=True)
    reader[['user_id','sku_id','model_id','type','cate','brand']] = reader[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(50000)
            chunks.append(chunk)
        except StopIteration:
            loop = False
            print("Iteration is stopped")
    action = pd.concat(chunks, ignore_index=True)

    return action
def get_actions_20():

    reader = pd.read_csv(action_2_path, iterator=True)
    reader[['user_id','sku_id','model_id','type','cate','brand']] = reader[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(50000)
            chunks.append(chunk)
        except StopIteration:
            loop = False
            print("Iteration is stopped")
    action = pd.concat(chunks, ignore_index=True)

    return action
def get_actions_30():

    reader = pd.read_csv(action_3_path, iterator=True)
    reader[['user_id','sku_id','model_id','type','cate','brand']] = reader[['user_id','sku_id','model_id','type','cate','brand']].astype('float32')
    chunks = []
    loop = True
    while loop:
        try:
            chunk = reader.get_chunk(50000)
            chunks.append(chunk)
```

```
        except StopIteration:
            loop = False
            print("Iteration is stopped")
    action = pd.concat(chunks, ignore_index=True)

    return action

# 读取并拼接所有行为记录文件
def get_all_action():
    action_1 = get_actions_1()
    action_2 = get_actions_2()
    action_3 = get_actions_3()
    actions = pd.concat([action_1, action_2, action_3]) # type: pd.DataFrame

    return actions

# 获取某个时间段的行为记录，大于等于起始时间，小于终止时间
def get_actions(start_date, end_date, all_actions):
    """
    :param start_date:
    :param end_date:
    :return: actions: pd.Dataframe
    """
    actions = all_actions[(all_actions.time >= start_date) & (all_actions.time < end_date)].copy()
    return actions
```

4.3.1 用户基本特征

- 下面分解：获取基本的用户特征，基于用户本身属性多为类别特征的特点，对age,sex,usr_lv_cd进行独热编码操作，对于用户注册时间暂时不处理

```
from sklearn import preprocessing

def get_basic_user_feat():
    # 针对年龄的中文字符问题处理，首先是读入的时候编码，填充空值，然后将其数值化，最后独热编码，此外对于sex也进行了数值类型转换
    user = pd.read_csv(user_path, encoding='gbk')

    user.dropna(axis=0, how='any',inplace=True)
    user['sex'] = user['sex'].astype(int)
    user['age'] = user['age'].astype(int)
    le = preprocessing.LabelEncoder()
    age_df = le.fit_transform(user['age'])
#     print list(le.classes_)

    age_df = pd.get_dummies(age_df, prefix='age')
    sex_df = pd.get_dummies(user['sex'], prefix='sex')
    user_lv_df = pd.get_dummies(user['user_lv_cd'], prefix='user_lv_cd')
    user = pd.concat([user['user_id'], age_df, sex_df, user_lv_df], axis=1)
    return user
```

```
user = pd.read_csv(user_path, encoding='gbk')
user.isnull().any() # 判断是否文件中有空值，False代表没有空值，True代表有空值
```

```
user_id        False
age             True
sex             True
user_lv_cd     False
user_reg_tm     True
dtype: bool
```

```
user[user.isnull().values==True] # 检查所有空值的列
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age	sex	user_lv_cd	user_reg_tm
34072	234073	NaN	NaN	1	NaN
34072	234073	NaN	NaN	1	NaN
34072	234073	NaN	NaN	1	NaN
38905	238906	NaN	NaN	1	NaN
38905	238906	NaN	NaN	1	NaN
38905	238906	NaN	NaN	1	NaN
67704	267705	NaN	NaN	1	NaN
67704	267705	NaN	NaN	1	NaN
67704	267705	NaN	NaN	1	NaN

```
user.dropna(axis=0, how='any',inplace=True) # 按行删除
user.isnull().any() # False代表没有缺失值
```

```
user_id       False
age           False
sex           False
user_lv_cd    False
user_reg_tm   False
dtype: bool
```

4.3.2 商品基本特征

- 根据商品文件获取基本的特征，针对属性a1,a2,a3进行独热编码，商品类别和品牌直接作为特征

```
def get_basic_product_feat():
    product = pd.read_csv(product_path)
    attr1_df = pd.get_dummies(product["a1"], prefix="a1")
    attr2_df = pd.get_dummies(product["a2"], prefix="a2")
    attr3_df = pd.get_dummies(product["a3"], prefix="a3")
    product = pd.concat([product[['sku_id', 'cate', 'brand']], attr1_df, attr2_df, attr3_df], axis=1)
    return product
```

4.3.3 评论特征

- 分时间段
- 对评论数进行独热编码

```
def get_comments_product_feat(end_date):
    comments = pd.read_csv(comment_path)
    comment_date_end = end_date
    comment_date_begin = comment_date[0]
    for date in reversed(comment_date):
        if date < comment_date_end:
            comment_date_begin = date
            break
    comments = comments[comments.dt==comment_date_begin]
    df = pd.get_dummies(comments['comment_num'], prefix='comment_num')
    # 为了防止某个时间段不具备评论数为0的情况（测试集出现过这种情况）
    for i in range(0, 5):
        if 'comment_num_' + str(i) not in df.columns:
            df['comment_num_' + str(i)] = 0
    df = df[['comment_num_0', 'comment_num_1', 'comment_num_2', 'comment_num_3', 'comment_num_4']]

    comments = pd.concat([comments, df], axis=1) # type: pd.DataFrame
        #del comments['dt']
        #del comments['comment_num']
    comments = comments[['sku_id', 'has_bad_comment', 'bad_comment_rate','comment_num_0', 'comment_num_1',
                         'comment_num_2', 'comment_num_3', 'comment_num_4']]
    return comments
```

```
train_start_date = '2016-02-01'
train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
train_end_date = train_end_date.strftime('%Y-%m-%d')
day = 3

start_date = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=day)
start_date = start_date.strftime('%Y-%m-%d')
```

```
comments = pd.read_csv(comment_path)
comment_date_end = train_end_date
comment_date_begin = comment_date[0]
for date in reversed(comment_date):
     if date < comment_date_end:
        comment_date_begin = date
        break
comments = comments[comments.dt==comment_date_begin]
df = pd.get_dummies(comments['comment_num'], prefix='comment_num')
for i in range(0, 5):
    if 'comment_num_' + str(i) not in df.columns:
         df['comment_num_' + str(i)] = 0
df = df[['comment_num_0', 'comment_num_1', 'comment_num_2', 'comment_num_3', 'comment_num_4']]

comments = pd.concat([comments, df], axis=1)

comments = comments[['sku_id', 'has_bad_comment', 'bad_comment_rate','comment_num_0', 'comment_num_1',
                        'comment_num_2', 'comment_num_3', 'comment_num_4']]
comments.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sku_id	has_bad_comment	bad_comment_rate	comment_num_0	comment_num_1	comment_num_2	comment_num_3	comment_num_4
0	1000	1	0.0417	0	0	0	1	0
1	10000	0	0.0000	0	0	1	0	0
2	100011	1	0.0376	0	0	0	0	1
3	100018	0	0.0000	0	0	0	1	0
4	100020	0	0.0000	0	0	0	1	0

4.3.4 行为特征

- 分时间段
- 对行为类别进行独热编码
- 分别按照用户-类别行为分组和用户-类别-商品行为分组统计，然后计算
- 用户对同类别下其他商品的行为计数
- 针对用户对同类别下目标商品的行为计数与该时间段的行为均值作差

```
def get_action_feat(start_date, end_date, all_actions, i):
    actions = get_actions(start_date, end_date, all_actions)
    actions = actions[['user_id', 'sku_id', 'cate','type']]
    # 不同时间累积的行为计数（3,5,7,10,15,21,30）
    df = pd.get_dummies(actions['type'], prefix='action_before_%s' %i)
    before_date = 'action_before_%s' %i
    actions = pd.concat([actions, df], axis=1)  # type: pd.DataFrame
    # 分组统计，用户-类别-商品,不同用户对不同类别下商品的行为计数
    actions = actions.groupby(['user_id', 'sku_id','cate'], as_index=False).sum()
    # 分组统计，用户-类别，不同用户对不同商品类别的行为计数
    user_cate = actions.groupby(['user_id','cate'], as_index=False).sum()
    del user_cate['sku_id']
    del user_cate['type']
    actions = pd.merge(actions, user_cate, how='left', on=['user_id','cate'])
    #本类别下其他商品点击量
    # 前述两种分组含有相同名称的不同行为的计数，系统会自动针对名称调整添加后缀,x,y，所以这里作差统计的是同一类别下其他商品的行为计数
    actions[before_date+'_1.0_y'] = actions[before_date+'_1.0_y'] - actions[before_date+'_1.0_x']
    actions[before_date+'_2.0_y'] = actions[before_date+'_2.0_y'] - actions[before_date+'_2.0_x']
```

```
    actions[before_date+'_3.0_y'] = actions[before_date+'_3.0_y'] - actions[before_date+'_3.0_x']
    actions[before_date+'_4.0_y'] = actions[before_date+'_4.0_y'] - actions[before_date+'_4.0_x']
    actions[before_date+'_5.0_y'] = actions[before_date+'_5.0_y'] - actions[before_date+'_5.0_x']
    actions[before_date+'_6.0_y'] = actions[before_date+'_6.0_y'] - actions[before_date+'_6.0_x']
    # 统计用户对不同类别下商品计数与该类别下商品行为计数均值（对时间）的差值
    actions[before_date+'minus_mean_1'] = actions[before_date+'_1.0_x'] - (actions[before_date+'_1.0_x']/i)
    actions[before_date+'minus_mean_2'] = actions[before_date+'_2.0_x'] - (actions[before_date+'_2.0_x']/i)
    actions[before_date+'minus_mean_3'] = actions[before_date+'_3.0_x'] - (actions[before_date+'_3.0_x']/i)
    actions[before_date+'minus_mean_4'] = actions[before_date+'_4.0_x'] - (actions[before_date+'_4.0_x']/i)
    actions[before_date+'minus_mean_5'] = actions[before_date+'_5.0_x'] - (actions[before_date+'_5.0_x']/i)
    actions[before_date+'minus_mean_6'] = actions[before_date+'_6.0_x'] - (actions[before_date+'_6.0_x']/i)
    del actions['type']
    # 保留cate特征
#     del actions['cate']

    return actions
```

```
all_actions = get_all_action()

actions = get_actions(start_date, train_end_date, all_actions)
actions = actions[['user_id', 'sku_id', 'cate','type']]
    # 不同时间累积的行为计数（3,5,7,10,15,21,30）
df = pd.get_dummies(actions['type'], prefix='action_before_%s' %3)
before_date = 'action_before_%s' %3
actions = pd.concat([actions, df], axis=1)  # type: pd.DataFrame
    # 分组统计，用户-类别-商品,不同用户对不同类别下商品的行为计数
actions = actions.groupby(['user_id', 'sku_id','cate'], as_index=False).sum()
actions.head(20)
# 在3天之内，行为/type为2，action_before_3_1.0
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	cate	type	action_before_3_1.0	action_before_3_2.0	action_before_3_3.0	action_before_3_4.0	action_before_3_5.0	act
0	200002.0	7199.0	4.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0
1	200002.0	24369.0	7.0	66.0	12.0	0.0	0.0	0.0	0.0	9.0
2	200002.0	28973.0	4.0	120.0	12.0	0.0	0.0	0.0	0.0	18.
3	200002.0	73364.0	4.0	72.0	18.0	0.0	0.0	0.0	0.0	9.0
4	200002.0	75588.0	5.0	60.0	6.0	0.0	0.0	0.0	0.0	9.0
5	200002.0	78335.0	8.0	72.0	0.0	0.0	0.0	0.0	0.0	12.
6	200002.0	88764.0	5.0	60.0	6.0	0.0	0.0	0.0	0.0	9.0
7	200002.0	93295.0	8.0	156.0	12.0	0.0	0.0	0.0	0.0	24.
8	200002.0	118303.0	4.0	114.0	6.0	0.0	0.0	0.0	0.0	18.
9	200002.0	149851.0	4.0	96.0	6.0	0.0	0.0	0.0	0.0	15.
10	200003.0	1203.0	4.0	60.0	6.0	0.0	0.0	0.0	0.0	9.0
11	200003.0	3067.0	8.0	84.0	12.0	0.0	0.0	0.0	0.0	12.
12	200003.0	4919.0	8.0	78.0	6.0	0.0	0.0	0.0	0.0	12.
13	200003.0	24371.0	8.0	78.0	6.0	0.0	0.0	0.0	0.0	12.
14	200003.0	39425.0	8.0	78.0	6.0	0.0	0.0	0.0	0.0	12.
15	200003.0	117882.0	4.0	60.0	6.0	0.0	0.0	0.0	0.0	9.0
16	200003.0	131300.0	8.0	60.0	6.0	0.0	0.0	0.0	0.0	9.0
17	200003.0	135272.0	4.0	120.0	12.0	0.0	0.0	0.0	0.0	18.
18	200008.0	88312.0	7.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0
19	200008.0	118238.0	7.0	372.0	12.0	0.0	0.0	0.0	0.0	60.

```
# 某一个客户对第4大类别3天内做了为"1"的行为的次数为48
user_cate = actions.groupby(['user_id','cate'], as_index=False).sum()
del user_cate['sku_id']
del user_cate['type']
user_cate.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	cate	action_before_3_1.0	action_before_3_2.0	action_before_3_3.0	action_before_3_4.0	action_before_3_5.0	action_before_3_6.0
0	200002.0	4.0	48.0	0.0	0.0	0.0	0.0	60.0
1	200002.0	5.0	12.0	0.0	0.0	0.0	0.0	18.0
2	200002.0	7.0	12.0	0.0	0.0	0.0	0.0	9.0
3	200002.0	8.0	12.0	0.0	0.0	0.0	0.0	36.0
4	200003.0	4.0	24.0	0.0	0.0	0.0	0.0	36.0

```
actions = pd.merge(actions, user_cate, how='left', on=['user_id','cate'])
actions.head()
# x指的是特定的商品, "sku_id"; y指的是品类"cate"
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	cate	type	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x
0	200002.0	7199.0	4.0	6.0	6.0	0.0	0.0	0.0	0.0
1	200002.0	24369.0	7.0	66.0	12.0	0.0	0.0	0.0	0.0
2	200002.0	28973.0	4.0	120.0	12.0	0.0	0.0	0.0	0.0
3	200002.0	73364.0	4.0	72.0	18.0	0.0	0.0	0.0	0.0
4	200002.0	75588.0	5.0	60.0	6.0	0.0	0.0	0.0	0.0

```
# 差异/区别/比重, 在某个品类里面, 除了某个特定的商品, 其他的商品是什么情况
actions[before_date+'_1_y'] = actions[before_date+'_1.0_y'] - actions[before_date+'_1.0_x']
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	cate	type	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x
0	200002.0	7199.0	4.0	6.0	6.0	0.0	0.0	0.0	0.0
1	200002.0	24369.0	7.0	66.0	12.0	0.0	0.0	0.0	0.0
2	200002.0	28973.0	4.0	120.0	12.0	0.0	0.0	0.0	0.0
3	200002.0	73364.0	4.0	72.0	18.0	0.0	0.0	0.0	0.0
4	200002.0	75588.0	5.0	60.0	6.0	0.0	0.0	0.0	0.0

4.3.5 累积用户特征

- 分时间段
- 用户不同行为的
- 购买转化率
- 均值

```
all_actions
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time	model_id	type	cate	brand
0	266079.0	138778.0	2016-01-31 23:59:02	NaN	1.0	8.0	403.0
1	266079.0	138778.0	2016-01-31 23:59:03	0.0	6.0	8.0	403.0
2	200719.0	61226.0	2016-01-31 23:59:07	NaN	1.0	8.0	30.0
3	200719.0	61226.0	2016-01-31 23:59:08	0.0	6.0	8.0	30.0
4	263587.0	72348.0	2016-01-31 23:59:08	NaN	1.0	5.0	159.0
...	...	...	...	...	...	...	...
11485419	213906.0	103132.0	2016-02-29 23:59:59	216.0	6.0	8.0	545.0
11485420	301058.0	20869.0	2016-02-29 23:59:59	NaN	2.0	9.0	630.0
11485421	213906.0	103132.0	2016-02-29 23:59:59	0.0	6.0	8.0	545.0
11485422	213906.0	103132.0	2016-02-29 23:59:59	0.0	6.0	8.0	545.0
11485423	213906.0	103132.0	2016-02-29 23:59:59	217.0	6.0	8.0	545.0

34456272 rows × 7 columns

```
def get_accumulate_user_feat(end_date, all_actions, day):
    start_date = datetime.strptime(end_date, '%Y-%m-%d') - timedelta(days=day)
    start_date = start_date.strftime('%Y-%m-%d')
    before_date = 'user_action_%s' % day

    feature = [
        'user_id', before_date + '_1', before_date + '_2', before_date + '_3',
        before_date + '_4', before_date + '_5', before_date + '_6',
        before_date + '_1_ratio', before_date + '_2_ratio',
        before_date + '_3_ratio', before_date + '_5_ratio',
        before_date + '_6_ratio', before_date + '_1_mean',
        before_date + '_2_mean', before_date + '_3_mean',
        before_date + '_4_mean', before_date + '_5_mean',
        before_date + '_6_mean', before_date + '_1_std',
        before_date + '_2_std', before_date + '_3_std', before_date + '_4_std',
        before_date + '_5_std', before_date + '_6_std'
    ]

    actions = get_actions(start_date, end_date, all_actions)
    df = pd.get_dummies(actions['type'], prefix=before_date)
```

```
    actions['date'] = pd.to_datetime(actions['time']).apply(lambda x: x.date())

    actions = pd.concat([actions[['user_id', 'date']], df], axis=1)

    actions[before_date + '_1_ratio'] =  np.log(1 + actions[before_date + '_4.0']) - np.log(1 + actions[before_date +'_1.0'])
    actions[before_date + '_2_ratio'] =  np.log(1 + actions[before_date + '_4.0']) - np.log(1 + actions[before_date +'_2.0'])
    actions[before_date + '_3_ratio'] =  np.log(1 + actions[before_date + '_4.0']) - np.log(1 + actions[before_date +'_3.0'])
    actions[before_date + '_5_ratio'] =  np.log(1 + actions[before_date + '_4.0']) - np.log(1 + actions[before_date +'_5.0'])
    actions[before_date + '_6_ratio'] =  np.log(1 + actions[before_date + '_4.0']) - np.log(1 + actions[before_date +'_6.0'])
    # 均值
    actions[before_date + '_1_mean'] = actions[before_date + '_1.0'] / day
    actions[before_date + '_2_mean'] = actions[before_date + '_2.0'] / day
    actions[before_date + '_3_mean'] = actions[before_date + '_3.0'] / day
    actions[before_date + '_4_mean'] = actions[before_date + '_4.0'] / day
    actions[before_date + '_5_mean'] = actions[before_date + '_5.0'] / day
    actions[before_date + '_6_mean'] = actions[before_date + '_6.0'] / day
    #actions = pd.merge(actions, actions_date, how='left', on='user_id')
    #actions = actions[feature]
    return actions
```

```
train_start_date = '2016-02-01'
train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
train_end_date = train_end_date.strftime('%Y-%m-%d')
day = 3

start_date = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=day)
start_date = start_date.strftime('%Y-%m-%d')
before_date = 'user_action_%s' % day
```

```
before_date
```

```
'user_action_3'
```

```
# 取3天的时间判断
print (start_date)
print (train_end_date)
```

```
2016-02-01
2016-02-04
```

```
all_actions.shape
```

```
(34456272, 7)
```

```
actions = get_actions(start_date, train_end_date, all_actions)
actions.shape
```

```
(3015330, 7)
```

```
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time	model_id	type	cate	brand
29	272629.0	107774.0	2016-02-01 00:00:00	NaN	1.0	10.0	36.0
30	272629.0	107774.0	2016-02-01 00:00:00	NaN	1.0	10.0	36.0
31	272629.0	107774.0	2016-02-01 00:00:00	0.0	6.0	10.0	36.0
32	272629.0	107774.0	2016-02-01 00:00:00	NaN	1.0	10.0	36.0
33	272629.0	107774.0	2016-02-01 00:00:00	216.0	6.0	10.0	36.0

```
df = pd.get_dummies(actions['type'], prefix=before_date)
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_action_3_1.0	user_action_3_2.0	user_action_3_3.0	user_action_3_4.0	user_action_3_5.0	user_action_3_6.0
29	1	0	0	0	0	0
30	1	0	0	0	0	0
31	0	0	0	0	0	1
32	1	0	0	0	0	0
33	0	0	0	0	0	1

```
actions['date'] = pd.to_datetime(actions['time']).apply(lambda x: x.date())
actions = pd.concat([actions[['user_id', 'date']], df], axis=1)
actions_date = actions.groupby(['user_id', 'date']).sum()
actions_date.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

		user_action_3_1.0	user_action_3_2.0	user_action_3_3.0	user_action_3_4.0	user_action_3_5.0	user_action_3_6.0
user_id	**date**						
200002.0	**2016-02-01**	12.0	0.0	0.0	0.0	0.0	36.0
	2016-02-02	12.0	0.0	0.0	0.0	0.0	9.0
	2016-02-03	60.0	0.0	0.0	0.0	0.0	78.0
200003.0	**2016-02-02**	36.0	0.0	0.0	0.0	0.0	57.0
	2016-02-03	24.0	0.0	0.0	0.0	0.0	36.0

```
actions_date = actions_date.unstack()
actions_date.fillna(0, inplace=True)
actions_date.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead tr th {
    text-align: left;
}

.dataframe thead tr:last-of-type th {
    text-align: right;
}
```

	user_action_3_1.0			user_action_3_2.0			user_action_3_3.0			user_action_3_4.0			user_action_3_5.0			user_action_3_6.	
date	**2016-02-01**	**2016-02-02**	**2016-02-03**	**2016-02-01**	**2016-02-02**	**2016-02-03**	**2016-02-01**	**2016-02-02**	**2016-02-03**	**2016-02-01**	**2016-02-02**	**2016-02-03**	**2016-02-01**	**2016-02-02**	**2016-02-03**	**2016-02-01**	**2016-02-02**
user_id																	
200002.0	12.0	12.0	60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	36.0	9.0
200003.0	0.0	36.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	57.0
200008.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0	0.0

```
actions = actions.groupby(['user_id'], as_index=False).sum()
actions.head()
# 对于用户来说，什么样的行为才能购买
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	user_action_3_1.0	user_action_3_2.0	user_action_3_3.0	user_action_3_4.0	user_action_3_5.0	user_action_3_6.0
0	200002.0	84.0	0.0	0.0	0.0	0.0	123.0
1	200003.0	60.0	0.0	0.0	0.0	0.0	93.0
2	200008.0	24.0	0.0	0.0	0.0	0.0	60.0
3	200023.0	3.0	0.0	0.0	0.0	0.0	0.0
4	200030.0	24.0	0.0	0.0	0.0	0.0	51.0

```
# _4.0：代表的是购买，转化率：logA-logB = logA/B ， user_action_3_1_ratio
actions[before_date + '_1_ratio'] =  np.log(1 + actions[before_date + '_4.0']) - np.log(1 + actions[before_date +'_1.0'])
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	user_action_3_1.0	user_action_3_2.0	user_action_3_3.0	user_action_3_4.0	user_action_3_5.0	user_action_3_6.0	user_action_3_1_ratio
0	200002.0	84.0	0.0	0.0	0.0	0.0	123.0	-4.442651
1	200003.0	60.0	0.0	0.0	0.0	0.0	93.0	-4.110874
2	200008.0	24.0	0.0	0.0	0.0	0.0	60.0	-3.218876
3	200023.0	3.0	0.0	0.0	0.0	0.0	0.0	-1.386294
4	200030.0	24.0	0.0	0.0	0.0	0.0	51.0	-3.218876

```
# 均值
actions[before_date + '_1_mean'] = actions[before_date + '_1.0'] / day
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	user_action_3_1.0	user_action_3_2.0	user_action_3_3.0	user_action_3_4.0	user_action_3_5.0	user_action_3_6.0	user_action_3_1_ratio
0	200002.0	84.0	0.0	0.0	0.0	0.0	123.0	-4.442651
1	200003.0	60.0	0.0	0.0	0.0	0.0	93.0	-4.110874
2	200008.0	24.0	0.0	0.0	0.0	0.0	60.0	-3.218876
3	200023.0	3.0	0.0	0.0	0.0	0.0	0.0	-1.386294
4	200030.0	24.0	0.0	0.0	0.0	0.0	51.0	-3.218876

4.3.6 用户近期行为特征

- 在上面针对用户进行累积特征提取的基础上，分别提取用户近一个月、近三天的特征，然后提取一个月内用户除去最近三天的行为占据一个月的行为的比重

```
def get_recent_user_feat(end_date, all_actions):

    actions_3 = get_accumulate_user_feat(end_date, all_actions, 3)# 通过终止时间往前推3天
    actions_30 = get_accumulate_user_feat(end_date, all_actions, 30)# 通过终止时间往前推30天
    actions = pd.merge(actions_3, actions_30, how ='left', on='user_id')
    del actions_3
    del actions_30
    # 一个月内用户除去最近三天的行为占据一个月的行为的比重
    actions['recent_action1'] =  np.log(1 + actions['user_action_30_1.0']-actions['user_action_3_1.0']) - np.log(1 + actions['user_action_30_1.0'])
    actions['recent_action2'] =  np.log(1 + actions['user_action_30_2.0']-actions['user_action_3_2.0']) - np.log(1 + actions['user_action_30_2.0'])
    actions['recent_action3'] =  np.log(1 + actions['user_action_30_3.0']-actions['user_action_3_3.0']) - np.log(1 + actions['user_action_30_3.0'])
    actions['recent_action4'] =  np.log(1 + actions['user_action_30_4.0']-actions['user_action_3_4.0']) - np.log(1 + actions['user_action_30_4.0'])
    actions['recent_action5'] =  np.log(1 + actions['user_action_30_5.0']-actions['user_action_3_5.0']) - np.log(1 + actions['user_action_30_5.0'])
    actions['recent_action6'] =  np.log(1 + actions['user_action_30_6.0']-actions['user_action_3_6.0']) - np.log(1 + actions['user_action_30_6.0'])



    return actions
```

4.3.7 用户对同类别下各种商品的行为

- 用户对各个类别的各项行为操作统计
- 用户对各个类别操作行为统计占对所有类别操作行为统计的比重

```
#增加了用户对不同类别的交互特征
def get_user_cate_feature(start_date, end_date, all_actions):
    actions = get_actions(start_date, end_date, all_actions)
    actions = actions[['user_id', 'cate', 'type']]
    df = pd.get_dummies(actions['type'], prefix='type')
    actions = pd.concat([actions[['user_id', 'cate']], df], axis=1)
    actions = actions.groupby(['user_id', 'cate']).sum()
    actions = actions.unstack()
    actions.columns = actions.columns.swaplevel(0, 1)
```

```
actions.columns = actions.columns.droplevel()
actions.columns = [
    'cate_4_type1', 'cate_5_type1', 'cate_6_type1', 'cate_7_type1',
    'cate_8_type1', 'cate_9_type1', 'cate_10_type1', 'cate_11_type1',
    'cate_4_type2', 'cate_5_type2', 'cate_6_type2', 'cate_7_type2',
    'cate_8_type2', 'cate_9_type2', 'cate_10_type2', 'cate_11_type2',
    'cate_4_type3', 'cate_5_type3', 'cate_6_type3', 'cate_7_type3',
    'cate_8_type3', 'cate_9_type3', 'cate_10_type3', 'cate_11_type3',
    'cate_4_type4', 'cate_5_type4', 'cate_6_type4', 'cate_7_type4',
    'cate_8_type4', 'cate_9_type4', 'cate_10_type4', 'cate_11_type4',
    'cate_4_type5', 'cate_5_type5', 'cate_6_type5', 'cate_7_type5',
    'cate_8_type5', 'cate_9_type5', 'cate_10_type5', 'cate_11_type5',
    'cate_4_type6', 'cate_5_type6', 'cate_6_type6', 'cate_7_type6',
    'cate_8_type6', 'cate_9_type6', 'cate_10_type6', 'cate_11_type6'
]
actions = actions.fillna(0)
actions['cate_action_sum'] = actions.sum(axis=1)
actions['cate8_percentage'] = (
    actions['cate_8_type1'] + actions['cate_8_type2'] +
    actions['cate_8_type3'] + actions['cate_8_type4'] +
    actions['cate_8_type5'] + actions['cate_8_type6']
) / actions['cate_action_sum']
actions['cate4_percentage'] = (
    actions['cate_4_type1'] + actions['cate_4_type2'] +
    actions['cate_4_type3'] + actions['cate_4_type4'] +
    actions['cate_4_type5'] + actions['cate_4_type6']
) / actions['cate_action_sum']
actions['cate5_percentage'] = (
    actions['cate_5_type1'] + actions['cate_5_type2'] +
    actions['cate_5_type3'] + actions['cate_5_type4'] +
    actions['cate_5_type5'] + actions['cate_5_type6']
) / actions['cate_action_sum']
actions['cate6_percentage'] = (
    actions['cate_6_type1'] + actions['cate_6_type2'] +
    actions['cate_6_type3'] + actions['cate_6_type4'] +
    actions['cate_6_type5'] + actions['cate_6_type6']
) / actions['cate_action_sum']
actions['cate7_percentage'] = (
    actions['cate_7_type1'] + actions['cate_7_type2'] +
    actions['cate_7_type3'] + actions['cate_7_type4'] +
    actions['cate_7_type5'] + actions['cate_7_type6']
) / actions['cate_action_sum']
actions['cate9_percentage'] = (
    actions['cate_9_type1'] + actions['cate_9_type2'] +
    actions['cate_9_type3'] + actions['cate_9_type4'] +
    actions['cate_9_type5'] + actions['cate_9_type6']
) / actions['cate_action_sum']
actions['cate10_percentage'] = (
    actions['cate_10_type1'] + actions['cate_10_type2'] +
    actions['cate_10_type3'] + actions['cate_10_type4'] +
    actions['cate_10_type5'] + actions['cate_10_type6']
) / actions['cate_action_sum']
actions['cate11_percentage'] = (
    actions['cate_11_type1'] + actions['cate_11_type2'] +
    actions['cate_11_type3'] + actions['cate_11_type4'] +
    actions['cate_11_type5'] + actions['cate_11_type6']
) / actions['cate_action_sum']

actions['cate8_type1_percentage'] = np.log(
    1 + actions['cate_8_type1']) - np.log(
        1 + actions['cate_8_type1'] + actions['cate_4_type1'] +
        actions['cate_5_type1'] + actions['cate_6_type1'] +
        actions['cate_7_type1'] + actions['cate_9_type1'] +
        actions['cate_10_type1'] + actions['cate_11_type1'])

actions['cate8_type2_percentage'] = np.log(
    1 + actions['cate_8_type2']) - np.log(
        1 + actions['cate_8_type2'] + actions['cate_4_type2'] +
        actions['cate_5_type2'] + actions['cate_6_type2'] +
        actions['cate_7_type2'] + actions['cate_9_type2'] +
        actions['cate_10_type2'] + actions['cate_11_type2'])
actions['cate8_type3_percentage'] = np.log(
    1 + actions['cate_8_type3']) - np.log(
        1 + actions['cate_8_type3'] + actions['cate_4_type3'] +
        actions['cate_5_type3'] + actions['cate_6_type3'] +
        actions['cate_7_type3'] + actions['cate_9_type3'] +
        actions['cate_10_type3'] + actions['cate_11_type3'])
actions['cate8_type4_percentage'] = np.log(
    1 + actions['cate_8_type4']) - np.log(
```

```
            1 + actions['cate_8_type4'] + actions['cate_4_type4'] +
            actions['cate_5_type4'] + actions['cate_6_type4'] +
            actions['cate_7_type4'] + actions['cate_9_type4'] +
            actions['cate_10_type4'] + actions['cate_11_type4'])
    actions['cate8_type5_percentage'] = np.log(
        1 + actions['cate_8_type5']) - np.log(
            1 + actions['cate_8_type5'] + actions['cate_4_type5'] +
            actions['cate_5_type5'] + actions['cate_6_type5'] +
            actions['cate_7_type5'] + actions['cate_9_type5'] +
            actions['cate_10_type5'] + actions['cate_11_type5'])
    actions['cate8_type6_percentage'] = np.log(
        1 + actions['cate_8_type6']) - np.log(
            1 + actions['cate_8_type6'] + actions['cate_4_type6'] +
            actions['cate_5_type6'] + actions['cate_6_type6'] +
            actions['cate_7_type6'] + actions['cate_9_type6'] +
            actions['cate_10_type6'] + actions['cate_11_type6'])
    actions['user_id'] = actions.index
    actions = actions[[
        'user_id', 'cate8_percentage', 'cate4_percentage', 'cate5_percentage',
        'cate6_percentage', 'cate7_percentage', 'cate9_percentage',
        'cate10_percentage', 'cate11_percentage', 'cate8_type1_percentage',
        'cate8_type2_percentage', 'cate8_type3_percentage',
        'cate8_type4_percentage', 'cate8_type5_percentage',
        'cate8_type6_percentage'
    ]]
    return actions
```

```
train_start_date = '2016-02-01'
train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
train_end_date = train_end_date.strftime('%Y-%m-%d')
day = 3

start_date = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=day)
start_date = start_date.strftime('%Y-%m-%d')

print (start_date)
print (train_end_date)
```

```
2016-02-01
2016-02-04
```

```
actions = get_actions(start_date, train_end_date, all_actions)
actions = actions[['user_id', 'cate', 'type']]
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	cate	type
29	272629.0	10.0	1.0
30	272629.0	10.0	1.0
31	272629.0	10.0	6.0
32	272629.0	10.0	1.0
33	272629.0	10.0	6.0

```
df = pd.get_dummies(actions['type'], prefix='type')
actions = pd.concat([actions[['user_id', 'cate']], df], axis=1)
actions = actions.groupby(['user_id', 'cate']).sum()
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

		type_1.0	type_2.0	type_3.0	type_4.0	type_5.0	type_6.0
user_id	**cate**						
200002.0	**4.0**	48.0	0.0	0.0	0.0	0.0	60.0
	5.0	12.0	0.0	0.0	0.0	0.0	18.0
	7.0	12.0	0.0	0.0	0.0	0.0	9.0
	8.0	12.0	0.0	0.0	0.0	0.0	36.0
200003.0	**4.0**	24.0	0.0	0.0	0.0	0.0	36.0


```
actions = actions.unstack() # 花括号结构变成表结构
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead tr th {
    text-align: left;
}

.dataframe thead tr:last-of-type th {
    text-align: right;
}
```

	type_1.0								type_2.0		...	type_5.0		type_6.0						
cate	**4.0**	**5.0**	**6.0**	**7.0**	**8.0**	**9.0**	**10.0**	**11.0**	**4.0**	**5.0**	**...**	**10.0**	**11.0**	**4.0**	**5.0**	**6.0**	**7.0**	**8.0**	**9.0**	**10**
user_id																				
200002.0	48.0	12.0	NaN	12.0	12.0	NaN	NaN	NaN	0.0	0.0	...	NaN	NaN	60.0	18.0	NaN	9.0	36.0	NaN	Na
200003.0	24.0	NaN	NaN	NaN	36.0	NaN	NaN	NaN	0.0	NaN	...	NaN	NaN	36.0	NaN	NaN	NaN	57.0	NaN	Na
200008.0	NaN	NaN	NaN	24.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	60.0	NaN	NaN	Na
200023.0	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	Na
200030.0	24.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	...	NaN	NaN	51.0	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 48 columns

```
actions.columns
```

```
MultiIndex([('type_1.0',  4.0),
            ('type_1.0',  5.0),
            ('type_1.0',  6.0),
            ('type_1.0',  7.0),
            ('type_1.0',  8.0),
            ('type_1.0',  9.0),
            ('type_1.0', 10.0),
            ('type_1.0', 11.0),
            ('type_2.0',  4.0),
            ('type_2.0',  5.0),
            ('type_2.0',  6.0),
            ('type_2.0',  7.0),
            ('type_2.0',  8.0),
            ('type_2.0',  9.0),
            ('type_2.0', 10.0),
```

```
            ('type_2.0', 11.0),
            ('type_3.0',  4.0),
            ('type_3.0',  5.0),
            ('type_3.0',  6.0),
            ('type_3.0',  7.0),
            ('type_3.0',  8.0),
            ('type_3.0',  9.0),
            ('type_3.0', 10.0),
            ('type_3.0', 11.0),
            ('type_4.0',  4.0),
            ('type_4.0',  5.0),
            ('type_4.0',  6.0),
            ('type_4.0',  7.0),
            ('type_4.0',  8.0),
            ('type_4.0',  9.0),
            ('type_4.0', 10.0),
            ('type_4.0', 11.0),
            ('type_5.0',  4.0),
            ('type_5.0',  5.0),
            ('type_5.0',  6.0),
            ('type_5.0',  7.0),
            ('type_5.0',  8.0),
            ('type_5.0',  9.0),
            ('type_5.0', 10.0),
            ('type_5.0', 11.0),
            ('type_6.0',  4.0),
            ('type_6.0',  5.0),
            ('type_6.0',  6.0),
            ('type_6.0',  7.0),
            ('type_6.0',  8.0),
            ('type_6.0',  9.0),
            ('type_6.0', 10.0),
            ('type_6.0', 11.0)],
           names=[None, 'cate'])
```

```
actions.columns = actions.columns.swaplevel(0, 1)#接受两个级别编号或名称,并返回一个互换了级别的新对象(但数据不会发生变化)
actions.columns
```

```
MultiIndex([( 4.0, 'type_1.0'),
            ( 5.0, 'type_1.0'),
            ( 6.0, 'type_1.0'),
            ( 7.0, 'type_1.0'),
            ( 8.0, 'type_1.0'),
            ( 9.0, 'type_1.0'),
            (10.0, 'type_1.0'),
            (11.0, 'type_1.0'),
            ( 4.0, 'type_2.0'),
            ( 5.0, 'type_2.0'),
            ( 6.0, 'type_2.0'),
            ( 7.0, 'type_2.0'),
            ( 8.0, 'type_2.0'),
            ( 9.0, 'type_2.0'),
            (10.0, 'type_2.0'),
            (11.0, 'type_2.0'),
            ( 4.0, 'type_3.0'),
            ( 5.0, 'type_3.0'),
            ( 6.0, 'type_3.0'),
            ( 7.0, 'type_3.0'),
            ( 8.0, 'type_3.0'),
            ( 9.0, 'type_3.0'),
            (10.0, 'type_3.0'),
            (11.0, 'type_3.0'),
            ( 4.0, 'type_4.0'),
            ( 5.0, 'type_4.0'),
            ( 6.0, 'type_4.0'),
            ( 7.0, 'type_4.0'),
            ( 8.0, 'type_4.0'),
            ( 9.0, 'type_4.0'),
            (10.0, 'type_4.0'),
            (11.0, 'type_4.0'),
            ( 4.0, 'type_5.0'),
            ( 5.0, 'type_5.0'),
            ( 6.0, 'type_5.0'),
            ( 7.0, 'type_5.0'),
```

```
           ( 8.0, 'type_5.0'),
           ( 9.0, 'type_5.0'),
           (10.0, 'type_5.0'),
           (11.0, 'type_5.0'),
           ( 4.0, 'type_6.0'),
           ( 5.0, 'type_6.0'),
           ( 6.0, 'type_6.0'),
           ( 7.0, 'type_6.0'),
           ( 8.0, 'type_6.0'),
           ( 9.0, 'type_6.0'),
           (10.0, 'type_6.0'),
           (11.0, 'type_6.0')],
          names=['cate', None])
```

```
actions.columns = actions.columns.droplevel()
actions.columns
```

```
Index(['type_1.0', 'type_1.0', 'type_1.0', 'type_1.0', 'type_1.0', 'type_1.0',
       'type_1.0', 'type_1.0', 'type_2.0', 'type_2.0', 'type_2.0', 'type_2.0',
       'type_2.0', 'type_2.0', 'type_2.0', 'type_2.0', 'type_3.0', 'type_3.0',
       'type_3.0', 'type_3.0', 'type_3.0', 'type_3.0', 'type_3.0', 'type_3.0',
       'type_4.0', 'type_4.0', 'type_4.0', 'type_4.0', 'type_4.0', 'type_4.0',
       'type_4.0', 'type_4.0', 'type_5.0', 'type_5.0', 'type_5.0', 'type_5.0',
       'type_5.0', 'type_5.0', 'type_5.0', 'type_5.0', 'type_6.0', 'type_6.0',
       'type_6.0', 'type_6.0', 'type_6.0', 'type_6.0', 'type_6.0', 'type_6.0'],
      dtype='object')
```

```
actions.columns = [
        'cate_4_type1', 'cate_5_type1', 'cate_6_type1', 'cate_7_type1',
        'cate_8_type1', 'cate_9_type1', 'cate_10_type1', 'cate_11_type1',
        'cate_4_type2', 'cate_5_type2', 'cate_6_type2', 'cate_7_type2',
        'cate_8_type2', 'cate_9_type2', 'cate_10_type2', 'cate_11_type2',
        'cate_4_type3', 'cate_5_type3', 'cate_6_type3', 'cate_7_type3',
        'cate_8_type3', 'cate_9_type3', 'cate_10_type3', 'cate_11_type3',
        'cate_4_type4', 'cate_5_type4', 'cate_6_type4', 'cate_7_type4',
        'cate_8_type4', 'cate_9_type4', 'cate_10_type4', 'cate_11_type4',
        'cate_4_type5', 'cate_5_type5', 'cate_6_type5', 'cate_7_type5',
        'cate_8_type5', 'cate_9_type5', 'cate_10_type5', 'cate_11_type5',
        'cate_4_type6', 'cate_5_type6', 'cate_6_type6', 'cate_7_type6',
        'cate_8_type6', 'cate_9_type6', 'cate_10_type6', 'cate_11_type6'
    ]
actions.columns
```

```
Index(['cate_4_type1', 'cate_5_type1', 'cate_6_type1', 'cate_7_type1',
       'cate_8_type1', 'cate_9_type1', 'cate_10_type1', 'cate_11_type1',
       'cate_4_type2', 'cate_5_type2', 'cate_6_type2', 'cate_7_type2',
       'cate_8_type2', 'cate_9_type2', 'cate_10_type2', 'cate_11_type2',
       'cate_4_type3', 'cate_5_type3', 'cate_6_type3', 'cate_7_type3',
       'cate_8_type3', 'cate_9_type3', 'cate_10_type3', 'cate_11_type3',
       'cate_4_type4', 'cate_5_type4', 'cate_6_type4', 'cate_7_type4',
       'cate_8_type4', 'cate_9_type4', 'cate_10_type4', 'cate_11_type4',
       'cate_4_type5', 'cate_5_type5', 'cate_6_type5', 'cate_7_type5',
       'cate_8_type5', 'cate_9_type5', 'cate_10_type5', 'cate_11_type5',
       'cate_4_type6', 'cate_5_type6', 'cate_6_type6', 'cate_7_type6',
       'cate_8_type6', 'cate_9_type6', 'cate_10_type6', 'cate_11_type6'],
      dtype='object')
```

```
actions = actions.fillna(0) # 拿0填充，因为用户没有行为
actions['cate_action_sum'] = actions.sum(axis=1)
actions.head()
# 一个用户对第4大类，执行了行为1的动作
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate_4_type1	cate_5_type1	cate_6_type1	cate_7_type1	cate_8_type1	cate_9_type1	cate_10_type1	cate_11_type1	cate_4_type2	cat
user_id										
200002.0	48.0	12.0	0.0	12.0	12.0	0.0	0.0	0.0	0.0	0.0
200003.0	24.0	0.0	0.0	0.0	36.0	0.0	0.0	0.0	0.0	0.0
200008.0	0.0	0.0	0.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0
200023.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
200030.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 49 columns

```
# 第第八大类的6种行为，各自的占比
actions['cate8_percentage'] = (
        actions['cate_8_type1'] + actions['cate_8_type2'] +
        actions['cate_8_type3'] + actions['cate_8_type4'] +
        actions['cate_8_type5'] + actions['cate_8_type6']
    ) / actions['cate_action_sum']
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate_4_type1	cate_5_type1	cate_6_type1	cate_7_type1	cate_8_type1	cate_9_type1	cate_10_type1	cate_11_type1	cate_4_type2	cat
user_id										
200002.0	48.0	12.0	0.0	12.0	12.0	0.0	0.0	0.0	0.0	0.0
200003.0	24.0	0.0	0.0	0.0	36.0	0.0	0.0	0.0	0.0	0.0
200008.0	0.0	0.0	0.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0
200023.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
200030.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 50 columns

```
actions['cate8_type1_percentage'] = np.log(
        1 + actions['cate_8_type1']) - np.log(
            1 + actions['cate_8_type1'] + actions['cate_4_type1'] +
            actions['cate_5_type1'] + actions['cate_6_type1'] +
            actions['cate_7_type1'] + actions['cate_9_type1'] +
            actions['cate_10_type1'] + actions['cate_11_type1'])
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate_4_type1	cate_5_type1	cate_6_type1	cate_7_type1	cate_8_type1	cate_9_type1	cate_10_type1	cate_11_type1	cate_4_type2	cat
user_id										
200002.0	48.0	12.0	0.0	12.0	12.0	0.0	0.0	0.0	0.0	0.0
200003.0	24.0	0.0	0.0	0.0	36.0	0.0	0.0	0.0	0.0	0.0
200008.0	0.0	0.0	0.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0
200023.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
200030.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 51 columns

4.3.8 累积商品特征

- 分时间段
- 针对商品的不同行为的
- 购买转化率
- 均值
- 标准差

```python
def get_accumulate_product_feat(start_date, end_date, all_actions):
    feature = [
        'sku_id', 'product_action_1', 'product_action_2',
        'product_action_3', 'product_action_4',
        'product_action_5', 'product_action_6',
        'product_action_1_ratio', 'product_action_2_ratio',
        'product_action_3_ratio', 'product_action_5_ratio',
        'product_action_6_ratio', 'product_action_1_mean',
        'product_action_2_mean', 'product_action_3_mean',
        'product_action_4_mean', 'product_action_5_mean',
        'product_action_6_mean', 'product_action_1_std',
        'product_action_2_std', 'product_action_3_std', 'product_action_4_std',
        'product_action_5_std', 'product_action_6_std'
    ]

    actions = get_actions(start_date, end_date, all_actions)
    df = pd.get_dummies(actions['type'], prefix='product_action')
    # 按照商品-日期分组，计算某个时间段该商品的各项行为的标准差
    actions['date'] = pd.to_datetime(actions['time']).apply(lambda x: x.date())
    actions = pd.concat([actions[['sku_id', 'date']], df], axis=1)

    actions = actions.groupby(['sku_id'], as_index=False).sum()
    days_interal = (datetime.strptime(end_date, '%Y-%m-%d') - datetime.strptime(start_date, '%Y-%m-%d')).days
    actions['product_action_1_ratio'] =  np.log(1 + actions['product_action_4.0']) - np.log(1 + actions['product_action_1.0'])
    actions['product_action_2_ratio'] =  np.log(1 + actions['product_action_4.0']) - np.log(1 + actions['product_action_2.0'])
    actions['product_action_3_ratio'] =  np.log(1 + actions['product_action_4.0']) - np.log(1 + actions['product_action_3.0'])
    actions['product_action_5_ratio'] =  np.log(1 + actions['product_action_4.0']) - np.log(1 + actions['product_action_5.0'])
    actions['product_action_6_ratio'] =  np.log(1 + actions['product_action_4.0']) - np.log(1 + actions['product_action_6.0'])
    # 计算各种行为的均值
    actions['product_action_1_mean'] = actions[
        'product_action_1.0'] / days_interal
    actions['product_action_2_mean'] = actions[
        'product_action_2.0'] / days_interal
    actions['product_action_3_mean'] = actions[
        'product_action_3.0'] / days_interal
    actions['product_action_4_mean'] = actions[
        'product_action_4.0'] / days_interal
    actions['product_action_5_mean'] = actions[
        'product_action_5.0'] / days_interal
    actions['product_action_6_mean'] = actions[
        'product_action_6.0'] / days_interal
    #actions = pd.merge(actions, actions_date, how='left', on='sku_id')
    #actions = actions[feature]
    return actions
```

```
train_start_date = '2016-02-01'
train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
train_end_date = train_end_date.strftime('%Y-%m-%d')
day = 3

start_date = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=day)
start_date = start_date.strftime('%Y-%m-%d')

print (start_date)
print (train_end_date)
```

```
2016-02-01
2016-02-04
```

```
actions = get_actions(start_date, train_end_date, all_actions)
df = pd.get_dummies(actions['type'], prefix='product_action')

actions['date'] = pd.to_datetime(actions['time']).apply(lambda x: x.date())
actions = pd.concat([actions[['sku_id', 'date']], df], axis=1)
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sku_id	date	product_action_1.0	product_action_2.0	product_action_3.0	product_action_4.0	product_action_5.0	product_action_6.0
29	107774.0	2016-02-01	1	0	0	0	0	0
30	107774.0	2016-02-01	1	0	0	0	0	0
31	107774.0	2016-02-01	0	0	0	0	0	1
32	107774.0	2016-02-01	1	0	0	0	0	0
33	107774.0	2016-02-01	0	0	0	0	0	1

```
actions = actions.groupby(['sku_id'], as_index=False).sum()
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sku_id	product_action_1.0	product_action_2.0	product_action_3.0	product_action_4.0	product_action_5.0	product_action_6.0
0	2.0	6.0	0.0	0.0	0.0	0.0	9.0
1	37.0	6.0	0.0	0.0	0.0	0.0	9.0
2	40.0	12.0	0.0	0.0	0.0	0.0	27.0
3	50.0	24.0	0.0	6.0	0.0	0.0	42.0
4	52.0	261.0	0.0	3.0	0.0	0.0	336.0

```
days_interal = (datetime.strptime(train_end_date, '%Y-%m-%d') - datetime.strptime(start_date, '%Y-%m-%d')).days
days_interal
```

```
3
```

```
actions['product_action_1_ratio'] =  np.log(1 + actions['product_action_4.0']) - np.log(1 + actions['product_action_1.0'])
actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sku_id	product_action_1.0	product_action_2.0	product_action_3.0	product_action_4.0	product_action_5.0	product_action_6.0	product_action_1_
0	2.0	6.0	0.0	0.0	0.0	0.0	9.0	-1.945910
1	37.0	6.0	0.0	0.0	0.0	0.0	9.0	-1.945910
2	40.0	12.0	0.0	0.0	0.0	0.0	27.0	-2.564949
3	50.0	24.0	0.0	6.0	0.0	0.0	42.0	-3.218876
4	52.0	261.0	0.0	3.0	0.0	0.0	336.0	-5.568345

4.3.9 类别特征

分时间段下各个商品类别的

- 购买转化率
- 标准差
- 均值

```
def get_accumulate_cate_feat(start_date, end_date, all_actions):
    feature = ['cate','cate_action_1', 'cate_action_2', 'cate_action_3', 'cate_action_4', 'cate_action_5',
               'cate_action_6', 'cate_action_1_ratio', 'cate_action_2_ratio',
               'cate_action_3_ratio', 'cate_action_5_ratio', 'cate_action_6_ratio', 'cate_action_1_mean',
               'cate_action_2_mean', 'cate_action_3_mean', 'cate_action_4_mean', 'cate_action_5_mean',
               'cate_action_6_mean', 'cate_action_1_std', 'cate_action_2_std', 'cate_action_3_std',
               'cate_action_4_std', 'cate_action_5_std', 'cate_action_6_std']
    actions = get_actions(start_date, end_date, all_actions)
    actions['date'] = pd.to_datetime(actions['time']).apply(lambda x: x.date())
    df = pd.get_dummies(actions['type'], prefix='cate_action')
    actions = pd.concat([actions[['cate','date']], df], axis=1)
    # 按照类别分组，统计各个商品类别下行为的转化率
    actions = actions.groupby(['cate'], as_index=False).sum()
    days_interal = (datetime.strptime(end_date, '%Y-%m-%d')-datetime.strptime(start_date, '%Y-%m-%d')).days

    actions['cate_action_1_ratio'] =(np.log(1 + actions['cate_action_4.0']) - np.log(1 + actions['cate_action_1.0']))
    actions['cate_action_2_ratio'] =(np.log(1 + actions['cate_action_4.0']) - np.log(1 + actions['cate_action_2.0']))
    actions['cate_action_3_ratio'] =(np.log(1 + actions['cate_action_4.0']) - np.log(1 + actions['cate_action_3.0']))
    actions['cate_action_5_ratio'] =(np.log(1 + actions['cate_action_4.0']) - np.log(1 + actions['cate_action_5.0']))
    actions['cate_action_6_ratio'] =(np.log(1 + actions['cate_action_4.0']) - np.log(1 + actions['cate_action_6.0']))
    # 按照类别分组，统计各个商品类别下行为在一段时间的均值
    actions['cate_action_1_mean'] = actions['cate_action_1.0'] /  days_interal
    actions['cate_action_2_mean'] = actions['cate_action_2.0'] /  days_interal
    actions['cate_action_3_mean'] = actions['cate_action_3.0'] /  days_interal
    actions['cate_action_4_mean'] = actions['cate_action_4.0'] /  days_interal
    actions['cate_action_5_mean'] = actions['cate_action_5.0'] /  days_interal
    actions['cate_action_6_mean'] = actions['cate_action_6.0'] /  days_interal
    #actions = pd.merge(actions, actions_date, how ='left',on='cate')
    #actions = actions[feature]
    return actions
```

4.4 构造训练集/测试集

- 标签,采用滑动窗口的方式，构造训练集的时候针对产生购买的行为标记为1
- 整合特征

```
def get_labels(start_date, end_date, all_actions):
    actions = get_actions(start_date, end_date, all_actions)

    # 修改为预测购买了商品8的用户预测
    actions = actions[(actions['type'] == 4) & (actions['cate']==8)]

    actions = actions.groupby(['user_id', 'sku_id'], as_index=False).sum()
    actions['label'] = 1
    actions = actions[['user_id', 'sku_id', 'label']]
    return actions
```

```
train_start_date = '2016-03-01'
train_actions = None
all_actions = get_all_action()
print ("get all actions!")
```

```
get all actions!
```

```
all_actions.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	time	model_id	type	cate	brand
0	266079.0	138778.0	2016-01-31 23:59:02	NaN	1.0	8.0	403.0
1	266079.0	138778.0	2016-01-31 23:59:03	0.0	6.0	8.0	403.0
2	200719.0	61226.0	2016-01-31 23:59:07	NaN	1.0	8.0	30.0
3	200719.0	61226.0	2016-01-31 23:59:08	0.0	6.0	8.0	30.0
4	263587.0	72348.0	2016-01-31 23:59:08	NaN	1.0	5.0	159.0

```
all_actions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 34456272 entries, 0 to 11485423
Data columns (total 7 columns):
user_id      float32
sku_id       float32
time         object
model_id     float32
type         float32
cate         float32
brand        float32
dtypes: float32(6), object(1)
memory usage: 1.3+ GB
```

```
all_actions.shape
```

```
(34456272, 7)
```

```python
user = get_basic_user_feat()
print ('get_basic_user_feat finsihed')
```

```
get_basic_user_feat finsihed
```

```python
user.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	age_0	age_1	age_2	age_3	age_4	age_5	age_6	sex_0	sex_1	sex_2	user_lv_cd_1	user_lv_cd_2	user_lv_cd_3	user_lv_
0	200001.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	200002.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
2	200003.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	200004.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
4	200005.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0

```python
product = get_basic_product_feat()
print ('get_basic_product_feat finsihed')
```

```
get_basic_product_feat finsihed
```

```python
product.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sku_id	cate	brand	a1_-1	a1_1	a1_2	a1_3	a2_-1	a2_1	a2_2	a3_-1	a3_1	a3_2
0	10	8	489	0	0	0	1	0	1	0	0	1	0
1	100002	8	489	0	0	0	1	0	0	1	0	0	1
2	100003	8	30	0	1	0	0	1	0	0	1	0	0
3	100006	8	545	0	1	0	0	0	0	1	0	1	0
4	10001	8	244	1	0	0	0	0	1	0	0	0	1

```python
train_start_date = '2016-03-01'
train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
train_end_date
```

```
datetime.datetime(2016, 3, 4, 0, 0)
```

```
train_end_date = train_end_date.strftime('%Y-%m-%d')
# 修正prod_acc,cate_acc的时间跨度
start_days = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=30)
start_days = start_days.strftime('%Y-%m-%d')
print (train_end_date)
```

```
2016-03-04
```

```
start_days
```

```
'2016-02-03'
```

4.5.1 构造训练集

```
def make_actions(user, product, all_actions, train_start_date):
    train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
    train_end_date = train_end_date.strftime('%Y-%m-%d')
    # 修正prod_acc,cate_acc的时间跨度
    start_days = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=30)
    start_days = start_days.strftime('%Y-%m-%d')
    print (train_end_date)
    user_acc = get_recent_user_feat(train_end_date, all_actions)
    print ('get_recent_user_feat finsihed')

    user_cate = get_user_cate_feature(train_start_date, train_end_date, all_actions)
    print ('get_user_cate_feature finished')

    product_acc = get_accumulate_product_feat(start_days, train_end_date, all_actions)
    print ('get_accumulate_product_feat finsihed')
    cate_acc = get_accumulate_cate_feat(start_days, train_end_date, all_actions)
    print ('get_accumulate_cate_feat finsihed')
    comment_acc = get_comments_product_feat(train_end_date)
    print ('get_comments_product_feat finished')
    # 标记
    test_start_date = train_end_date
    test_end_date = datetime.strptime(test_start_date, '%Y-%m-%d') + timedelta(days=5)
    test_end_date = test_end_date.strftime('%Y-%m-%d')
    labels = get_labels(test_start_date, test_end_date, all_actions)
    print ("get labels")

    actions = None
    for i in (3, 5, 7, 10, 15, 21, 30):
        start_days = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=i)
        start_days = start_days.strftime('%Y-%m-%d')
        if actions is None:
            actions = get_action_feat(start_days, train_end_date, all_actions, i)
        else:
            # 注意这里的拼接key
            actions = pd.merge(actions, get_action_feat(start_days, train_end_date, all_actions, i), how='left',
                               on=['user_id', 'sku_id', 'cate'])

    actions = pd.merge(actions, user, how='left', on='user_id')
    actions = pd.merge(actions, user_acc, how='left', on='user_id')
    user_cate.index.name = ""
    actions = pd.merge(actions, user_cate, how='left', on='user_id')
    # 注意这里的拼接key
    actions = pd.merge(actions, product, how='left', on=['sku_id', 'cate'])
    actions = pd.merge(actions, product_acc, how='left', on='sku_id')
    actions = pd.merge(actions, cate_acc, how='left', on='cate')
    actions = pd.merge(actions, comment_acc, how='left', on='sku_id')
    actions = pd.merge(actions, labels, how='left', on=['user_id', 'sku_id'])
    # 主要是填充拼接商品基本特征、评论特征、标签之后的空值
    actions = actions.fillna(0)
#     return actions
    # 采样
    action_postive = actions[actions['label'] == 1]
    action_negative = actions[actions['label'] == 0]
    del actions
```

```
    neg_len = len(action_postive) * 10
    action_negative = action_negative.sample(n=neg_len)
    action_sample = pd.concat([action_postive, action_negative], ignore_index=True)

    return action_sample
```

```
def make_train_set(train_start_date, setNums ,f_path, all_actions):
    train_actions = None
    user = get_basic_user_feat()
    print ('get_basic_user_feat finsihed')
    product = get_basic_product_feat()
    print ('get_basic_product_feat finsihed')
    # 滑窗,构造多组训练集/验证集
    for i in range(setNums):
        print (train_start_date)
        if train_actions is None:
            train_actions = make_actions(user, product, all_actions, train_start_date)
        else:
            train_actions = pd.concat([train_actions, make_actions(user, product, all_actions, train_start_date)],
                                          ignore_index=True)
        # 接下来每次移动一天
        train_start_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=1)
        train_start_date = train_start_date.strftime('%Y-%m-%d')
        print ("round {0}/{1} over!".format(i+1, setNums))

    train_actions.to_csv(f_path, index=False)
```

```
train_start_date = '2016-02-01'
train_end_date = datetime.strptime(train_start_date, '%Y-%m-%d') + timedelta(days=3)
train_end_date

train_end_date = train_end_date.strftime('%Y-%m-%d')
# 修正prod_acc,cate_acc的时间跨度
start_days = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=30)
start_days = start_days.strftime('%Y-%m-%d')
print (train_end_date)
```

```
2016-02-04
```

```
user_cate = get_user_cate_feature(train_start_date, train_end_date, all_actions)
print ('get_user_cate_feature finished')
```

```
get_user_cate_feature finished
```

```
product_acc = get_accumulate_product_feat(start_days, train_end_date, all_actions)
print ('get_accumulate_product_feat finsihed')
```

```
get_accumulate_product_feat finsihed
```

```
cate_acc = get_accumulate_cate_feat(start_days, train_end_date, all_actions)
print ('get_accumulate_cate_feat finsihed')
```

```
get_accumulate_cate_feat finsihed
```

```
# 训练集
train_start_date = '2016-02-01'
make_train_set(train_start_date, 20, 'train_set.csv',all_actions)
```

```
get_basic_user_feat finsihed
get_basic_product_feat finsihed
2016-02-01
2016-02-04
```

4.5.2 构造验证集(线下测试集)

```
def make_test_set(train_start_date, train_end_date):
    start_days = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=30)
    start_days = start_days.strftime('%Y-%m-%d')
    all_actions = get_all_action()
    print ("get all actions!")
    user = get_basic_user_feat()
    print ('get_basic_user_feat finsihed')
    product = get_basic_product_feat()
    print ('get_basic_product_feat finsihed')

    user_acc = get_recent_user_feat(train_end_date, all_actions)
    print ('get_accumulate_user_feat finsihed')

    user_cate = get_user_cate_feature(train_start_date, train_end_date, all_actions)
    print ('get_user_cate_feature finished')

    product_acc = get_accumulate_product_feat(start_days, train_end_date, all_actions)
    print ('get_accumulate_product_feat finsihed')
    cate_acc = get_accumulate_cate_feat(start_days, train_end_date, all_actions)
    print ('get_accumulate_cate_feat finsihed')
    comment_acc = get_comments_product_feat(train_end_date)

    actions = None
    for i in (3, 5, 7, 10, 15, 21, 30):
        start_days = datetime.strptime(train_end_date, '%Y-%m-%d') - timedelta(days=i)
        start_days = start_days.strftime('%Y-%m-%d')
        if actions is None:
            actions = get_action_feat(start_days, train_end_date, all_actions,i)
        else:
            actions = pd.merge(actions, get_action_feat(start_days, train_end_date,all_actions,i), how='left',
                               on=['user_id', 'sku_id', 'cate'])

    actions = pd.merge(actions, user, how='left', on='user_id')
    actions = pd.merge(actions, user_acc, how='left', on='user_id')
    user_cate.index.name = ""
    actions = pd.merge(actions, user_cate, how='left', on='user_id')
    # 注意这里的拼接key
    actions = pd.merge(actions, product, how='left', on=['sku_id', 'cate'])
    actions = pd.merge(actions, product_acc, how='left', on='sku_id')
    actions = pd.merge(actions, cate_acc, how='left', on='cate')
    actions = pd.merge(actions, comment_acc, how='left', on='sku_id')

    actions = actions.fillna(0)


    actions.to_csv("test_set.csv", index=False)
```

```
make_val_set('2016-02-23', '2016-02-26', 'val_3.csv')
```

4.5 模型设计

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import sys
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
import operator
from matplotlib import pylab as plt
from datetime import datetime
import time
from sklearn.model_selection import GridSearchCV
```

```
data = pd.read_csv('train_set.csv')
data.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	acti
0	202633.0	12564.0	8.0	3.0	0.0	0.0	0.0	0.0	6.0
1	218498.0	149854.0	8.0	12.0	0.0	0.0	0.0	0.0	12.0
2	221842.0	75877.0	8.0	9.0	0.0	0.0	0.0	0.0	15.0
3	222886.0	154636.0	8.0	60.0	3.0	0.0	0.0	0.0	78.0
4	235240.0	38222.0	8.0	90.0	3.0	0.0	0.0	0.0	84.0

5 rows × 251 columns

```
data.columns
```

```
Index(['user_id', 'sku_id', 'cate', 'action_before_3_1.0_x',
       'action_before_3_2.0_x', 'action_before_3_3.0_x',
       'action_before_3_4.0_x', 'action_before_3_5.0_x',
       'action_before_3_6.0_x', 'action_before_3_1.0_y',
       ...
       'cate_action_5_mean', 'cate_action_6_mean', 'has_bad_comment',
       'bad_comment_rate', 'comment_num_0', 'comment_num_1', 'comment_num_2',
       'comment_num_3', 'comment_num_4', 'label'],
      dtype='object', length=251)
```

```
data_x = data.loc[:,data.columns != 'label']
data_y = data.loc[:,data.columns == 'label']
data_y
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	label
0	1.0
1	1.0
2	1.0
3	1.0
4	1.0
...	...
14614	0.0
14615	0.0
14616	0.0
14617	0.0
14618	0.0

14619 rows × 1 columns

```
data_x.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	acti
0	202633.0	12564.0	8.0	3.0	0.0	0.0	0.0	0.0	6.0
1	218498.0	149854.0	8.0	12.0	0.0	0.0	0.0	0.0	12.0
2	221842.0	75877.0	8.0	9.0	0.0	0.0	0.0	0.0	15.0
3	222886.0	154636.0	8.0	60.0	3.0	0.0	0.0	0.0	78.0
4	235240.0	38222.0	8.0	90.0	3.0	0.0	0.0	0.0	84.0

5 rows × 250 columns

```
x_train, x_test, y_train, y_test = train_test_split(data_x,data_y,test_size = 0.2, random_state = 0)
```

```
x_test.shape
```

```
(2924, 250)
```

```
x_val = x_test.iloc[:1500,:]
y_val = y_test.iloc[:1500,:]

x_test = x_test.iloc[1500:,:]
y_test = y_test.iloc[1500:,:]
```

```
print (x_val.shape)
print (x_test.shape)
```

```
(1500, 250)
(1424, 250)
```

```
# 删掉user_id和sku_id两列
del x_train['user_id']
del x_train['sku_id']

del x_val['user_id']
del x_val['sku_id']

x_train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	action_before_3_6.0_x
2157	4.0	12.0	0.0	0.0	0.0	0.0	12.0
2464	8.0	36.0	3.0	0.0	0.0	0.0	42.0
10326	5.0	6.0	0.0	3.0	0.0	0.0	0.0
7025	8.0	24.0	0.0	0.0	0.0	3.0	27.0
6625	4.0	6.0	0.0	0.0	0.0	0.0	9.0

5 rows × 248 columns

```
dtrain = xgb.DMatrix(x_train, label=y_train)
dvalid = xgb.DMatrix(x_val, label=y_val)
```

```
param = {'n_estimators': 1000, 'max_depth': 3, 'min_child_weight': 5, 'gamma': 0, 'subsample': 1.0,
             'colsample_bytree': 0.8, 'scale_pos_weight':10, 'eta': 0.1,  'objective': 'binary:logistic',
             'eval_metric':'auc'}
```

```
num_round = param['n_estimators']

plst = param.items()
evallist = [(dtrain, 'train'), (dvalid, 'eval')]
bst = xgb.train(plst, dtrain, num_round, evallist, early_stopping_rounds=10)
bst.save_model('bst.model')
```

```
[0] train-auc:0.936457  eval-auc:0.931955
Multiple eval metrics have been passed: 'eval-auc' will be used for early stopping.

Will train until eval-auc hasn't improved in 10 rounds.
[1] train-auc:0.948588  eval-auc:0.946668
[2] train-auc:0.950761  eval-auc:0.949238
[3] train-auc:0.952042  eval-auc:0.950185
[4] train-auc:0.954199  eval-auc:0.952755
[5] train-auc:0.955445  eval-auc:0.954173
[6] train-auc:0.955791  eval-auc:0.954146
[7] train-auc:0.955894  eval-auc:0.953923
[8] train-auc:0.957458  eval-auc:0.953992
[9] train-auc:0.9572  eval-auc:0.953444
[10]  train-auc:0.958348  eval-auc:0.954144
[11]  train-auc:0.958507  eval-auc:0.954101
[12]  train-auc:0.959617  eval-auc:0.955641
[13]  train-auc:0.960018  eval-auc:0.955285
[14]  train-auc:0.960389  eval-auc:0.956115
[15]  train-auc:0.961506  eval-auc:0.956203
```

```
[16]	train-auc:0.961488	eval-auc:0.956195
[17]	train-auc:0.96192	eval-auc:0.955742
[18]	train-auc:0.962073	eval-auc:0.95603
[19]	train-auc:0.963318	eval-auc:0.956583
[20]	train-auc:0.963766	eval-auc:0.956977
[21]	train-auc:0.96397	eval-auc:0.957096
[22]	train-auc:0.964204	eval-auc:0.956926
[23]	train-auc:0.964346	eval-auc:0.9572
[24]	train-auc:0.964611	eval-auc:0.957432
[25]	train-auc:0.964829	eval-auc:0.957761
[26]	train-auc:0.964932	eval-auc:0.957737
[27]	train-auc:0.965726	eval-auc:0.957591
[28]	train-auc:0.96593	eval-auc:0.957855
[29]	train-auc:0.966388	eval-auc:0.958107
[30]	train-auc:0.9665	eval-auc:0.957841
[31]	train-auc:0.967164	eval-auc:0.958541
[32]	train-auc:0.967379	eval-auc:0.958669
[33]	train-auc:0.967877	eval-auc:0.958341
[34]	train-auc:0.968663	eval-auc:0.958671
[35]	train-auc:0.968954	eval-auc:0.958208
[36]	train-auc:0.969542	eval-auc:0.958895
[37]	train-auc:0.969942	eval-auc:0.95948
[38]	train-auc:0.970083	eval-auc:0.959629
[39]	train-auc:0.970408	eval-auc:0.959235
[40]	train-auc:0.970618	eval-auc:0.959459
[41]	train-auc:0.970968	eval-auc:0.959855
[42]	train-auc:0.971313	eval-auc:0.960424
[43]	train-auc:0.97172	eval-auc:0.96036
[44]	train-auc:0.971962	eval-auc:0.960845
[45]	train-auc:0.972123	eval-auc:0.961095
[46]	train-auc:0.972502	eval-auc:0.960728
[47]	train-auc:0.972696	eval-auc:0.96119
[48]	train-auc:0.972847	eval-auc:0.961265
[49]	train-auc:0.973202	eval-auc:0.961563
[50]	train-auc:0.973328	eval-auc:0.96177
[51]	train-auc:0.973535	eval-auc:0.961967
[52]	train-auc:0.973956	eval-auc:0.962337
[53]	train-auc:0.974147	eval-auc:0.962656
[54]	train-auc:0.974356	eval-auc:0.96297
[55]	train-auc:0.974679	eval-auc:0.963289
[56]	train-auc:0.974803	eval-auc:0.963353
[57]	train-auc:0.974974	eval-auc:0.96364
[58]	train-auc:0.975141	eval-auc:0.963922
[59]	train-auc:0.975188	eval-auc:0.963944
[60]	train-auc:0.975476	eval-auc:0.964103
[61]	train-auc:0.975777	eval-auc:0.964066
[62]	train-auc:0.97595	eval-auc:0.964492
[63]	train-auc:0.976106	eval-auc:0.96455
[64]	train-auc:0.976349	eval-auc:0.964465
[65]	train-auc:0.976684	eval-auc:0.964736
[66]	train-auc:0.97698	eval-auc:0.964928
[67]	train-auc:0.977133	eval-auc:0.96505
[68]	train-auc:0.977175	eval-auc:0.965205
[69]	train-auc:0.977276	eval-auc:0.965284
[70]	train-auc:0.977463	eval-auc:0.965268
[71]	train-auc:0.977618	eval-auc:0.965279
[72]	train-auc:0.977692	eval-auc:0.965332
[73]	train-auc:0.977772	eval-auc:0.965428
[74]	train-auc:0.978007	eval-auc:0.965449
[75]	train-auc:0.978245	eval-auc:0.96554
[76]	train-auc:0.97836	eval-auc:0.965561
[77]	train-auc:0.978434	eval-auc:0.965678
[78]	train-auc:0.978483	eval-auc:0.965699
[79]	train-auc:0.978644	eval-auc:0.965582
[80]	train-auc:0.97886	eval-auc:0.966024
[81]	train-auc:0.978958	eval-auc:0.966152
[82]	train-auc:0.979094	eval-auc:0.966301
[83]	train-auc:0.979288	eval-auc:0.96654
[84]	train-auc:0.979479	eval-auc:0.966716
[85]	train-auc:0.97956	eval-auc:0.966865
[86]	train-auc:0.979744	eval-auc:0.966918
[87]	train-auc:0.979815	eval-auc:0.966976
[88]	train-auc:0.98	eval-auc:0.967333
[89]	train-auc:0.98007	eval-auc:0.967434
[90]	train-auc:0.980193	eval-auc:0.967365
[91]	train-auc:0.980312	eval-auc:0.96753
[92]	train-auc:0.980349	eval-auc:0.967535
[93]	train-auc:0.980404	eval-auc:0.967562
[94]	train-auc:0.980447	eval-auc:0.96754
```

```
[95]	train-auc:0.980648	eval-auc:0.967508
[96]	train-auc:0.980707	eval-auc:0.967583
[97]	train-auc:0.980766	eval-auc:0.967673
[98]	train-auc:0.980972	eval-auc:0.967796
[99]	train-auc:0.98098	eval-auc:0.967764
[100]	train-auc:0.981032	eval-auc:0.967774
[101]	train-auc:0.981096	eval-auc:0.967849
[102]	train-auc:0.981131	eval-auc:0.967945
[103]	train-auc:0.981216	eval-auc:0.967998
[104]	train-auc:0.981255	eval-auc:0.968035
[105]	train-auc:0.981469	eval-auc:0.967987
[106]	train-auc:0.981694	eval-auc:0.968205
[107]	train-auc:0.981877	eval-auc:0.968397
[108]	train-auc:0.981971	eval-auc:0.968248
[109]	train-auc:0.982032	eval-auc:0.968333
[110]	train-auc:0.98206	eval-auc:0.968471
[111]	train-auc:0.982229	eval-auc:0.968423
[112]	train-auc:0.98227	eval-auc:0.968434
[113]	train-auc:0.982336	eval-auc:0.968509
[114]	train-auc:0.982393	eval-auc:0.968572
[115]	train-auc:0.982447	eval-auc:0.968578
[116]	train-auc:0.982462	eval-auc:0.968567
[117]	train-auc:0.982499	eval-auc:0.968631
[118]	train-auc:0.982694	eval-auc:0.968604
[119]	train-auc:0.982832	eval-auc:0.968705
[120]	train-auc:0.982997	eval-auc:0.968801
[121]	train-auc:0.983053	eval-auc:0.968812
[122]	train-auc:0.983111	eval-auc:0.968791
[123]	train-auc:0.983257	eval-auc:0.968908
[124]	train-auc:0.983367	eval-auc:0.968982
[125]	train-auc:0.983468	eval-auc:0.968993
[126]	train-auc:0.983596	eval-auc:0.968993
[127]	train-auc:0.983783	eval-auc:0.969078
[128]	train-auc:0.983955	eval-auc:0.969062
[129]	train-auc:0.984115	eval-auc:0.969136
[130]	train-auc:0.984134	eval-auc:0.96912
[131]	train-auc:0.984193	eval-auc:0.969067
[132]	train-auc:0.984197	eval-auc:0.969131
[133]	train-auc:0.984225	eval-auc:0.969168
[134]	train-auc:0.984383	eval-auc:0.969296
[135]	train-auc:0.984427	eval-auc:0.969275
[136]	train-auc:0.9846	eval-auc:0.969413
[137]	train-auc:0.984594	eval-auc:0.96953
[138]	train-auc:0.984606	eval-auc:0.969551
[139]	train-auc:0.984707	eval-auc:0.969679
[140]	train-auc:0.984833	eval-auc:0.969615
[141]	train-auc:0.984999	eval-auc:0.969599
[142]	train-auc:0.985064	eval-auc:0.969748
[143]	train-auc:0.98517	eval-auc:0.9697
[144]	train-auc:0.985254	eval-auc:0.969786
[145]	train-auc:0.985374	eval-auc:0.96978
[146]	train-auc:0.985395	eval-auc:0.969796
[147]	train-auc:0.985448	eval-auc:0.969791
[148]	train-auc:0.985556	eval-auc:0.969727
[149]	train-auc:0.98564	eval-auc:0.969764
[150]	train-auc:0.985761	eval-auc:0.969801
[151]	train-auc:0.985786	eval-auc:0.969892
[152]	train-auc:0.985891	eval-auc:0.96994
[153]	train-auc:0.986012	eval-auc:0.970052
[154]	train-auc:0.986149	eval-auc:0.970094
[155]	train-auc:0.986164	eval-auc:0.970067
[156]	train-auc:0.986255	eval-auc:0.970105
[157]	train-auc:0.986264	eval-auc:0.970179
[158]	train-auc:0.986318	eval-auc:0.970248
[159]	train-auc:0.986336	eval-auc:0.970333
[160]	train-auc:0.986354	eval-auc:0.970349
[161]	train-auc:0.986417	eval-auc:0.970525
[162]	train-auc:0.986543	eval-auc:0.970514
[163]	train-auc:0.986645	eval-auc:0.970594
[164]	train-auc:0.986686	eval-auc:0.970669
[165]	train-auc:0.986787	eval-auc:0.970679
[166]	train-auc:0.986878	eval-auc:0.970615
[167]	train-auc:0.986987	eval-auc:0.970796
[168]	train-auc:0.986969	eval-auc:0.970866
[169]	train-auc:0.987041	eval-auc:0.97085
[170]	train-auc:0.987135	eval-auc:0.970961
[171]	train-auc:0.987291	eval-auc:0.970812
[172]	train-auc:0.987358	eval-auc:0.970711
[173]	train-auc:0.987456	eval-auc:0.970812
```

```
[174] train-auc:0.987483  eval-auc:0.970764
[175] train-auc:0.987488  eval-auc:0.970791
[176] train-auc:0.987544  eval-auc:0.970812
[177] train-auc:0.987628  eval-auc:0.970748
[178] train-auc:0.987671  eval-auc:0.97077
[179] train-auc:0.98783 eval-auc:0.971041
[180] train-auc:0.987995  eval-auc:0.9711
[181] train-auc:0.988107  eval-auc:0.971078
[182] train-auc:0.988118  eval-auc:0.971089
[183] train-auc:0.988185  eval-auc:0.971089
[184] train-auc:0.98822 eval-auc:0.971126
[185] train-auc:0.988258  eval-auc:0.971206
[186] train-auc:0.988304  eval-auc:0.971174
[187] train-auc:0.988319  eval-auc:0.971233
[188] train-auc:0.988422  eval-auc:0.971302
[189] train-auc:0.988526  eval-auc:0.971323
[190] train-auc:0.988629  eval-auc:0.971413
[191] train-auc:0.988647  eval-auc:0.971355
[192] train-auc:0.988719  eval-auc:0.971509
[193] train-auc:0.988828  eval-auc:0.971563
[194] train-auc:0.988875  eval-auc:0.971658
[195] train-auc:0.988889  eval-auc:0.971727
[196] train-auc:0.988925  eval-auc:0.97168
[197] train-auc:0.98904 eval-auc:0.971695
[198] train-auc:0.989114  eval-auc:0.97177
[199] train-auc:0.989165  eval-auc:0.97168
[200] train-auc:0.989256  eval-auc:0.971547
[201] train-auc:0.98933 eval-auc:0.971685
[202] train-auc:0.989404  eval-auc:0.971653
[203] train-auc:0.989417  eval-auc:0.971653
[204] train-auc:0.989456  eval-auc:0.971669
[205] train-auc:0.98949 eval-auc:0.971695
[206] train-auc:0.989517  eval-auc:0.971674
[207] train-auc:0.989535  eval-auc:0.971616
[208] train-auc:0.989592  eval-auc:0.971536
Stopping. Best iteration:
[198] train-auc:0.989114  eval-auc:0.97177
```

```
print (bst.attributes())
```

```
{'best_iteration': '198', 'best_msg': '[198]\ttrain-auc:0.989114\teval-auc:0.97177', 'best_score': '0.97177'}
```

```
def create_feature_map(features):
    outfile = open(r'xgb.fmap', 'w')
    i = 0
    for feat in features:
        outfile.write('{0}\t{1}\tq\n'.format(i, feat))
        i = i + 1
    outfile.close()


features = list(x_train.columns[:])
create_feature_map(features)
```

```
def feature_importance(bst_xgb):
    importance = bst_xgb.get_fscore(fmap=r'xgb.fmap')
    importance = sorted(importance.items(), key=operator.itemgetter(1), reverse=True)

    df = pd.DataFrame(importance, columns=['feature', 'fscore'])
    df['fscore'] = df['fscore'] / df['fscore'].sum()
    file_name = 'feature_importance_' + str(datetime.now().date())[5:] + '.csv'
    df.to_csv(file_name)

feature_importance(bst)
```

```
# 特征重要度
fi = pd.read_csv('feature_importance_10-24.csv')
fi.sort_values("fscore", inplace=True, ascending=False)
fi.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Unnamed: 0	feature	fscore
0	0	brand	0.039548
1	1	bad_comment_rate	0.032486
2	2	user_action_30_2_ratio	0.027542
3	3	action_before_7_5.0_x	0.026130
4	4	product_action_2_ratio	0.022599

```
x_test.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	sku_id	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x
6765	243630.0	63557.0	9.0	3.0	0.0	0.0	0.0	0.0
13767	241426.0	161251.0	9.0	3.0	0.0	0.0	0.0	0.0
9672	270410.0	30383.0	5.0	30.0	0.0	0.0	0.0	0.0
9116	268523.0	49103.0	8.0	54.0	6.0	0.0	0.0	0.0
10055	261660.0	55190.0	4.0	6.0	0.0	0.0	0.0	0.0

5 rows × 250 columns

```
users = x_test[['user_id', 'sku_id', 'cate']].copy()
del x_test['user_id']
del x_test['sku_id']
x_test_DMatrix = xgb.DMatrix(x_test)
y_pred = bst.predict(x_test_DMatrix, ntree_limit=bst.best_ntree_limit)
```

```
x_test['pred_label'] = y_pred
x_test.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	action_before_3_6.0_x
6765	9.0	3.0	0.0	0.0	0.0	0.0	36.0
13767	9.0	3.0	0.0	0.0	0.0	0.0	3.0
9672	5.0	30.0	0.0	0.0	0.0	0.0	27.0
9116	8.0	54.0	6.0	0.0	0.0	0.0	114.0
10055	4.0	6.0	0.0	0.0	0.0	0.0	6.0

5 rows × 249 columns

```
def label(column):
    if column['pred_label'] > 0.5:
        #rint ('yes')
        column['pred_label'] = 1
    else:
        column['pred_label'] = 0
    return column
x_test = x_test.apply(label,axis = 1)
x_test.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	action_before_3_6.0_x
6765	9.0	3.0	0.0	0.0	0.0	0.0	36.0
13767	9.0	3.0	0.0	0.0	0.0	0.0	3.0
9672	5.0	30.0	0.0	0.0	0.0	0.0	27.0
9116	8.0	54.0	6.0	0.0	0.0	0.0	114.0
10055	4.0	6.0	0.0	0.0	0.0	0.0	6.0

5 rows × 249 columns

```
x_test['true_label'] = y_test
x_test.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	action_before_3_6.0_x
6765	9.0	3.0	0.0	0.0	0.0	0.0	36.0
13767	9.0	3.0	0.0	0.0	0.0	0.0	3.0
9672	5.0	30.0	0.0	0.0	0.0	0.0	27.0
9116	8.0	54.0	6.0	0.0	0.0	0.0	114.0
10055	4.0	6.0	0.0	0.0	0.0	0.0	6.0

5 rows × 250 columns

```
x_test['user_id'] = users['user_id']
x_test['sku_id'] = users['sku_id']
x_test.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cate	action_before_3_1.0_x	action_before_3_2.0_x	action_before_3_3.0_x	action_before_3_4.0_x	action_before_3_5.0_x	action_before_3_6.0_x
6765	9.0	3.0	0.0	0.0	0.0	0.0	36.0
13767	9.0	3.0	0.0	0.0	0.0	0.0	3.0
9672	5.0	30.0	0.0	0.0	0.0	0.0	27.0
9116	8.0	54.0	6.0	0.0	0.0	0.0	114.0
10055	4.0	6.0	0.0	0.0	0.0	0.0	6.0

5 rows × 252 columns

```
# 所有购买用户
all_user_set = x_test[x_test['true_label']==1]['user_id'].unique()
print (len(all_user_set))
# 所有预测购买的用户
all_user_test_set = x_test[x_test['pred_label'] == 1]['user_id'].unique()
print (len(all_user_test_set))
all_user_test_item_pair = x_test[x_test['pred_label'] == 1]['user_id'].map(str) + '-' + x_test[x_test['pred_label'] == 1]['sku_id'].map(str)
all_user_test_item_pair = np.array(all_user_test_item_pair)
print (len(all_user_test_item_pair))
```

```
126
224
243
```

```
pos, neg = 0,0
for user_id in all_user_test_set:
    if user_id in all_user_set:
        pos += 1
    else:
        neg += 1
all_user_acc = 1.0 * pos / ( pos + neg)
all_user_recall = 1.0 * pos / len(all_user_set)
print ('所有用户中预测购买用户的准确率为 ' + str(all_user_acc))
print ('所有用户中预测购买用户的召回率' + str(all_user_recall))
```

```
所有用户中预测购买用户的准确率为 0.5357142857142857
所有用户中预测购买用户的召回率0.9523809523809523
```

```
#所有实际商品对
all_user_item_pair = x_test[x_test['true_label']==1]['user_id'].map(str) + '-' + x_test[x_test['true_label']==1]['sku_id'].map(str)
all_user_item_pair = np.array(all_user_item_pair)
#print (len(all_user_item_pair))
#print(all_user_item_pair)
pos, neg = 0, 0
for user_item_pair in all_user_test_item_pair:
    #print (user_item_pair)
    if user_item_pair in all_user_item_pair:
        pos += 1
    else:
        neg += 1
all_item_acc = 1.0 * pos / ( pos + neg)
all_item_recall = 1.0 * pos / len(all_user_item_pair)
print ('所有用户中预测购买商品的准确率为 ' + str(all_item_acc))
print ('所有用户中预测购买商品的召回率' + str(all_item_recall))
F11 = 6.0 * all_user_recall * all_user_acc / (5.0 * all_user_recall + all_user_acc)
F12 = 5.0 * all_item_acc * all_item_recall / (2.0 * all_item_recall + 3 * all_item_acc)
score = 0.4 * F11 + 0.6 * F12
print ('F11=' + str(F11))
print ('F12=' + str(F12))
print ('score=' + str(score))
```

```
所有用户中预测购买商品的准确率为 0.5679012345679012
所有用户中预测购买商品的召回率0.9583333333333334
F11=0.5778491171749598
F12=0.7516339869281046
score=0.6821200390268466
```