Ran Song - Programming Skills Demo

January 19, 2022

[1]:
```
import os
import numpy as np
import pandas as pd
```

[11]:
```
# read train datasheet from 3rd row
pd.read_excel('./eloData/Data_Dictionary.xlsx', header=2, sheet_name='train')
```

[11]:

	Columns	Description
0	card_id	Unique card identifier
1	first_active_month	'YYYY-MM', month of first purchase
2	feature_1	Anonymized card categorical feature
3	feature_2	Anonymized card categorical feature
4	feature_3	Anonymized card categorical feature
5	target	Loyalty numerical score calculated 2 months af…

[34]:
```
pd.read_csv('./eloData/sample_submission.csv', header=0).head(5)
```

[34]:

	card_id	target
0	C_ID_0ab67a22ab	0
1	C_ID_130fd0cbdd	0
2	C_ID_b709037bc5	0
3	C_ID_d27d835a9f	0
4	C_ID_2b5e3df5c2	0

[37]:
```
# check basic information
pd.read_csv('./eloData/sample_submission.csv', header=0).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123623 entries, 0 to 123622
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   card_id  123623 non-null  object
 1   target   123623 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.9+ MB
```

0.1 Train & Test Data

0.1.1 1.train & test dataset

Considering the large scale of subsequent datasets, we can import the gc package in advance for memory management. When actually cleaning up the memory, use `del` to delete the object first, and then use `gc.collect()` to manually cleaning up the memory.

[3]:
```
import gc
```

[4]:
```
train = pd.read_csv('./eloData/train.csv')
test =  pd.read_csv('./eloData/test.csv')
```

[6]:
```
(train.shape, test.shape)
```

[6]:
```
((201917, 6), (123623, 5))
```

[7]:
```
train.head(5)
```

[7]:

	first_active_month	card_id	feature_1	feature_2	feature_3	target
0	2017-06	C_ID_92a2005557	5	2	1	-0.820283
1	2017-01	C_ID_3d0044924f	4	1	0	0.392913
2	2016-08	C_ID_d639edf6cd	2	2	0	0.688056
3	2017-09	C_ID_186d6a6901	4	3	0	0.142495
4	2017-11	C_ID_cdbd2c0db2	1	3	0	-0.159749

[8]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201917 entries, 0 to 201916
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   first_active_month  201917 non-null  object
 1   card_id             201917 non-null  object
 2   feature_1           201917 non-null  int64
 3   feature_2           201917 non-null  int64
 4   feature_3           201917 non-null  int64
 5   target              201917 non-null  float64
dtypes: float64(1), int64(3), object(2)
memory usage: 9.2+ MB
```

[10]:
```
pd.read_excel('./eloData/Data_Dictionary.xlsx', header=2, sheet_name='train')
```

[10]:

	Columns	Description
0	card_id	Unique card identifier
1	first_active_month	'YYYY-MM', month of first purchase
2	feature_1	Anonymized card categorical feature
3	feature_2	Anonymized card categorical feature
4	feature_3	Anonymized card categorical feature
5	target	Loyalty numerical score calculated 2 months af…

[12]:
```
test.head(5)
```

[12]:

	first_active_month	card_id	feature_1	feature_2	feature_3
0	2017-04	C_ID_0ab67a22ab	3	3	1
1	2017-01	C_ID_130fd0cbdd	2	3	0
2	2017-08	C_ID_b709037bc5	5	1	1
3	2017-12	C_ID_d27d835a9f	2	1	0
4	2015-12	C_ID_2b5e3df5c2	5	1	1

[13]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123623 entries, 0 to 123622
Data columns (total 5 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   first_active_month  123622 non-null  object
 1   card_id             123623 non-null  object
 2   feature_1           123623 non-null  int64 
 3   feature_2           123623 non-null  int64 
 4   feature_3           123623 non-null  int64 
dtypes: int64(3), object(2)
memory usage: 4.7+ MB
```

0.1.2 2. Analyze data quality

Next is simple data exploration. In the actual modeling process, we will first check the correctness of the data, and check for missing values, outliers, etc.

