Final

March 15, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
```

[28]:
```
data = pd.read_csv('/Users/mac/Desktop/ /WISERCLUB/Final/data.csv').dropna()
```

[25]:
```
data.columns.values
```

[25]:
```
array(['Unnamed: 0', 'Unnamed: 0.1', 'dt', 'phone_no', 'member_id',
       'commodity_code', 'commodity_name', 'commodity_origin_money',
       'coupon_id', 'coupon_money', 'one_category_name',
       'two_category_name', 'commodity_income', 'pay_money',
       'coffeestore_share_money'], dtype=object)
```

0.1 Q1.1 Find the time span of the order data.

Acording to the result, we can see the time span is from 2019 Jan 20th to 2019 Mar 1st

[29]:
```
np.sort(data['dt'].unique())
```

[29]:
```
array(['2019-01-20', '2019-01-21', '2019-01-22', '2019-01-23',
       '2019-01-24', '2019-01-25', '2019-01-26', '2019-01-27',
       '2019-01-28', '2019-01-29', '2019-01-30', '2019-01-31',
       '2019-02-01', '2019-02-02', '2019-02-03', '2019-02-04',
       '2019-02-05', '2019-02-06', '2019-02-07', '2019-02-08',
       '2019-02-09', '2019-02-10', '2019-02-11', '2019-02-12',
       '2019-02-13', '2019-02-14', '2019-02-15', '2019-02-16',
       '2019-02-17', '2019-02-18', '2019-02-19', '2019-02-20',
       '2019-02-21', '2019-02-22', '2019-02-23', '2019-02-24',
       '2019-02-25', '2019-02-26', '2019-02-27', '2019-02-28',
       '2019-03-01'], dtype=object)
```

0.2 Q1.2 Find the number of orders each day.

According to the groupby sentence, we can see the result

0.2.1 Do we need to design two different strategies for sales in workdays and sales in weekends.?

Although the number of sales are evenly low From Feb 4th to Feb 10th, that is because it is the Spring Festival in China, in other times of the year, we can see the trend that the number of orders in workdays are significantly larger than the number of orders in weekends.

```
[26]: data.groupby('dt')['Unnamed: 0'].count()
```

```
[26]: dt
2019-01-20    35566
2019-01-21    66503
2019-01-22    69318
2019-01-23    75717
2019-01-24    77973
2019-01-25    75889
2019-01-26    47252
2019-01-27    40411
2019-01-28    66383
2019-01-29    65133
2019-01-30    62009
2019-01-31    59121
2019-02-01    52242
2019-02-02    37236
2019-02-03    24390
2019-02-04     4108
2019-02-05     5085
2019-02-06     4585
2019-02-07     4408
2019-02-08     4238
2019-02-09     4501
2019-02-10     4072
2019-02-11    17160
2019-02-12    21985
2019-02-13    26773
2019-02-14    24514
2019-02-15    26079
2019-02-16    14675
2019-02-17    12222
2019-02-18    29899
2019-02-19    31482
2019-02-20    31427
2019-02-21    32281
2019-02-22    33708
2019-02-23    19153
2019-02-24    16237
2019-02-25    34267
```

