Coupon Using Prediction of Tmall Customers

Background: "Tmall", formerly known as Taobao Mall, is a comprehensive on-line shopping website and is a brand new B2C (Business-to-Consumer) brand created by Ma Yun Taobao. It integrates thousands of brands and manufacturers, providing one-stop solutions for merchants and consumers, providing 100% quality-guaranteed goods, and good after-sales service such as 7-day return without reason, and shopping-points rewards.

Question: Based on the data given, using logistic regression to predict whether the customers will use the coupon.

1. Check the data dictionary and know the meaning of each field

Column	definition
ID	record id
age	age
job	job title
marital	marital status
default	wheather got default in HuaBei
returned	returned products or not before
loan	billed with Huabei or not
coupon_used_in_last6_month	the number of coupon used in last 6 months
coupon_used_in_last_month	the number of coupon used in last 1 months
coupon_ind	used coupon or not in current activity

2. Data Cleaning and Pre-processing (abnormal data clean, variable conversion/rename, etc.)

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [2]:
```
Tmall=pd.read_csv('L2_Week3.csv')
```

Data cleaning and pre-processing

In [59]:
```
Tmall.info()
```

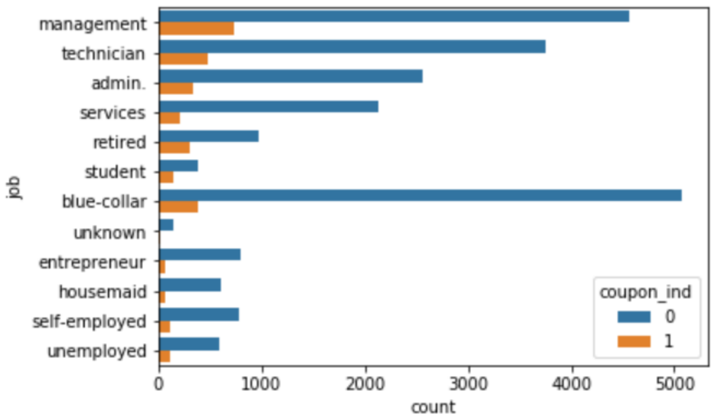
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25317 entries, 0 to 25316
Data columns (total 10 columns):
ID                              25317 non-null int64
age                             25317 non-null int64
job                             25317 non-null object
marital                         25317 non-null object
default                         25317 non-null object
returned                        25317 non-null object
loan                            25317 non-null object
coupon_used_in_last6_month      25317 non-null int64
coupon_used_in_last_month       25317 non-null int64
coupon_ind                      25317 non-null int64
dtypes: int64(5), object(5)
memory usage: 1.9+ MB
```

```
In [60]: Tmall.job.value_counts() #got unknown value inside
```

```
Out[60]: blue-collar      5456
         management       5296
         technician       4241
         admin.           2909
         services         2342
         retired          1273
         self-employed     884
         entrepreneur      856
         unemployed        701
         housemaid         663
         student           533
         unknown           163
         Name: job, dtype: int64
```

```
In [61]: sns.countplot(y='job',hue='coupon_ind',data=Tmall) #check the coupon used status of different jobs
```

```
Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0xe30db00>
```



The number of job_unknown is quiet few and most are the ones didn't use the coupon, this part got low value for analysis; users in management used coupons more than other customers while the number of not used coupon also quiet high, so as the users in blue-collar; from the diagram, the job's effect on coupon using is not so obvious.