[18]:
```
train['card_id'].nunique() == train.shape[0]
```

[18]: True

[17]:
```
test['card_id'].nunique() == test.shape[0]
```

[17]: True

[21]:
```
test['card_id'].nunique()+ train['card_id'].nunique()  ==␣
↪len(set(test['card_id'].values.tolist() + train['card_id'].values.tolist()))
```

[21]:
```
True
```

[24]:
```
train.isnull().sum()
```

[24]:
```
first_active_month    0
card_id               0
feature_1             0
feature_2             0
feature_3             0
target                0
dtype: int64
```

[23]:
```
test.isnull().sum()
```

[23]:
```
first_active_month    1
card_id               0
feature_1             0
feature_2             0
feature_3             0
dtype: int64
```

[52]:
```
statistics = train['target'].describe()
statistics
```

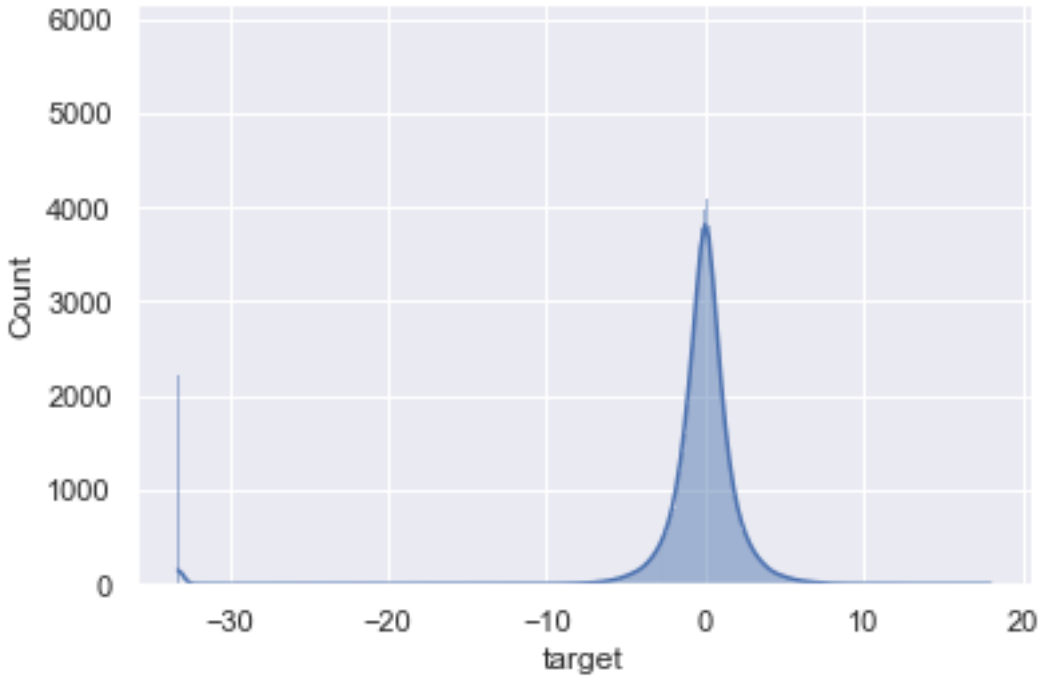
[52]:
```
count    201917.000000
mean         -0.393636
std           3.850500
min         -33.219281
25%          -0.883110
50%          -0.023437
75%           0.765453
max          17.965068
Name: target, dtype: float64
```

Since it is a continuous variable, we can observe the distribution using a probability density histogram:

[27]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[51]:
```
sns.set()
sns.histplot(train['target'], kde=True)
```

[51]:
```
<AxesSubplot:xlabel='target', ylabel='Count'>
```



It can be found that most of the user loyalty scores are concentrated between `[-10, 10]` and conform to the normal distribution. The only thing to note is that there are individual outliers below `-30`. This data will be used in the subsequent analysis. Additional attention is required. We can simply see how many users have a tag value less than `30`:

[74]:
```
(train['target'] < -30).sum()  # 1% of entire 200K dataset
```

[74]: 2207

for continuous variables, normally we use 3δ principle to check for abnormal data.