```
2019-02-26    31557
2019-02-27    31065
2019-02-28    33428
2019-03-01    35842
Name: Unnamed: 0, dtype: int64
```

0.3 Q1.3 Find the number of users.

[7]:
```python
len(data['member_id'].unique())
```

[7]:
```
389394
```

0.4 Q1.4 Find ten commodities with the highest sales,Draw graphs with x-axis the commodity name and y-axis the # of orders.

[8]:
```python
data['commodity_name'].value_counts(ascending = False).head(10)
```

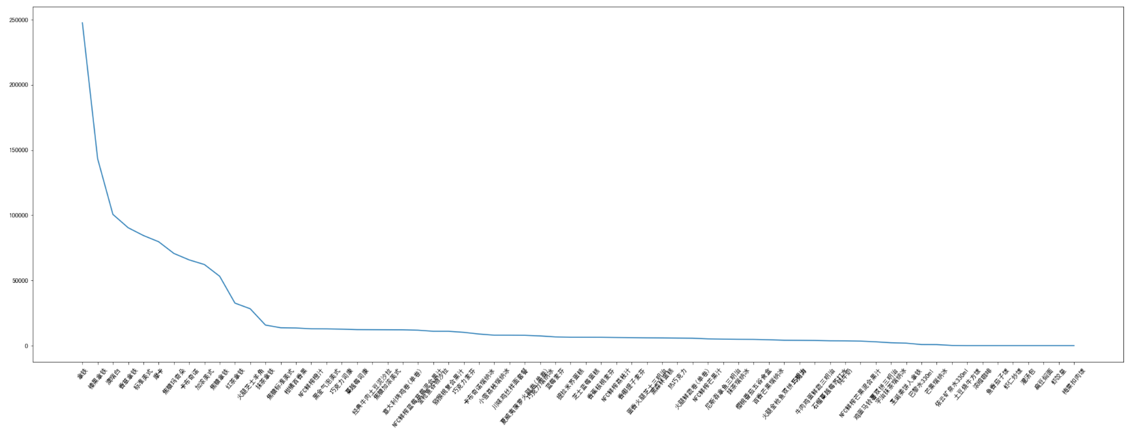
[8]:
```
         247597
        143337
         100645
          90411
          84436
           79657
         70681
          65737
          62174
          53178
Name: commodity_name, dtype: int64
```

[9]:
```python
x = data['commodity_name'].value_counts().index.tolist()
```

[10]:
```python
y = data['commodity_name'].value_counts().values
```

[11]:
```python
plt.rcParams['font.sans-serif']=['SimHei']
plt.figure(figsize=(30, 10))
plt.xticks(rotation=50)
plt.plot(x,y)
```

[11]:
```
[<matplotlib.lines.Line2D at 0x126e30940>]
```



0.5 Q1.5 Find the discount rate of each order and concat it onto the original dataset with column name discount_rate. You may use pay_money, coffeestore_share_money, commodity_origin_money and commodity_income.

```
[12]: discount =␣
       ↪data[['pay_money','coffeestore_share_money','commodity_origin_money','commodity_income']]
```

```
[13]: commodity_income = discount['commodity_income']
      commodity_origin_money= discount['commodity_origin_money']
      discount_rate = commodity_income/commodity_origin_money
```

```
[14]: discount_rate = discount_rate.rename('discount_rate')
```

```
[15]: data_discount = pd.concat([data,discount_rate],axis=1)
```

0.6 Q1.6 Find the average discount of each week. One week should consist of Sunday to Saturday.

```
[16]: data_discount['dt']=pd.to_datetime(data_discount['dt'])
```

```
[17]: data_discount['dayofweek'] = data_discount['dt'].dt.dayofweek
```

```
[18]: dt = data_discount['dt'].tolist()
```

```
[19]: data_discount['wn']=pd.to_datetime(dt).strftime("%U")
```

```
[20]: data_discount.groupby('wn')['discount_rate'].mean()
```

[20]:
```
wn
03    0.314009
04    0.348994
05    0.310017
06    0.413531
07    0.452827
08    0.463887
Name: discount_rate, dtype: float64
```

0.7 Q1.7 Find the Retention Rate of any five days. It is the ratio of users purchasing again on the next day. For example, if you want to compute the Retention Rate on 2019-02-10, then you need to find users who bought goods on 02-09 and 02-10.

[64]:
```
def retention_rate(y,m,d):
    begin = datetime.date(y,m,d)
    end = begin + datetime.timedelta(days=1)
    a = set(data[data['dt']==str(begin)]['phone_no'])
    b = set(data[data['dt']==str(end)]['phone_no'])
    c = a.intersection(b)
    rate = len(set(c))/len(set(a))
    return rate
```

[65]:
```
retention_rate(2019,1,28)
```

[65]:
```
0.07337974311206981
```

0.8 Q1.8 Find the Week Retention Rate of any day, which means finding users buying at that day and buying again within the next seven days.

[72]:
```
def week_retention_rate(y,m,d):
    begin = datetime.date(y,m,d)
    end = begin + datetime.timedelta(days=1)
    a = set(data[data['dt']==str(begin)]['phone_no'])
    b = set(data[data['dt']==str(end)]['phone_no'])
    old = set(data[data['dt']>str(begin)]['phone_no'])
    new = a.difference(old)
    c = new.intersection(b)
    rate = len(set(c))/len(set(new))
    return rate
```

[67]:
```
week_retention_rate(2019,1,28)
```

[67]:
```
0.5505441709971566
```

0.9 Q1.9 Find the Week Retention Rate of any day for new users , which means finding users buying at that day for the first time and buying again within the next seven days.

```
[70]: def week_retention_rate_newusers(y,m,d):
          begin = datetime.date(y,m,d)
          a = set(data[data['dt']==str(begin)]['phone_no'])
          old = set(data[data['dt']<str(begin)]['phone_no'])
          new = a.difference(old)
          phone_no = []
          for i in range(1,8):
              dt = begin + datetime.timedelta(days=i)
              b = set(data[data['dt']==str(dt)]['phone_no'])
              c = new.intersection(b)
              phone_no.extend(c)
          rate = len(set(phone_no))/len(set(new))
          return rate
```

0.10 Q1.10 Find the Retention Rate WITHIN one week of new users. You could choose any week you want, but it must consist of Sunday to Saturday. You need to find users buying the first product and buying again within that week.

```
[77]: data_wn = data
      dt = data_wn['dt'].tolist()
      data_wn['wn']=pd.to_datetime(dt).strftime("%U")
```

```
[95]: def oneweek_retention_rate_newusers(x,y):
          a = set(data_wn[data_wn['wn']==x]['phone_no'])
          b = set(data_wn[data_wn['wn']==y]['phone_no'])
          old = set(data[data['wn']<x]['phone_no'])
          new = a.difference(old)
          c = new.intersection(b)
          rate = len(set(c))/len(set(new))
          return rate
```

```
[96]: oneweek_retention_rate_newusers('06','07')
```

```
[96]: 0.23986682107701215
```

0.11 Q1.11 Find "Active Users" (which means the number of orders of one user is greater equal to 5).

[34]:
```
active=data['phone_no'].value_counts()
```

[35]:
```
active_users = active[active >=5]
```

0.12 Q1.12 Write the table you get in 11 as a csv file with filename ActiveUser.csv.

[5]:
```
import os
```

[12]:
```
path = os.getcwd()
path1 = path + r'/active_users.csv'
```

[10]:
```
active_users.to_csv(path1)
```

```
/Users/mac/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
FutureWarning: The signature of `Series.to_csv` was aligned to that of
`DataFrame.to_csv`, and argument 'header' will change its default value from
False to True: please pass an explicit value to suppress this warning.
  """Entry point for launching an IPython kernel.
```

0.13 Q1.13 Provide a description of the number of orders for each active user (# of ActiveUser, mean, range) Python Users Only : Find the std, variance, skewness and kurtosis.

[43]:
```
active_users.agg(['mean','max','min','std','var','skew','kurt'])
```

[43]:
```
mean      8.986923
max     564.000000
min       5.000000
std       5.700658
var      32.497505
skew     12.481889
kurt    973.308878
Name: phone_no, dtype: float64
```

0.14 Q2.1 Remove the first column of the data in data.csv , because it is just a copy of index.

[28]:
```
data = data.drop(['Unnamed: 0'], axis=1)
```

0.15 Q2.2 Boss: To implement Collaborative Filtering in recommendation systems, we need a user-item table to show the number of orders for each user and each item.

Try to construct user-item table. An example of user-item pair: (Phone_No,   )

[25]:
```
data.pivot_table('Unnamed: 0.1',index ='phone_no',columns =␣
↪'commodity_name',aggfunc = 'count',fill_value = 0)
```

[25]:
```
commodity_name  NFC     NFC      NFC        NFC      NFC          \
phone_no
51379898              0          0              0          0               0
57047978              0          0              0          0               0
61120518              0          0              0          0               0
62288158              0          0              0          0               1
64618166              0          0              0          0               0
…                  …        …          …        …             …
19997912482           0          0              0          0               0
19999208056           0          0              0          0               0
19999597999           0          0              0          0               0
33757911877           0          0              0          0               0
85293820835           0          0              0          0               0

commodity_name     330ml                     …             \
phone_no                                                 …
51379898                 0      0      0         0       0  …    0           0
57047978                 0      0      0         0       0  …    0           0
61120518                 0      0      0         0       0  …    0           0
62288158                 0      0      0         0       0  …    0           0
64618166                 0      0      0         0       0  …    0           0
…                     …    …    …      …      …   …   …      …
19997912482              0      1      0         0       0  …    0           0
19999208056              0      0      0         0       0  …    0           0
19999597999              0      1      0         0       0  …    0           0
33757911877              0      0      0         0       0  …    0           0
85293820835              0      0      0         0       0  …    0           0

commodity_name                                    \
phone_no
51379898              0        0      0        0       0            0       0
57047978              0        0      0        0       0            0       0
61120518              0        0      0        0       0            0       0
62288158              0        0      0        0       0            0       0
64618166              0        0      0        0       0            0       0
…                  …      …    …      …     …         …     …
19997912482           0        0      0        0       0            0       0
19999208056           0        0      0        0       0            0       0
```

```
19999597999            0         0      0         0       0             0       1
33757911877            0         0      0         0       0             0       0
85293820835            0         0      0         0       0             0       0

commodity_name
phone_no
51379898              0
57047978              0
61120518              0
62288158              0
64618166              0
…                  …
19997912482           0
19999208056           0
19999597999           0
33757911877           0
85293820835           0

[389394 rows x 66 columns]
```

0.16 Boss: Life is not like a Markov Chain, which means everyone's past behavior is correlated with his present one. And that is why we could exploit past purchase behavior to predict their future buying trends.

Try to construct a dataset to show this past purchasing behavior trend. For convenience, several instructions are proposed as follows

a. Two days correspond to one dimension.

b. The last two days of the time span of the data should be the future, which means it corresponds to the target field for the following data mining models.

c. The length of each user vector must be maximized.

d. The dataset should be a DataFrame in Pandas, so you could customize the columns as you wish. For example, if the time span is from 2019-02-01 to 2019-02-10, then there are 10 days altogether. So each user corresponds to a 5-dimensional vector, with 4 features and 1 target dimension. The vector [4, 0, 0, 0, 1] means this user bought one good between 02-09 and 02-10, and four goods between 02-01 and 02-02. Additionally, the length of each user vector MUST BE 5 because of the rule 3.

```
[3]: data['dt']=pd.to_datetime(data['dt'])
     original = data.set_index('dt')[['phone_no','commodity_name']]
```

```
[4]: trend = original.pivot_table(index = 'phone_no', columns ='dt' ,aggfunc =␣
     ↪'count',fill_value = 0)
```

[5]:
```
trend_result= trend.stack('dt').unstack('phone_no').resample('2D').sum().
 ↪stack('phone_no').unstack('dt')
```

[5]:
```
len(data['phone_no'].unique())
```

[5]:
```
389394
```

0.17 Part3

0.17.1 Boss: For the target field, 1 means he purchased in the future and 0 means he did not. Then you could use traditional classification algorithms to predict the future behaviors of all users.

1. Transform the data you got from the last section into an array in Numpy.

[6]:
```
trend_result.iloc[:,20][trend_result.iloc[:,20]!=0]=1
```

[7]:
```
trend_result = trend_result.to_numpy()
```

2. Split the data into features X and targets Y.

[8]:
```
Y = trend_result[:,-1]
```

[9]:
```
X = trend_result[:,0:-1]
```

3. Use Adaboost, Random Forest in Sklearn to construct the model for prediction with 3-fold cross validation.a. (Optional) Use Xgboost.b. Boss: We could, but we do not use Naive Bayes or Support Vector Machine in this project. True of False? Explain.

[10]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_predict
```

[11]:
```
clf=RandomForestClassifier()
```

[20]:
```
print(np.mean(cross_val_score(clf, X, Y, cv=3)))
```

```
/Users/mac/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/Users/mac/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
/Users/mac/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
0.9324771393797583
```

[23]:
```
clf1 = AdaBoostClassifier()
```

[24]:
```
print(np.mean(cross_val_score(clf1, X, Y, cv=3)))
```

```
0.935805383991398
```

[36]:
```
len(Y[Y==0])/len(Y) # if the model classify each sample as 0, the accuracy is␣
 ↪93.51428%
```

[36]:
```
0.9351428116509243
```

0.17.2 We can tell that although the accuracy is high for both model, however, the random forest rate is lower than the ratio of 0 in the data, and adaboost is just a slightly higher than the orignal rate.