```
In [62]: #check the field which type is object if got unknown values inside
         Tmall.marital.value_counts()
```

```
Out[62]: married     15245
         single       7157
         divorced     2915
         Name: marital, dtype: int64
```

```
In [63]: Tmall.default.value_counts()
```

```
Out[63]: no      24869
         yes       448
         Name: default, dtype: int64
```

```
In [64]: Tmall.returned.value_counts()
```

```
Out[64]: yes     14020
         no      11297
         Name: returned, dtype: int64
```

```
In [65]: Tmall.loan.value_counts()
```

```
Out[65]: no      21258
         yes      4059
         Name: loan, dtype: int64
```

In [3]:
```
#convert variables to numerical variable
Tmall=pd.get_dummies(Tmall)
```

In [67]:
```
Tmall.head()
```

Out[67]:

	ID	age	coupon_used_in_last6_month	coupon_used_in_last_month	coupon_ind	job_admin.	jol
0	1	43	2	0	0	0	
1	2	42	1	1	0	0	
2	3	47	2	0	0	1	
3	4	28	2	0	0	0	
4	5	42	5	0	0	0	

5 rows × 26 columns

In [4]:
```
Tmall.drop(['ID','job_unknown','marital_divorced','default_no','returned_no','loan_no'],axis=1,inplace=True)
```

In [5]:
```
Tmall=Tmall.rename(columns={'coupon_ind':'flag'})
```

n [78]:
```
Tmall.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25317 entries, 0 to 25316
Data columns (total 20 columns):
age                          25317 non-null int64
coupon_used_in_last6_month   25317 non-null int64
coupon_used_in_last_month    25317 non-null int64
flag                         25317 non-null int64
job_admin.                   25317 non-null uint8
job_blue-collar              25317 non-null uint8
job_entrepreneur             25317 non-null uint8
job_housemaid                25317 non-null uint8
job_management               25317 non-null uint8
job_retired                  25317 non-null uint8
job_self-employed            25317 non-null uint8
job_services                 25317 non-null uint8
job_student                  25317 non-null uint8
job_technician               25317 non-null uint8
job_unemployed               25317 non-null uint8
marital_married              25317 non-null uint8
marital_single               25317 non-null uint8
```

3. Select the key variables for modeling

Analyze 0/1 ratio of target variable and correlation with other variables

[79]:
```
Tmall.flag.value_counts(1)#sample is unbalanced, need over/under-sampling to reach balance
```

[79]:
```
0    0.883043
1    0.116957
Name: flag, dtype: float64
```

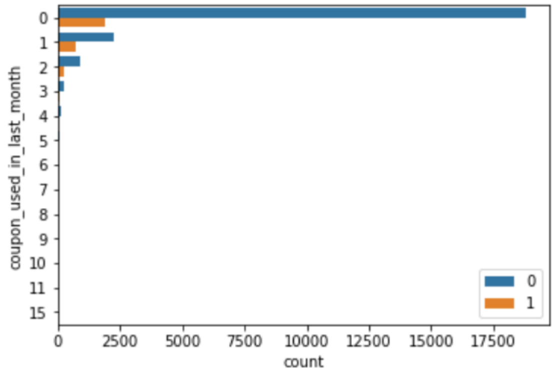
[80]:
```
Tmall.groupby('flag').mean()#correlation analysis
```

[80]:

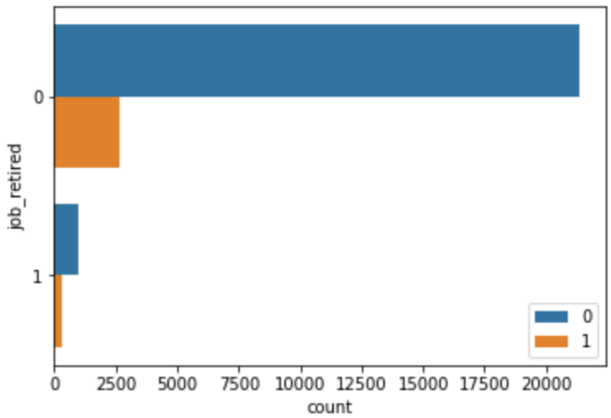
flag	age	coupon_used_in_last6_month	coupon_used_in_last_month	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_mai
0	40.819601	2.857846	0.260378	0.114868	0.226740	0.035293	0.027062	
1	41.809524	2.124282	0.537994	0.115164	0.130699	0.022627	0.019588	

The sample is unbalanced, need to do some actions for optimization afterwards.
But first, we can analyze from mean_summary: **coupon_used_in_last_month, job_retired, returned_yes** 0/1 differ a lot, can do simple visualization to analyze first.

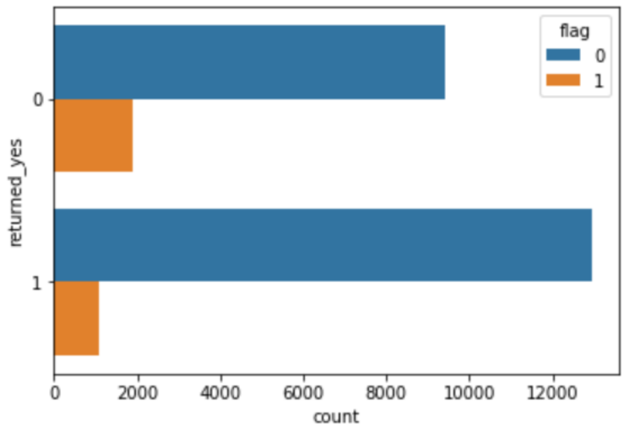
[91]:
```
sns.countplot(y='coupon_used_in_last_month',hue='flag',data=Tmall)
plt.legend(loc='lower right')
#coupon_flag both performanced low in the number of coupon_used_in_last_month, so this feature has the low reference value
```
ut[91]: <matplotlib.legend.Legend at 0x10b74160>



[90]:
```
sns.countplot(y='job_retired',hue='flag',data=Tmall)
plt.legend(loc='lower right')
```
ıt[90]: <matplotlib.legend.Legend at 0x10723da0>



[84]:
```
sns.countplot(y='returned_yes',hue='flag',data=Tmall)
#it's more possible to use the coupon for those who never returned before
```
t[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1061cb00>



Can be found that there's the weak effects on coupon using for these three variables.

[85]:

```
#coupon_used_in_last_month & returned_yes impact more on flag
Tmall.corr()[['flag']].sort_values('flag',ascending=False)
```

t[85]:

	flag
flag	1.000000
coupon_used_in_last_month	0.116550
job_retired	0.083868
job_student	0.069058
marital_single	0.057574
job_management	0.035234
age	0.029916
job_unemployed	0.023980
job_self-employed	0.001078
job_admin.	0.000298
job_technician	-0.004942
job_housemaid	-0.015041
job_entrepreneur	-0.022519
default_yes	-0.024608
job_services	-0.026688
marital_married	-0.054746
loan_yes	-0.065231
job_blue-collar	-0.075065
coupon_used_in_last6_month	-0.075173
returned_yes	-0.143589

4. Data Modeling

Data Modeling

In [79]:

```
x=Tmall[['coupon_used_in_last_month','job_retired','loan_yes','returned_yes']]
y=Tmall['flag']
```

In [80]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=100)
```

In [81]:

```
from sklearn import linear_model
lr=linear_model.LogisticRegression()
```

In [82]:

```
lr.fit(x_train,y_train)
```

Out[82]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [83]:
```
lr.intercept_
```
Out[83]: array([-1.70490371])

In [84]:
```
lr.coef_
```
Out[84]: array([[0.38253494, 0.65367682, -0.55787015, -0.90405667]])

Coefficient analysis: 1.coupon_used_in_last_month, job_retired showed positive correlation;
loan_yes, returned_yes showed negative correlation;

2.if used coupon in last month, then the possibility of using current coupon is e^0.38=1.46 times more than the other customers who didn't use coupon last month;

3.the using-current-coupon's possibility of people retired is e^0.38=1.46 times that of people have jobs;

4.the customer who used Huabei for billing maybe won't use current coupon, the possibility only is e^(-0.56)=0.57 times those who didn't use Huabei;

5.the customer who returned before also maybe won't use current coupon, the possibility only is e^(-0.9)=0.41 times those who never returned.

5. Model Assessment

Model Assessment

[85]:
```
y_pred_train=lr.predict(x_train)
y_pred_test=lr.predict(x_test)
```

[86]:
```
import sklearn.metrics as metrics
from sklearn.metrics import (classification_report,
                             roc_curve, auc)
```

[87]:
```
#check the confusion_matrix
metrics.confusion_matrix(y_train, y_pred_train)
```
t[87]:
```
array([[15589,    34],
       [ 2085,    13]], dtype=int64)
```

[88]:
```
metrics.confusion_matrix(y_test, y_pred_test)
```
t[88]:
```
array([[6721,   12],
       [ 862,    1]], dtype=int64)
```

n [89]:
```
#check the accuracy
metrics.accuracy_score(y_train, y_pred_train)
```
Out[89]: 0.8804243552846904

n [90]:
```
metrics.accuracy_score(y_test, y_pred_test)
```
Out[90]: 0.8849394418114798

```
n [91]: #check the precision&recall&F1-score
print(classification_report(y_train, y_pred_train))
```

```
             precision    recall  f1-score   support

          0       0.88      1.00      0.94     15623
          1       0.28      0.01      0.01      2098

avg / total       0.81      0.88      0.83     17721
```

```
n [92]: print(classification_report(y_test, y_pred_test))
```

```
             precision    recall  f1-score   support

          0       0.89      1.00      0.94      6733
          1       0.08      0.00      0.00       863

avg / total       0.79      0.88      0.83      7596
```

```
n [93]: #check the area under the curve
fpr, tpr, threshold=roc_curve(y_train, y_pred_train)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

```
0.5020100494693636
```

```
n [94]: fpr, tpr, threshold=roc_curve(y_test, y_pred_test)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

```
0.4996882410513651
```

From confusion matrix, accuracy, F1 score and area under the curve can see, this model isn't a qualified model, it is not precise and accurate. Recall is quiet low due to unbalanced sample, so accuracy also can not represent anything meaningful.

6. Model Optimization

Model Optimization

```
[15]: #data re-sampling: over-sampling
from imblearn.over_sampling import SMOTE
```

```
[163]: #seperate features and labels
# 1:inlcuding all variables as features
columns=Tmall.columns
features_columns=columns.drop('flag')
features=Tmall[features_columns]
labels=Tmall['flag']
```

```
[130]: # 2:re-select variables group as features
features_columns=[u'coupon_used_in_last_month',u'job_retired',
                  u'returned_yes', u'loan_yes']
features=Tmall[features_columns]
labels=Tmall['flag']
```

[131]:
```
features_train, features_test, labels_train, labels_test = train_test_split(features,
                                                                             labels,
                                                                             test_size=0.3,
                                                                             random_state=100)
```

[164]:
```
# 3:re-select sample_size (under condition 1)
features_train, features_test, labels_train, labels_test = train_test_split(features,
                                                                             labels,
                                                                             test_size=0.5,
                                                                             random_state=100)
```

[165]:
```
oversampler=SMOTE(random_state=100)
o_features_train,o_labels_train=oversampler.fit_sample(features_train,labels_train)
```

n [166]:
```
o_features_train.shape,o_labels_train.shape
```

Out[166]:
```
((22388L, 19L), (22388L,))
```

n [167]:
```
lr=linear_model.LogisticRegression()
lr.fit(o_features_train,o_labels_train)
```

Out[167]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Compare the results

In [128]:
```
# 1
lr.intercept_
```

Out[128]:
```
array([0.47427662])
```

[129]:
```
# 1
lr.coef_
```

ut[129]:
```
array([[-0.00495529, -0.13832846,  0.39640771,  0.29869164,  0.03019951,
        -0.17385206, -0.34929796,  0.5141064 ,  1.02069377,  0.13707739,
        -0.03716222,  0.74092394,  0.28677175,  0.46168682,  0.08111704,
         0.2332854 , -0.89016293, -0.93862611, -0.71039494]])
```

In [135]:
```
# 2
lr.intercept_
```

Out[135]:
```
array([0.25505115])
```

[136]:
```
# 2
lr.coef_
```

ut[136]:
```
array([[ 0.45724557,  0.59153382, -0.86599627, -0.53094198]])
```

In [168]:
```
# 3
lr.intercept_
```

Out[168]:
```
array([0.49306065])
```

[169]:
```
# 3
lr.coef_
```

ut[169]:
```
array([[-0.00352738, -0.14514534,  0.40248386,  0.22751968, -0.03975685,
        -0.0812331 , -0.19693163,  0.43130432,  0.94557428, -0.38990977,
        -0.07374383,  0.62084024,  0.15988295,  0.2863408 ,  0.08658016,
         0.25748121, -1.47265425, -0.93080918, -0.5754277 ]])
```

In [170]:
```
labels_pred_train=lr.predict(o_features_train)
labels_pred_test=lr.predict(features_test)
```

In [34]:
```
# 1
metrics.confusion_matrix(o_labels_train, labels_pred_train)
```
Out[34]:
```
array([[10273,  5350],
       [ 5280, 10343]], dtype=int64)
```
In [35]:
```
# 1
metrics.confusion_matrix(labels_test, labels_pred_test)
```
Out[35]:
```
array([[4501, 2232],
       [ 310,  553]], dtype=int64)
```
[139]:
```
# 2
metrics.confusion_matrix(o_labels_train, labels_pred_train)
```
ut[139]:
```
array([[ 9306,  6317],
       [ 5360, 10263]], dtype=int64)
```
[140]:
```
# 2
metrics.confusion_matrix(labels_test, labels_pred_test)
```
ut[140]:
```
array([[4095, 2638],
       [ 301,  562]], dtype=int64)
```
[172]:
```
# 3
metrics.confusion_matrix(o_labels_train, labels_pred_train)
```
t[172]:
```
array([[7320, 3874],
       [3845, 7349]], dtype=int64)
```
[173]:
```
# 3
metrics.confusion_matrix(labels_test, labels_pred_test)
```
t[173]:
```
array([[7355, 3807],
       [ 534,  963]], dtype=int64)
```
n [36]:
```
# 1
metrics.accuracy_score(o_labels_train, labels_pred_train)
```
Out[36]: 0.6597964539461051

n [37]:
```
# 1
metrics.accuracy_score(labels_test, labels_pred_test)
```
Out[37]: 0.6653501843075302

[141]:
```
# 2
metrics.accuracy_score(o_labels_train, labels_pred_train)
```
ut[141]: 0.6262881648851053

[142]:
```
# 2
metrics.accuracy_score(labels_test, labels_pred_test)
```
ut[142]: 0.6130858346498157

[174]:
```
# 3
metrics.accuracy_score(o_labels_train, labels_pred_train)
```
ut[174]: 0.6552170805788815

[175]:
```
# 3
metrics.accuracy_score(labels_test, labels_pred_test)
```
ut[175]: 0.6570819180030019

In [38]:
```
# 1
print(classification_report(o_labels_train, labels_pred_train))
```
```
             precision    recall  f1-score   support

          0       0.66      0.66      0.66     15623
          1       0.66      0.66      0.66     15623

avg / total       0.66      0.66      0.66     31246
```

In [39]:
```
# 1
print(classification_report(labels_test, labels_pred_test))
```
```
             precision    recall  f1-score   support

          0       0.94      0.67      0.78      6733
          1       0.20      0.64      0.30       863

avg / total       0.85      0.67      0.73      7596
```

[143]:
```
# 2
print(classification_report(o_labels_train, labels_pred_train))
```
```
             precision    recall  f1-score   support

          0       0.63      0.60      0.61     15623
          1       0.62      0.66      0.64     15623

avg / total       0.63      0.63      0.63     31246
```

[144]:
```
# 2
print(classification_report(labels_test, labels_pred_test))
```
```
             precision    recall  f1-score   support

          0       0.93      0.61      0.74      6733
          1       0.18      0.65      0.28       863

avg / total       0.85      0.61      0.68      7596
```

[176]:
```
# 3
print(classification_report(o_labels_train, labels_pred_train))
```
```
             precision    recall  f1-score   support

          0       0.66      0.65      0.65     11194
          1       0.65      0.66      0.66     11194

avg / total       0.66      0.66      0.66     22388
```

[177]:
```
# 3
print(classification_report(labels_test, labels_pred_test))
```
```
             precision    recall  f1-score   support

          0       0.93      0.66      0.77     11162
          1       0.20      0.64      0.31      1497

avg / total       0.85      0.66      0.72     12659
```

In [40]:
```
# 1
fpr, tpr, threshold=roc_curve(o_labels_train, labels_pred_train)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

0.6597964539461051

In [41]:
```
# 1
fpr, tpr, threshold=roc_curve(labels_test, labels_pred_test)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

0.6546431947659606

[145]:
```
# 2
fpr, tpr, threshold=roc_curve(o_labels_train, labels_pred_train)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

0.6262881648851053

[146]:
```
# 2
fpr, tpr, threshold=roc_curve(labels_test, labels_pred_test)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

0.6297075558218898

[178]:
```
# 3
fpr, tpr, threshold=roc_curve(o_labels_train, labels_pred_train)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

0.6552170805788814

[179]:
```
# 3
fpr, tpr, threshold=roc_curve(labels_test, labels_pred_test)
roc_auc=auc(fpr, tpr)
print(roc_auc)
```

0.6511093320847033

After data re-sampling did twice adjustment, there's no large difference among these three models according to kinds of score method, but the model gets better compared to previous, so select the #1 as ideal one so far, that's including all variables and test size is 0.3.

[181]:
```
data=np.array([[-0.00495529, -0.13832846,  0.39640771,  0.29869164,  0.03019951,
               -0.17385206, -0.34929796,  0.5141064 ,  1.02069377,  0.13707739,
               -0.03716222,  0.74092394,  0.28677175,  0.46168682,  0.08111704,
                0.2332854 , -0.89016293, -0.93862611, -0.71039494]])
```

[182]:
```
data=pd.DataFrame(data)
```

[183]:
```
data.columns=features_columns
```

lr.coef_:

```
[188]: data.stack().sort_values(ascending=False)
```

```
t[188]: 0  job_retired                  1.020694
           job_student                  0.740924
           job_management               0.514106
           job_unemployed               0.461687
           coupon_used_in_last_month    0.396408
           job_admin.                   0.298692
           job_technician               0.286772
           marital_single               0.233285
           job_self-employed            0.137077
           marital_married              0.081117
           job_blue-collar              0.030200
           age                         -0.004955
           job_services                -0.037162
           coupon_used_in_last6_month  -0.138328
           job_entrepreneur            -0.173852
           job_housemaid               -0.349298
           loan_yes                    -0.710395
           default_yes                 -0.890163
           returned_yes                -0.938626
        dtype: float64
```

lr.intercept_ array([0.47427662]),

Coefficient analysis: job_retired impacts more among the all features with positive correlation, the possibility of retired users would use coupon is e^1.02=2.7 times larger than the users do not retire; return_yes has the largest effect on using coupon or not among the features with negative correlation, people returned before, the possibility of using coupon is e^(-0.93)=0.39 times that people never returned.。

```
Assessment:
confusion_matrix_test: array[[4501, 2232],
                             [ 310,  553]],
accuracy_score_test: 0.6653501843075302
classification_report_test:
```

	precision	recall	f1-score	support
0	0.94	0.67	0.78	6733
1	0.20	0.64	0.30	863
avg / total	0.85	0.67	0.73	7596

```
roc_auc_test: 0.6546431947659606
```

Conclusion:
1. When model assessment, should select the proper method to score based on different kinds of business, or comprehensive perspectives;
2. When encountering sample imbalance, select appropriate methods to adjust so as to ensure the accuracy of the model;
3. Try multiple times for model optimization properly and continually until reach ideal.