[62]:
```
statistics.loc['mean'] - 3 * statistics.loc['std']
```

[62]: -11.945136285536126

0.1.3 4. Consistency Analysis

We first compare the simple univariate distribution. Since the four variables are all discrete variables, we can compare through relative probability distribution. For example, first we look at the relative distribution of the first activation month

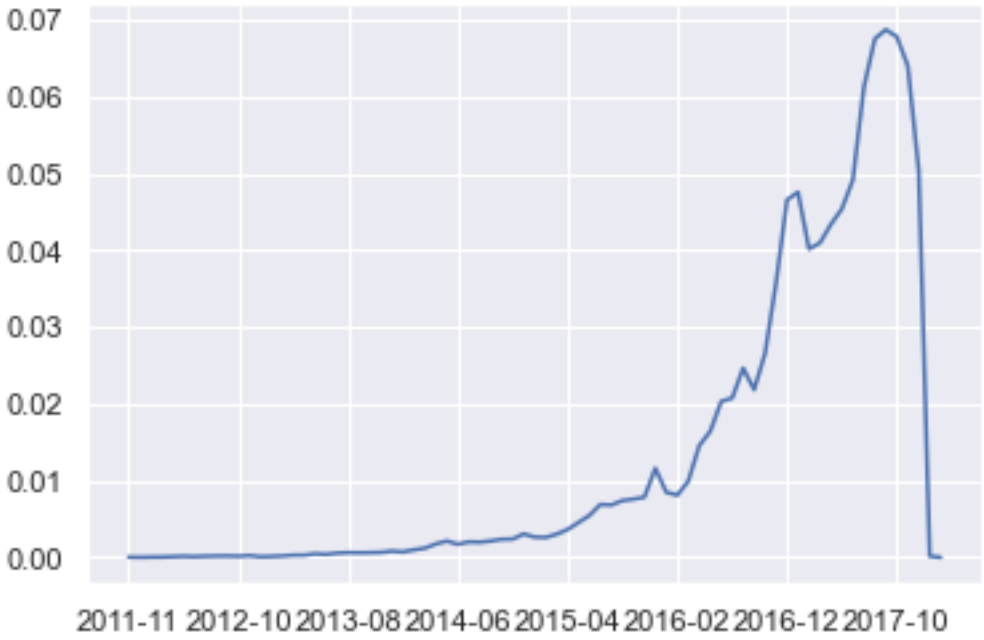
[80]:
```
features = ['first_active_month','feature_1','feature_2','feature_3']

train_count = train.shape[0]
test_count = test.shape[0]
```

[83]:
```
train['first_active_month'].value_counts().sort_index()/train_count
```

[83]:
```
2011-11    0.000040
2011-12    0.000010
2012-02    0.000035
2012-03    0.000050
2012-04    0.000089
             …
2017-10    0.067825
2017-11    0.064036
2017-12    0.050367
2018-01    0.000168
2018-02    0.000005
Name: first_active_month, Length: 75, dtype: float64
```

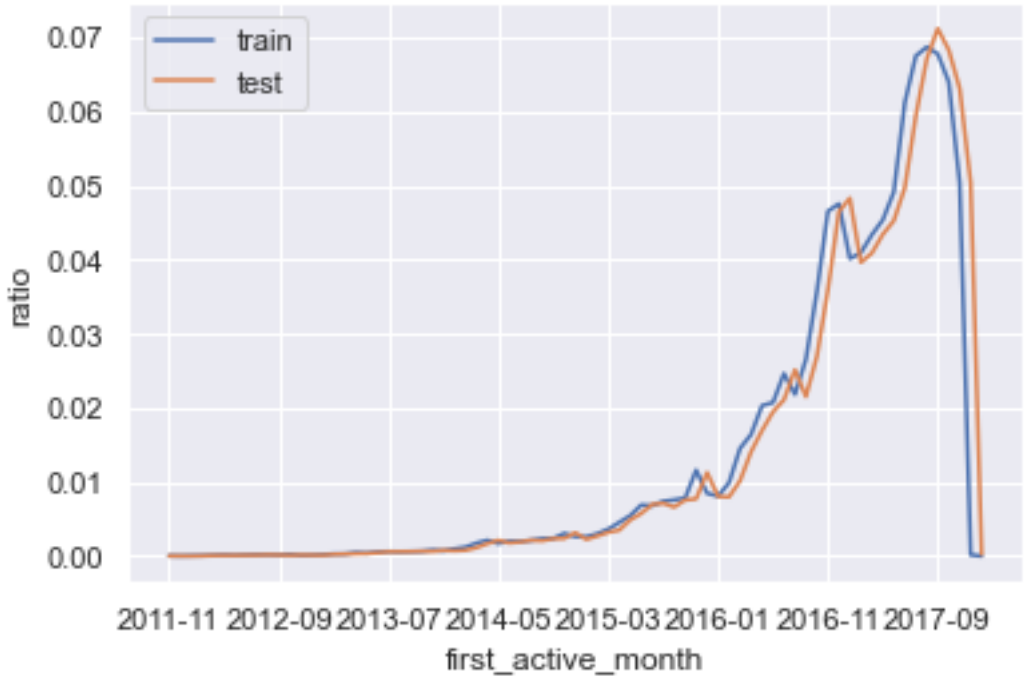
[85]:
```
(train['first_active_month'].value_counts().sort_index()/train_count).plot()
```

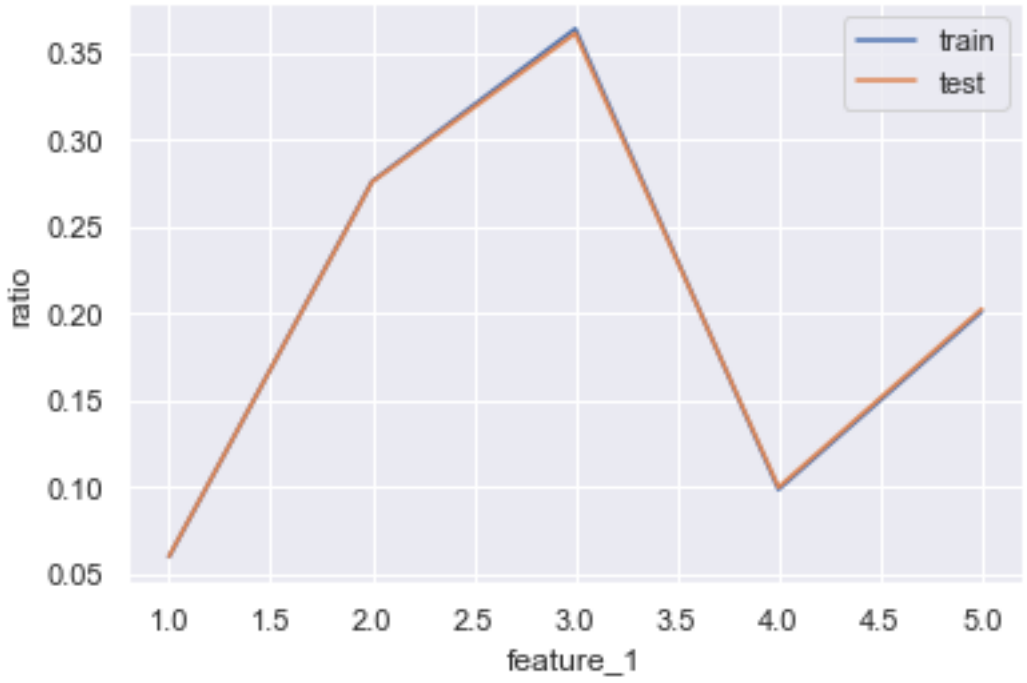
[85]:
```
<AxesSubplot:>
```

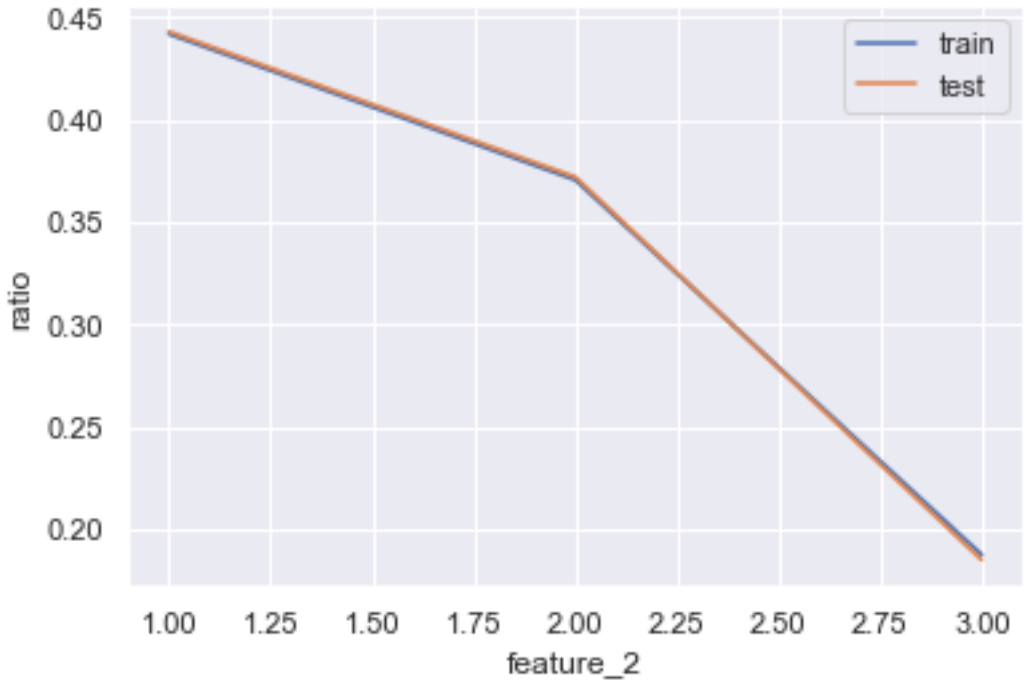


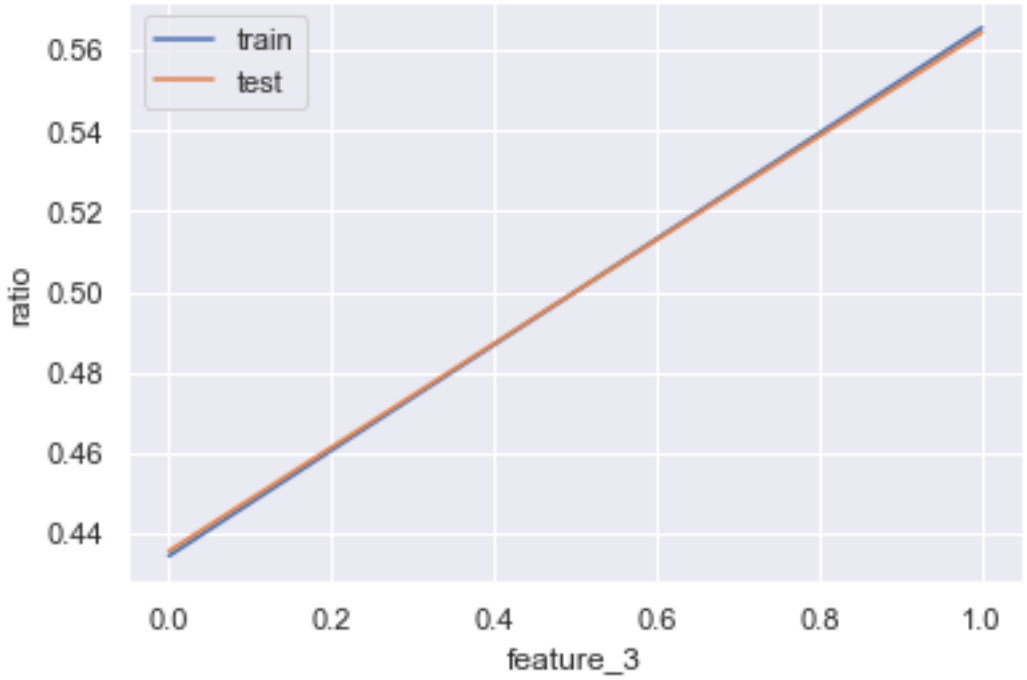
[87]:
```
for feature in features:
    (train[feature].value_counts().sort_index()/train_count).plot()
    (test[feature].value_counts().sort_index()/test_count).plot()
    plt.legend(['train','test'])
    plt.xlabel(feature)
    plt.ylabel('ratio')
```

```
plt.show()
```


train
test
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
ratio
2011-11
2012-09
2013-07
2014-05
2015-03
2016-01
2016-11
2017-09
first_active_month


train
test
0.35
0.30
0.25
0.20
0.15
0.10
0.05
ratio
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
feature_1


train
test
ratio
0.45
0.40
0.35
0.30
0.25
0.20
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
feature_2


train
test
ratio
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.0
0.2
0.4
0.6
0.8
1.0
feature_3

[ ]: