Instacart Market Basket Analysis

——Data Mining Individual Project Report

Prepared by

Qing Ruan

November 26, 2019

Table of Contents

1. Introduction 3
2. Individual Work Description 3
3. Portion of the Individual Work Description 4
3.1 Coding 4
3.1.1 Exploration of data: `order` part 4
3.2 Preprocessing 6
3.2.1 Features creation 6
3.2.2 Train/test dataset creation 12
3.2.3 Model building and evaluation: Logistic Regression 13
3.3 Documents for group report 15
3.3.1 EDA for order information 15
3.3.2 Features Creation 19
3.3.3 Train/Test dataset Creation 22
3.3.4 One model building: Logistic Regression 23
4. Results 26
4.1 Logistic regression model 26
4.2 K-Nearest-Neighbor model 27
4.3 Random Forest Model 28
5. Summary and Conclusions 30
6. Percentage of the code referenced 31
7. References 31

1. Introduction

Instacart released a dataset, "The Instacart Online Grocery Shopping Dataset 2017", which contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, they provided their orders and the sequence of products in each order. They also provided the time including the week and the hour of day the order was placed, and the relative time between orders. Using this data to test models, Instacart will be able to predict products that a user will buy again. Our project using this dataset from Kaggle aims to predict which previously purchased products would be in a consumer's next order by constructing various classification models.

The structure of this report is described as follow. The 2nd section shows an overview of the dataset and exploration of data. Section 3 is the part of data processing such as features creation. Section 4 shows models building and evaluation. These models include Logistic Regression, Naive Bayes and Random Forest. We will discuss all results of our experiments in more detail. The last part in section 5 presents the summary for our results and a potential improvement to the project.

Tasks for this project can be divided into two parts. One is for coding including EDA, data preprocessing and model building, and the other one is document, including slides making, group report and GUI.

Huang is in charge of coding for most part of EDA, one model building, documents of introduction and GUI for this project. I am in charge of coding for remaining part of EDA, features creation and one model building, and writing most part of group report. Ya is in charge of one model building, remaining part of group report and slides making.

2. Individual Work Description

My work for the project can be divide into two major parts as below.

- Coding:
  - exploration of data: `order` part
  - preprocessing: creating 14 features; trainset creation
  - one model building: logistic regression
- Document:

- group report: Introduction, Exploration of data, data preprocessing and part of models building

3. Portion of the Individual Work Description

Now I will describe my portion of the project in more detail.

3.1 Coding

3.1.1 Exploration of data: `order` part

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
os.getcwd()
os.chdir('/Users/qingruan/Desktop/DM_project')
aisles = pd.read_csv('instacart-market-basket-analysis/aisles.csv')
departments = pd.read_csv('instacart-market-basket-analysis/departments.csv')
train = pd.read_csv('instacart-market-basket-analysis/order_products__train.csv')
prior = pd.read_csv('instacart-market-basket-analysis/order_products__prior.csv')
orders = pd.read_csv('instacart-market-basket-analysis/orders.csv')
products = pd.read_csv('instacart-market-basket-analysis/products.csv')
sample = pd.read_csv('instacart-market-basket-analysis/sample_submission.csv')
print(orders.shape)
print(orders.columns)
print(orders.dtypes)
orders.count()

orders=pd.read_csv('instacart-market-basket-analysis/orders.csv')
# In[12]:
plt.figure(figsize=(12,8))
color = sns.color_palette()
dist_eval_set=orders.eval_set.value_counts()
sns.barplot(dist_eval_set.index,dist_eval_set.values, alpha=0.8,color=color[1])
plt.ylabel('orders')
plt.xlabel('eval_set type')
plt.title('Number of orders in each set')

plt.show()
# In[14]:
color = sns.color_palette()
group_ev=orders.groupby('eval_set')
x=[]
y=[]
for name,group in group_ev:
    x.append(name)
    y.append(group.user_id.unique().shape[0])
sns.barplot(x,y, alpha=0.8,color=color[2])
plt.ylabel('users')
plt.xlabel('eval_set type')
plt.title('Number of orders in each set')
```

```
47.
48. plt.show()
49. # In[15]:
50. group_ev=orders.groupby('eval_set')
51.
52. for name,group in group_ev:
53.     print(name)
54.     print(group.user_id.unique().shape[0])
55. # In[16]:
56. dist_no_orders=orders.groupby('user_id').order_number.max()
57. dist_no_orders=dist_no_orders.value_counts()
58.
59. plt.figure(figsize=(20,8))
60. sns.barplot(dist_no_orders.index,dist_no_orders.values)
61. plt.xlabel('orders')
62. plt.ylabel('users')
63. plt.title('Frequency of orders by users')
64. plt.show()
65. # In[17]:
66. dist_d_orders=orders.order_dow.value_counts()
67.
68. sns.barplot(dist_d_orders.index,dist_d_orders.values, palette=sns.color_palette('Blues_d',
    7))
69. plt.xlabel('day of week')
70. plt.ylabel('orders')
71. plt.title('Frequency of orders by day of week')
72.
73. plt.show()
74. # In[19]:
75. dist_h_orders=orders.order_hour_of_day.value_counts()
76. plt.figure(figsize=(10,8))
77. sns.barplot(dist_h_orders.index,dist_h_orders.values, palette=sns.color_palette('Greens_d'
    ,24))
78. plt.xlabel('hour of day')
79. plt.ylabel('orders')
80. plt.title('Frequency of orders by hour of day')
81. plt.show()
82. # In[18]:
83. grouped=orders.groupby(['order_dow','order_hour_of_day']).order_number.count().reset_index
    ()
84. # using reset_index to set order_dow and ordr_h_day as columns, or they would be index
85. time_orders=grouped.pivot('order_dow', 'order_hour_of_day','order_number')
86.
87. plt.figure(figsize=(10,5))
88. sns.heatmap(time_orders, cmap='YlOrRd')
89. plt.ylabel('Day of Week')
90. plt.xlabel('Hour of Day')
91. plt.title('Number of Orders Day of Week vs Hour of Day')
92. plt.show()
93. # In[20]:
94. dist_d_prior_orders=orders.days_since_prior_order.value_counts()
95.
96. plt.figure(figsize=(12,8))
97. sns.barplot(dist_d_prior_orders.index,dist_d_prior_orders.values, palette=sns.color_palett
    e('Greens_d',31))
98. plt.xlabel('days of prior order')
99. plt.ylabel('count')
100.    plt.title('Time interval between orders')
101.
102.    plt.show()
```

3.2 Preprocessing

3.2.1 Features creation

```
# # preprocessing

# ## check order_id for train/prior set and orders set

# In[3]:


# order_id for train
print(order_products_train.order_id.value_counts().shape[0])
print(orders.loc[orders.eval_set=='train'].shape[0])
# order_id for prior
print(order_products_prior.order_id.value_counts().shape[0])
print(orders.loc[orders.eval_set=='prior'].shape[0])


# From above, we can find that the order_id for both train & prior dataset and orders set is correspondont one by one, which is consistent with data description in the kaggle website overview.

# ## missing value

# In[4]:


print(orders.isnull().sum())
orders.index


# There are 206209 missing values in days_since_piror_orders, they means these orders are first order for the user, respectively. According to this meaning, we replace NaN with 0.


# In[5]:


orders=orders.fillna(0)
orders.isnull().sum()


# ## subsetting and merging for datasets

# In[6]:


#merge order_products_prior and orders
merge_oop=pd.merge(orders, order_products_prior, on='order_id',how='right') #equals to select evalset==0
merge_oop.head()


# In[7]:


#check merge results
print(order_products_prior.loc[order_products_prior['order_id']==2539329])
```

```
print(merge_oop.loc[merge_oop['order_id']==2539329])


# As for training, all of the data we used to analyze should come from prior relevant info
rmation excluding the last order information for each user, for the next feature selection
, we only use 'merge_oop' we created just now instead of orders and order_products_prior.

# Smillarly, 'order' dataset includes prior orders and last order used for training label.
 Therefore, we need to exclude information about last order for each user before we using
this dataset.
# In[8]:


orders_prior=orders.loc[orders.eval_set=='prior']
orders_prior.head()


# # 2.feature selection

# ## 1.user feature

# ### 1.1 number of orders for each user

# In[9]:


user_f_1=merge_oop.groupby('user_id').order_number.max().reset_index(name='n_orders_users'
)
user_f_1.head()


# ### 1.2 number of products for each user

# In[10]:


user_f_2=merge_oop.groupby('user_id').product_id.count().reset_index(name='n_products_user
s')
user_f_2.head()


# ### 1.3 average products of order for each user

# In[11]:


user_f_2['avg_products_users']=user_f_2.n_products_users/user_f_1.n_orders_users
user_f_2.head()


# ### 1.4 day of one week orderd most for each user

# In[12]:


    temp=merge_oop.groupby('user_id')['order_dow'].value_counts().reset_index(name='times_d
')
    temp.head()

```

```
104.
105.    # In[13]:
106.
107.
108.    # find most days ordered by each user
109.    user_f_3=temp.loc[temp.groupby('user_id')['times_d'].idxmax(),['user_id','order_dow']]

110.    user_f_3=user_f_3.rename(columns={'order_dow':'dow_most_user'})
111.    user_f_3.head()
112.
113.
114.
115.    # ### 1.5 time of one day ordered most for each user
116.
117.    # In[14]:
118.
119.
120.    temp=merge_oop.groupby('user_id')['order_hour_of_day'].value_counts().reset_index(name=
    'times_h')
121.    temp.head()
122.
123.
124.    # In[15]:
125.
126.
127.    # find most hours in one day ordered by each user
128.    user_f_4=temp.loc[temp.groupby('user_id')['times_h'].idxmax(),['user_id','order_hour_of
    _day']]
129.    user_f_4=user_f_4.rename(columns={'order_hour_of_day':'hod_most_user'})
130.    user_f_4.head()
131.
132.
133.    # ### 1.6 reorder_ratio for each user
134.
135.    # In[16]:
136.
137.
138.    user_f_5=merge_oop.groupby('user_id').reordered.mean().reset_index(name='reorder_ratio_
    user')
139.    user_f_5.head()
140.
141.
142.    # ### 1.7 shopping frequency for each user
143.
144.    # In[17]:
145.
146.
147.    order_user_group=orders_prior.groupby('user_id')
148.    user_f_6=(order_user_group.days_since_prior_order.sum()/order_user_group.days_since_pri
    or_order.count()).reset_index(name='shopping_freq')
149.    user_f_6.head()
150.
151.
152.    # ### user feature list
153.
154.    # In[18]:
155.
156.
157.    user=pd.merge(user_f_1,user_f_2,on='user_id')
158.    user=user.merge(user_f_3,on='user_id')
159.    user=user.merge(user_f_4,on='user_id')
```

```
160.    user=user.merge(user_f_5,on='user_id')
161.    user=user.merge(user_f_6,on='user_id')
162.    user.head()
163.
164.
165.    # ## 2.product feature
166.
167.    # ### 2.1 times of ordered for each product
168.
169.    # In[19]:
170.
171.
172.    prod_f_1=order_products_prior.groupby('product_id').order_id.count().reset_index(name='
    times_bought_prod')
173.    prod_f_1.head()
174.
175.
176.    # ### 2.2 reordered ratio for each product
177.
178.    # In[20]:
179.
180.
181.    prod_f_2=order_products_prior.groupby('product_id').reordered.mean().reset_index(name='
    reorder_ratio_prod')
182.    prod_f_2.head()
183.
184.
185.    # ### 2.3 average postions of cart for each product
186.
187.    # In[21]:
188.
189.
190.    prod_f_3=order_products_prior.groupby('product_id').add_to_cart_order.mean().reset_inde
    x(name='position_cart_prod')
191.    prod_f_3.head()
192.
193.
194.    # ### 2.4 reordered ratio for each department
195.
196.    # In[22]:
197.
198.
199.    prod_dep=pd.merge(department['department_id'],product[['department_id','product_id']],o
    n='department_id',how='right')
200.    totall_info=pd.merge(prod_dep,prod_f_2,on='product_id',how='right')
201.    totall_info.head()
202.
203.
204.    # In[23]:
205.
206.
207.    group=totall_info.groupby('department_id')
208.    prod_f_4=group.reorder_ratio_prod.mean().reset_index(name='reorder_ratio_dept')
209.
210.    prod_f_4=pd.merge(prod_f_4,totall_info,on='department_id')
211.    del prod_f_4['reorder_ratio_prod']
212.    prod_f_4.drop(['department_id'],axis=1,inplace=True)
213.    prod_f_4.head()
214.
215.
216.    # ### product feature list
```

```
217.
218.    # In[24]:
219.
220.
221.    prod=pd.merge(prod_f_1,prod_f_2,on='product_id')
222.    prod=prod.merge(prod_f_3,on='product_id')
223.    prod=prod.merge(prod_f_4,on='product_id')
224.    prod.head()
225.
226.
227.    # ## 3. user & product feature
228.
229.    # ### 3.1 times of one product bought by one user
230.
231.    # In[25]:
232.
233.
234.    user_prd_f_1=merge_oop.groupby(['user_id','product_id']).order_id.count().reset_index(n
    ame='times_bought_up')
235.    user_prd_f_1.head()
236.
237.
238.    # ### 3.2 reordered ratio of one product bought by one user
239.
240.    # In[26]:
241.
242.
243.    # number of orders for one user
244.    user_prd_f_2=merge_oop.groupby('user_id').order_number.max().reset_index(name='n_orders
    _users')
245.    # when the user bought the product for the first time
246.    temp=merge_oop.groupby(['user_id','product_id']).order_number.min().reset_index(name='f
    irst_bought_number')
247.    # merge two datasets
248.    user_prd_f_2=pd.merge(user_prd_f_2,temp,on='user_id')
249.    # how many orders performed after the user bought the product for the first time
250.    user_prd_f_2['order_range']=user_prd_f_2['n_orders_users']-
    user_prd_f_2['first_bought_number']+1
251.    #reordered ratio
252.    user_prd_f_2['reorder_ratio_up']=user_prd_f_1.times_bought_up/user_prd_f_2.order_range

253.    user_prd_f_2.head()
254.
255.
256.    # In[27]:
257.
258.
259.    user_prd_f_2=user_prd_f_2.loc[:,['user_id','product_id','reorder_ratio_up']]
260.    user_prd_f_2.head()
261.
262.
263.    # ### 3.3 ratio of one product bought in one user's last four orders
264.
265.    # In[28]:
266.
267.
268.    #Reversing the order number for each product.
269.    merge_oop['order_number_back']=merge_oop.groupby('user_id')['order_number'].transform(m
    ax)-merge_oop['order_number']+1
270.    merge_oop.head()
271.
```

```
272.
273.    # In[29]:
274.
275.
276.    # select orders where order_number_back <=4
277.    temp1=merge_oop.loc[merge_oop['order_number_back']<=4]
278.    temp1.head()
279.
280.
281.    # In[30]:
282.
283.
284.    # create feature
285.    user_prd_f_3=(temp1.groupby(['user_id','product_id'])['order_number_back'].count()/4).r
    eset_index(name='ratio_last4_orders_up')
286.    user_prd_f_3.head()
287.
288.
289.    # ### user & product feature list
290.
291.    # In[31]:
292.
293.
294.    # merge three features for the user&product list
295.    user_prd=pd.merge(user_prd_f_1,user_prd_f_2,on=['user_id','product_id'])
296.    user_prd=user_prd.merge(user_prd_f_3,on=['user_id','product_id'],how='left')
297.    user_prd.head()
298.
299.
300.    # After checking, we notice that some rows for raio_last4_orders_up have NaN values for
     our new feature. This happens as there might be products that the customer did not buy on
     its last four orders. For these cases, we turn NaN values into 0.
301.
302.    # In[32]:
303.
304.
305.    user_prd.ratio_last4_orders_up.fillna(0,inplace=True)
306.
307.
308.    # ## total features list
309.
310.    # In[33]:
311.
312.
313.    total_info=pd.merge(user_prd, user,on='user_id',how='left')
314.    total_info=total_info.merge(prod,on='product_id',how='left')
315.    total_info.head()
316.
317.
318.    # In[34]:
319.
320.
321.    total_info.info()
322.
323.
324.    # In[35]:
325.
326.
327.    total_info.head()
328.
329.
```

```
330.    # Totally, we select 14 features to do our models futher. They are'n_orders_users', 'n_
    products_users', 'avg_products_users',
331.    #         'dow_most_user','hod_most_user','reorder_ratio_user', 'shopping_freq',
332.    #         'product_id', 'times_bought_up', 'reorder_ratio_up',
333.    #         'times_last4_orders_up', 'times_bought_prod', 'reorder_ratio_prod',
334.    #         'position_cart_prod', 'reorder_ratio_dept'.
335.
336.    # In[36]:
337.
338.
339.    total_info.isnull().sum()
340.
341.
342.    # After checking, there is no missing value in this dataset. Then we will use it as obs
    ervations to build our model.
343.
344.    # # 3.creating train and test dataset
345.
346.    # ### set reordered as independent columns
347.
348.    # In[37]:
349.
350.
351.    #select order_id for train by orders.csv
352.    orders_y=orders.loc[((orders.eval_set=='train') | (orders.eval_set=='test')),['user_id'
    ,'order_id','eval_set']]
353.    orders_y.head()
354.
355.
356.    # In[38]:
357.
358.
359.    # add order_id to the total_info
360.    total_info=pd.merge(total_info,orders_y,on='user_id',how='left')
361.    total_info.head()
```

3.2.2 Train/test dataset creation

```
1.  # ## create train dataset
2.  # In[39]:
3.  # select reordered in order_products_train as our dependent variable
4.  total_info_train=total_info[total_info.eval_set=='train']
5.  total_info_train=pd.merge(total_info_train,order_products_train[['product_id','order_id','
    reordered']],on=['order_id','product_id'],how='left')
6.  total_info_train.head()
7.  #total_info=total_info.drop(['reordered_x','reordered_y','add_to_cart_order'],axis=1)
8.  # We add 'reordered' to total_info, and this column in original order_products_train.csv m
    eans for the train order(last order for each user), if the product has been bought before.
    After merged, it means if the product of certain user has been bought before. dataset and
    the same t. Therefore, only reorder==1 columns have been selected and reorder==Nan means t
    he product has not been selected in the user last order.
9.
10. # In[40]:
11. # fill Nan with 0
12. total_info_train['reordered'].fillna(0,inplace=True) ## inplace decides whether modify ori
    ginal dataset
13.
14.
15. # In[41]:
```

```
# drop order_id
total_info_train.drop(['order_id','eval_set'],axis=1,inplace=True)
total_info_train.head()


# In[42]:
# reset 'user_id' and 'product_id'index
total_info_train=total_info_train.set_index(['user_id','product_id'])
total_info_train.head()


# ### create test dataset

# In[43]:
# select test part
total_info_test=total_info[total_info.eval_set=='test']
total_info_test.head()


# In[44]:
# reset 'user_id' and 'product_id'index
total_info_test=total_info_test.set_index(['user_id','product_id'])
total_info_test.head()


# In[45]:
# only select features
total_info_test.drop(['order_id','eval_set'],axis=1,inplace=True)
total_info_test.head()


# In[46]:
total_info_train.shape,total_info_test.shape
```

3.2.3 Model building and evaluation: Logistic Regression

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score


# In[48]:


X=total_info_train.drop('reordered',axis=1)
y=total_info_train.reordered
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,stratify=y)


# In[49]:


y.value_counts()

```

```
# In[50]:


sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)


# In[51]:


clf = LogisticRegression()
clf.fit(X_train, y_train)


# In[52]:


clf.coef_


# In[53]:


# make predictions# predicton on test
y_pred = clf.predict(X_test)
y_pred_score = clf.predict_proba(X_test)


# In[54]:


print("\n")

print("Classification Report: ")
print(classification_report(y_test,y_pred))
print("\n")


print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
print("\n")

print("ROC_AUC : ", roc_auc_score(y_test,y_pred_score[:,1]) * 100)
print("\n")


# In[66]:


conf_matrix = confusion_matrix(y_test, y_pred)

df_cm = pd.DataFrame(conf_matrix, index=['0','1'], columns=['0','1'] )

plt.figure(figsize=(5,5))
hm = sns.heatmap(df_cm, cbar=False, annot=True, square=True, fmt='d', annot_kws={'size': 2
0}, yticklabels=df_cm.columns, xticklabels=df_cm.columns)
hm.yaxis.set_ticklabels(hm.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=20)
hm.xaxis.set_ticklabels(hm.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=20)
plt.ylabel('True label',fontsize=20)
plt.xlabel('Predicted label',fontsize=20)
```

```
84. plt.tight_layout()
85. plt.show()
```

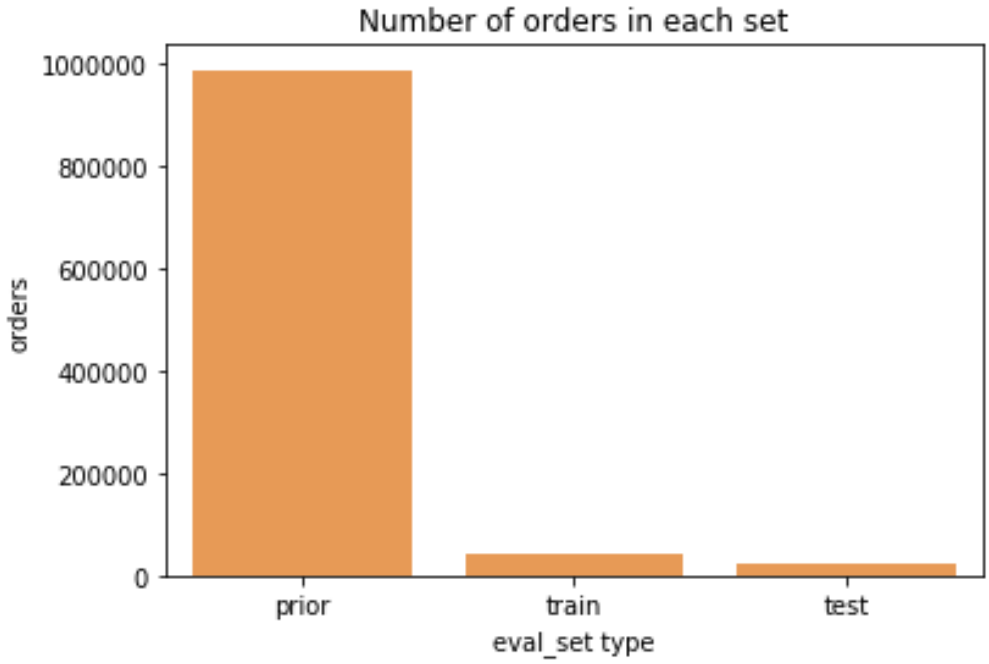
3.3 Documents for group report

3.3.1 EDA for order information

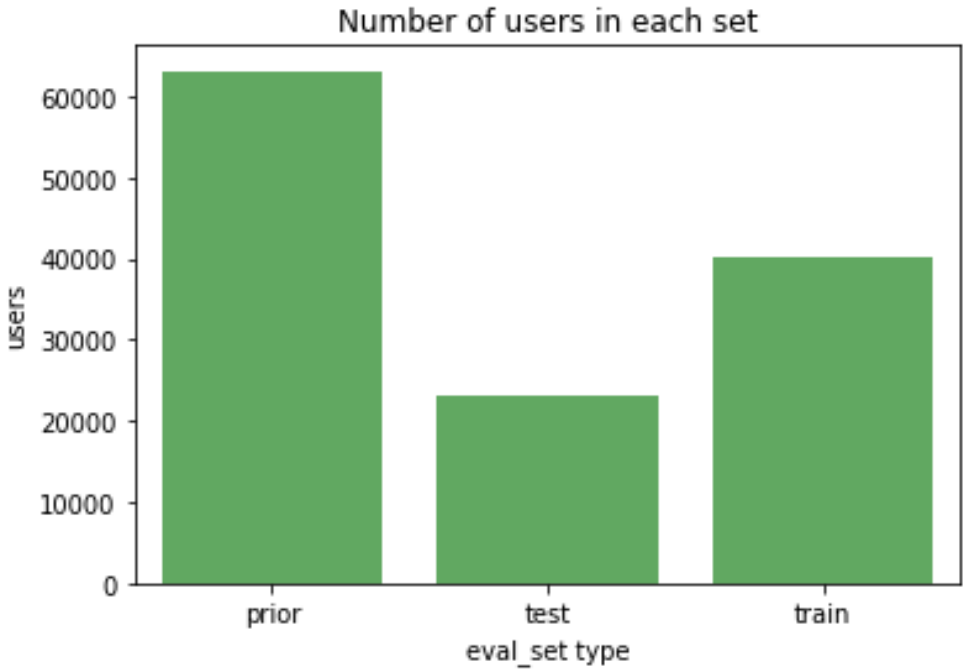
3.3.1.1 Three Sets

Firstly, we do the basic exploration for this csv. There are 3,421,083 observations totally. We check the missing values and data types for each column. The `days_since_prior_order` has missing values for the reason that the first order didn't have this attribute. Thus, we replace these Nan with zero instead later.

Now, we take a look at some characteristics about three sets.

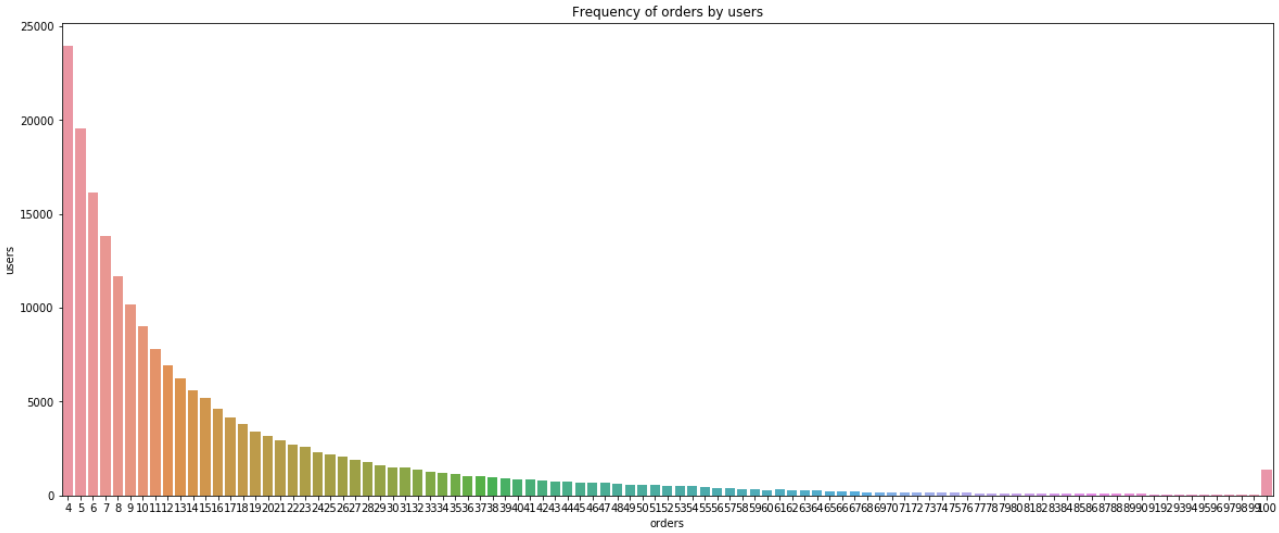


There are 985475 orders for the prior set, and the dataset extract the last order of each customer as train and test dataset, respectively. The train set has 40096 observations and the test dataset has 23004 observations.



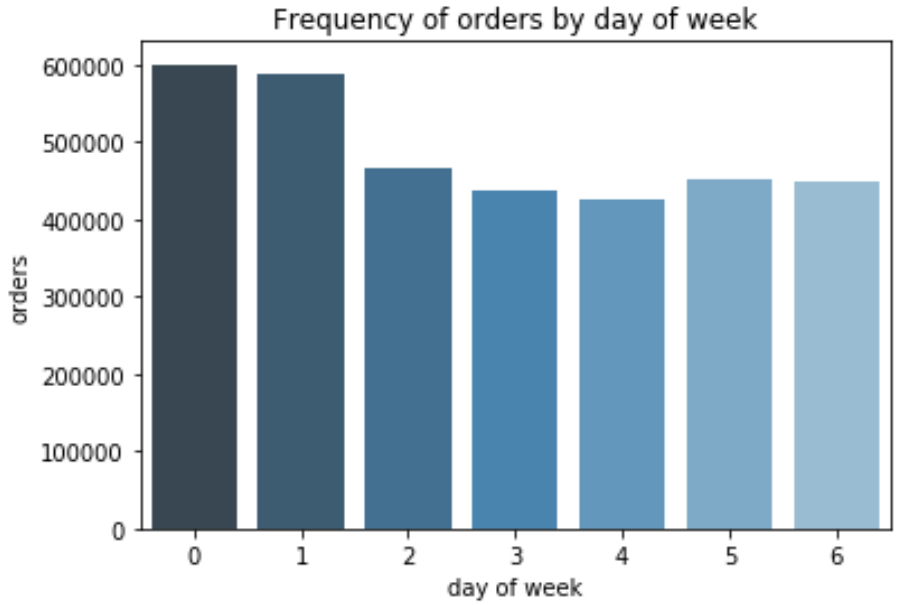
There are 63100 users in prior set, 23004 users in test set and 40096 users in train set.

3.3.1.2 Frequency of Orders

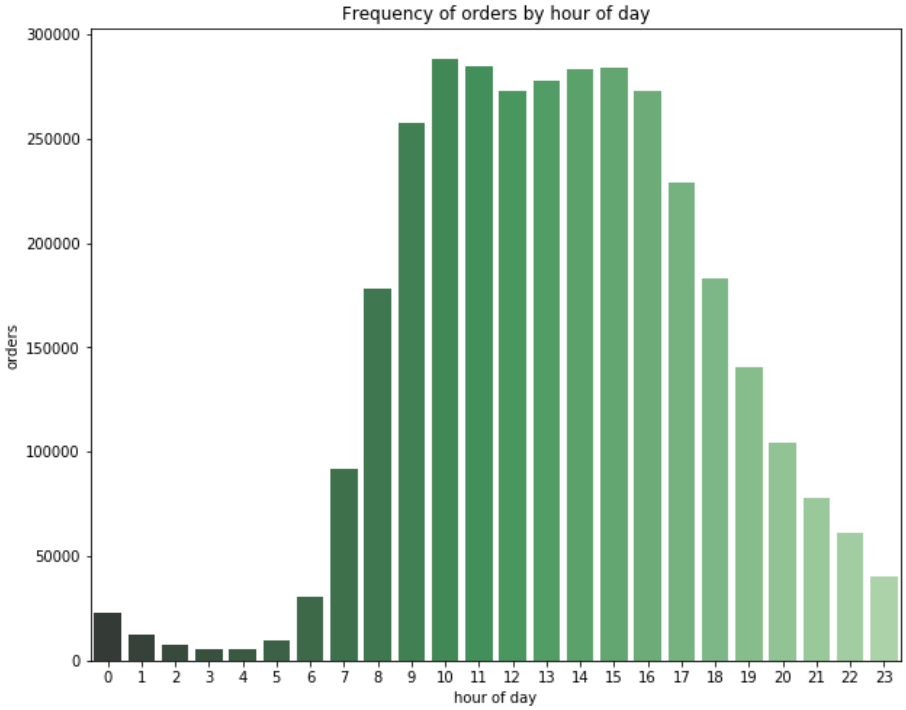


There are no orders less than 4 and is max capped at 100 as given in the data page.

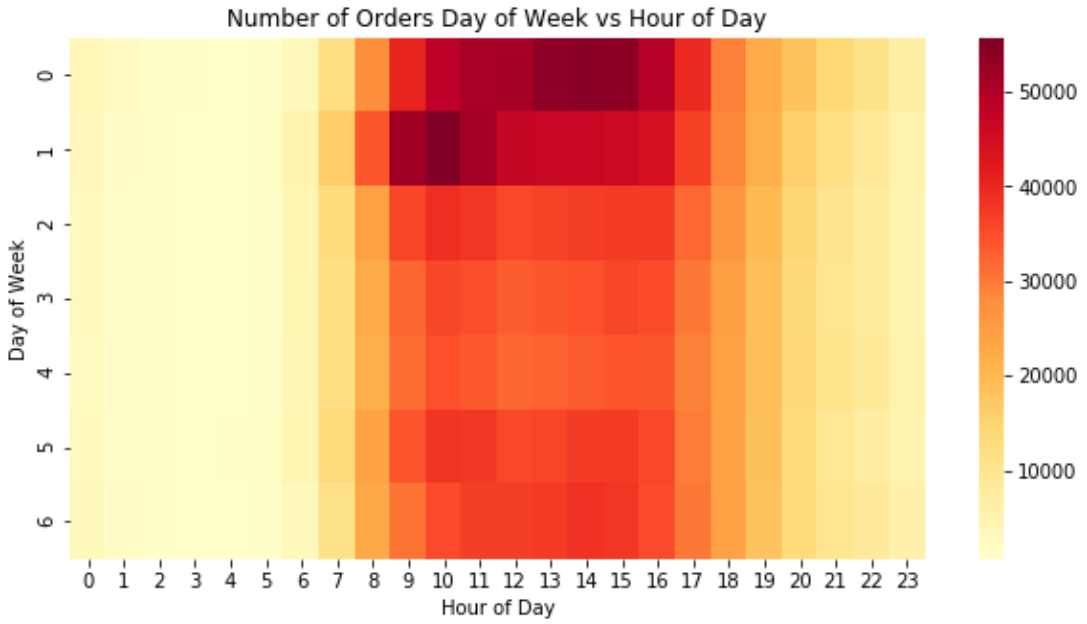
3.3.1.3 Time of Orders



It looks as though 0 represents Saturday and 1 represents Sunday. Wednesday is then the least popular day to make orders.

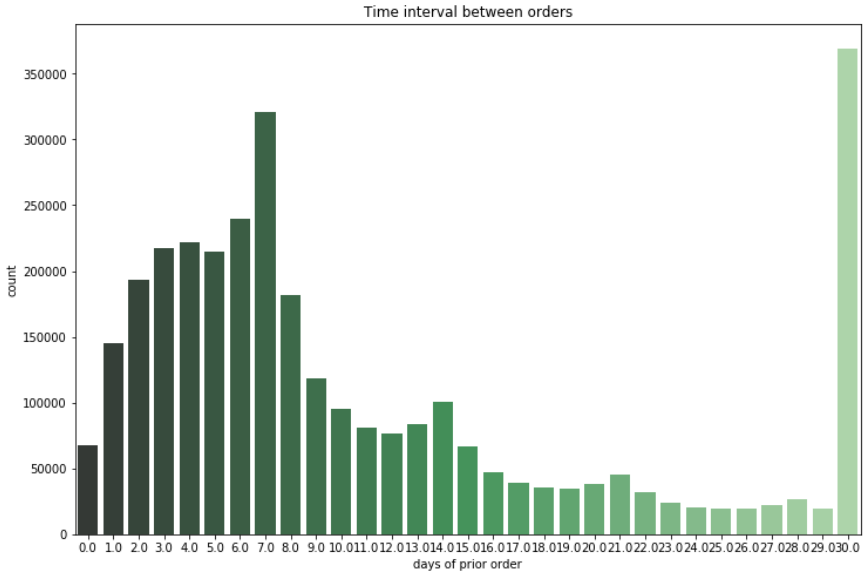


So, majority of the orders are made during daytime. The 10am hour is the most popular time to make orders, followed by a dip around lunch time and a pickup in the afternoon. Now let us combine the day of week and hour of day to see the distribution.



Saturday afternoon and Sunday morning are the most popular time to make orders.

3.3.1.4 Time Interval between Orders



While the most popular relative time between orders is monthly (30 days), there are "local maxima" at weekly (7 days), biweekly (14 days), triweekly (21 days), and quadriweekly (28 days).Therefore it appears that customers order once in every week (check the peak at 7 days) or once in a month (peak at 30 days). We could also see smaller peaks at 14, 21 and 28 days (weekly intervals).

3.3.2 Features Creation

We create a new Dataframe named `merge_oop` combining `orders` and `order_products_prior`. Here `merge_oop` only contains prior observations.

As for training, all of the data we used to analyze should come from prior relevant information excluding the last order information for each user, for the next feature selection part, we only use 'merge_oop' we created just now instead of orders and order_products_prior.

Similarly, 'order' dataset includes prior orders and last order used for training label. Therefore, we need to exclude information about last order for each user before using this dataset.

Now, we have two dataframes which are merge_oop and orders_prior.

Then we can create features using the merged dataframes we created before. There are three types of features we created as below. We will introduce them in more detail as below.

3.3.2.1 User Feature (`user`)

It describes the patterns of one user. There are seven sub features in this type. Here we use data from `merge_oop` dataframe we set before to create them.

(1) `n_orders_users`: The number of orders for each user.

Created a new dataframe named by `user_f_1`. Acquired by grouping users (`user_id`) from `merge_oop`and calculate the number of orders (`order_number`).

(2) `n_products_users`: The number of products for each user.

Create a new dataframe named by `user_f_2`. Acquired by grouping users (`user_id`) from `merge_oop` and calculate the number of products (`product_id`).

(3) `avg_products_users`: Average products of one order for each user.

Created a new column in `user_f_2`. Acquired by dividing `user_f_1` by `n_products_users` created above.

(4) `dow_most_user`: The day in one week ordered most for each user.

Created a new dataframe named by `user_f_3`. Calculate the number of orders for each day of week and select the day corresponding to the maximum count.

Numeric variable.

(5) `times_h`: The time of one day ordered most for each user.

Create a new dataframe named by `user_f_4`. Calculate the number of orders for different hour in one day from `merge_oop` dataframe and select the hour corresponding to the maximum count.

(6) `reorder_ratio_user`: Reorder ratio for each user.

Create a new dataframe named by `user_f_5`. Group `merge_oop` dataframe by 'user_id'. As the values for `reordered` columns is 0 or 1, we create reordered ratio by calculating the mean of it for each group.

(7) `shopping_freq`: Shopping frequency for each user.

Create a new dataframe named by `user_f_6`. Group `merge_oop` dataframe by 'user_id' and calculate the mean of intervals between orders (`days_since_prior_order`) for each user.

Now, joining on common `user_id` column, we merge all `user_f_*` (* from 1 to 6) into one dataframe and name it with `user` just like below.

	user_id	n_orders_users	n_products_users	avg_products_users	dow_most_user	hod_most_user	reorder_ratio_user	shopping_freq
0	1	10	59	5.900000	4	7	0.694915	17.600000
1	2	14	195	13.928571	2	9	0.476923	14.142857
2	3	12	88	7.333333	0	16	0.625000	11.083333
3	4	5	18	3.600000	4	15	0.055556	11.000000
4	5	4	37	9.250000	3	18	0.378378	10.000000

3.3.2.2 Product Feature

It describes the characteristics of one product. There are four sub features in this type. Here we use data from `order_products_prior` dataframe we set before to create them.

(1) `times_bought_prod`: Times of ordered for each product.

Create a new dataframe named with `prod_f_1`. Acquired by grouping product (``product_id) and calculate the count of orders (`order_id`).

(2) `reorder_ratio_prod`: Reordered ratio for each product.

Create a new dataframe named with `prod_f_2`. Acquired by grouping product (``product_id) and calculate the mean of reordered.

(3) `position_cart_prod`: Average positions of cart for each product.

Create a new dataframe named with `prod_f_3`. Acquired by grouping product (``product_id) and calculate the mean of cart-position (`add_to_order`) for each group.

(4) `reorder_ratio_prod`: Reorder ratio for each department.

Create a new dataframe named with `prod_f_4`. Merge `department`, `product` and `prod_f_3` with `product_id`, group this dataframe by departments and calculate the mean of `reorder_ratio_prod` in each group.

Now, joining on common `product_id` column, we merge all `prod_f_*` (* from 1 to 4) into one dataframe and name it with `prod` just like below.

	product_id	times_bought_prod	reorder_ratio_prod	position_cart_prod	department_id	reorder_ratio_dept
0	1	1852	0.613391	5.801836	19	0.438319
1	2	90	0.133333	9.888889	13	0.242846
2	3	277	0.732852	6.415162	7	0.471714
3	4	329	0.446809	9.507599	1	0.418642
4	5	15	0.600000	6.466667	13	0.242846

3.3.2.3 User & Product Feature

It describes the pattern of one user for specific product. There are three sub features in this type. Here we use data from `merge_oop` dataframe we set before to create them.

(1)`times_bought_up`: Times of one product bought by one user.

Create a new dataframe named with `user_prd_f_1`. Acquired by grouping user (`user_id`) and product (`product_id`) and calculate the count of orders (`order_id`).

(2) `reorder_ratio_up`: Reorder ratio of one product bought by one user.

Create a new dataframe named with `user_prd_f_2`. Before calculating, group `merge_oop` dataframe by user and product. First for one product create the number of orders since the user bought it for the first time (`order_range`) by subtracting order number first bought (minimum of the `order_number`) with number of orders. Then create the times of bought for this product (`times_bought_up`) and acquire the reorder_ratio_up by dividing `order_range`.

(3) `ratio_last4_orders_up`: Ratio of one product bought in one user's last four orders.

Create a new dataframe named with `user_prd_f_3`. Reverse the order number first for each product and then create a new column `order_number_back` which selects orders where order_number_back is smaller than 5. Group `merge_oop` by user and product and acquire this feature by calculating the count of `order_number_back` dividing four. This new dataframe is named with `user_prod_f_3` including `user_id`, `product_id` and `ratio_last4_orders_up`.

Now, joining on common `user_id` and `product_id` column, we merge all `user_prod_f_*` (* from 1 to 3) into one dataframe and name it with `user_prod` just like below.

	user_id	product_id	times_bought_up	reorder_ratio_up	ratio_last4_orders_up
0	1	196	10	1.000000	1.0
1	1	10258	9	1.000000	1.0
2	1	10326	1	0.166667	NaN
3	1	12427	10	1.000000	1.0
4	1	13032	3	0.333333	0.5

After checking, we notice that some rows for raio_last4_orders_up have NaN values for our new feature. This happens as there might be products that the customer did not buy on its last four orders. For these cases, we turn NaN values into 0.

Total features can be got by merging the three types of features in dataframes above and we name the new dataframe including all of features with `total_info`.

Finally, we select 14 features to do our models further. They are `n_orders_users`,`n_products_users`,`avg_products_users`,`dow_most_user`,`hod_most_user`,`reorder_ratio_user`,`shopping_freq`,`product_id`,`times_bought_up`,`reorder_ratio_up`,`times_last4_orders_up`,`times_bought_prod`,`reorder_ratio_prod`, `position_cart_prod`, `department_id`, `reorder_ratio_dept`.

Next, we will use it as observations to build our model.

3.3.3 Train/Test dataset Creation

3.3.3.1 Set dependent Variable

We create the train/test dataset on the basis of `total_info` dataframe by adding dependent variable `reordered`.

First, by selecting eval_set equals to train, we acquire the last `order_id` information for each user. Then, we add them to the `total_info` dataframe. On the other side, we only select `product_id`,`order_id` and `reordered` columns from `order_products_train` dataframe.

Notice that, here `reordered` means if in this user's last order, this product has been bought before. Merge it with `total_info` and select merging on `total_info` by common columns `order_id` and `product_id`. After merging, only the product occurred in prior orders can be merged, therefore only reordered ==1 can be got, and for other products the user did before but not for the last order, the reorder is Nan.

And we replace Nan with zero, which means this product did not occur in this user's last order.

3.3.3.2 Reset index for each observation

As each user_id and product_id is unique which means one specific product for one certain user, we set these two elements as index.

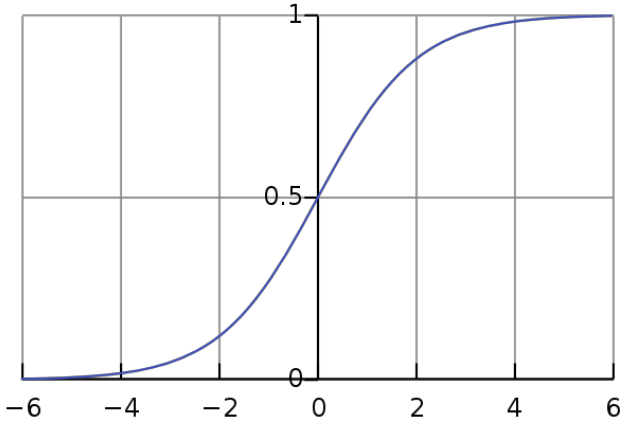
After preprocessing, we get the final dataset used to build our models named with `total_info_train`. It has 15 columns with 14 features and one dependent variable.

3.3.4 One model building: Logistic Regression

3.3.4.1 Principle and Methods

- Principle

Logistic regression is named for the function used at the core of the method, the logistic function. This function curve is shown as below. It is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.



Here is the logistic function equation:

$$f(x)=1/(1+e^{(-x)})$$

Logistic regression uses this equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights to predict an output value (y). Here the output value being modeled is a binary value (0 or 1). For example:

$$P(x)=e^{(b_0+b_1\times x)}/(1+e^{(b_0+b_1\times x)})$$

If $P(x)$ <0.5, binary value is 0, else is 1.

Here key for this algorithm is the coefficients (beta values), which can be acquired from our training data. Next, we will use this logistic regression to build our model and make predictions.

- Method

First we remove `reordered` column from total_info_train dataset as X dataset, and `reordered` alone as Y. Split train and test using sklearn package from python. Here the test size, we set 0.2, which means 80% of our observations are used as train and 20% of them are used as test. They are named with X_train and X_test.

Then we scale X_train and X_test by creating class StandardScaler and using fit function.

After train and test split and scaling X data, we build logistic regression model. All of the parameters we select default mode.

3.3.4.2 Analysis and Evaluation

The prediction results of test dataset and analysis are as below.

- Classification Report

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.99      0.95   1529168
         1.0       0.62      0.11      0.19    165765

    accuracy                           0.91   1694933
   macro avg       0.77      0.55      0.57   1694933
weighted avg       0.88      0.91      0.88   1694933


Accuracy :  90.6327860747298


ROC_AUC :  79.50233767175897
```

(1) Accuracy

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. The result shows that our model's accuracy reaches 90.488 means our model is approx. 90% accurate.

$$Accuracy = TP+TN/TP+TN+FP+FN$$

Here TP means true positives, TN means true negatives, FP means false positives and FN means false negatives.

(2) Precision

Precision means the ratio of correctly predicted positive observations (TP) to the total predicted positive observations (TP+FP).

$$Precision = TP / TP + FP$$

From the results, the precision for 0.0 is 0.91, which is a high value. It means that for the prediction that one product will not occur in user's next order is mostly correct. For the precision for 1.0 is 0.62, which means that over half for the prediction that one product will occur in user's next order is correct.

Finally, the weighted average precision score is 0.88.

(3) Recall

Recall means the ratio of correctly predicted positive observations to the all observations in actual class.

$$Recall = TP / TP + FN$$

Our objective is to predict which product will be bought in user's next order. So we focus more on prediction on 1.0. From the recall result, the value for 0.0 reaches 0.99, however, for 1.0 is 0.09.

(4) F-1 score

It means the harmonic mean of precision and recall taking both metrics into account in the following equation:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

We find that for 0.0 the score is high reaching 0.95 but for 1.0 is relatively low. The correct prediction ratio is 16%.

- ROC_AUC Score

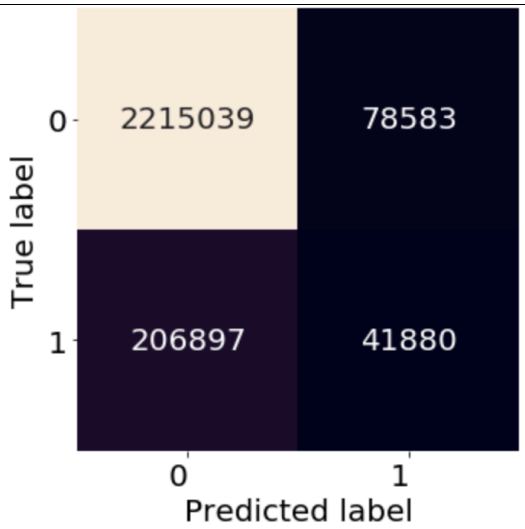
The AUC is the area under the ROC curve. It shows the tradeoff between sensitivity and specificity.

And roc_auc score gives us a good idea of how well the model performances. The AUC value lies between 0.5 to 1 where 0.5 denotes a bad classifier and 1 denotes an excellent classifier.

The area of this model's roc_auc curve is 78.22%, which approaches 80%. It in some degree shows that this a good classifier.

- Confusion Matrix

From the confusion matrix, we can get a better idea of what our model is getting right and what types of errors it made.



Above is the confusion matrix we got. There are two possible predicted classes: "0.0" and "1.0" which means not reordered and reordered for one product. Our classifier made a total of 1694933 predictions. Out of these cases, it predicted "1.0" 120463 times. In fact, 206897 product observations for one user did occur in his or her next order, but we predict them by mistake.

3.3.4.3 Conclusion

By using logistic regression model, we did a prediction on the next order for one user. The results reveal that the accuracy is 90% which is a high value. Most of our predictions are correct, however large amount of them came from the products did not occur in user next order.

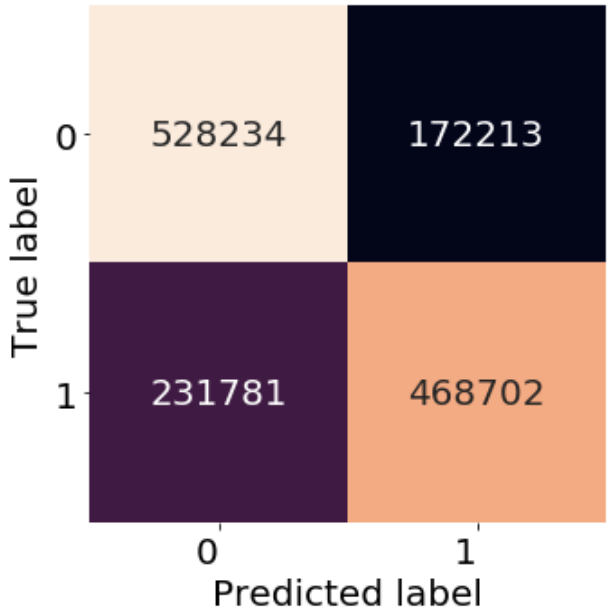
4. Results

4.1 Logistic regression model

From the classification report below we can see that the accuracy is 71.18%, which means the model can correctly classify 71.18% of the test dataset.

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.70      0.75      0.72    700447
         1.0       0.73      0.67      0.70    700483

    accuracy                           0.71   1400930
   macro avg       0.71      0.71      0.71   1400930
weighted avg       0.71      0.71      0.71   1400930


Accuracy :  71.16244209203886


ROC_AUC :  77.71164554475179
```

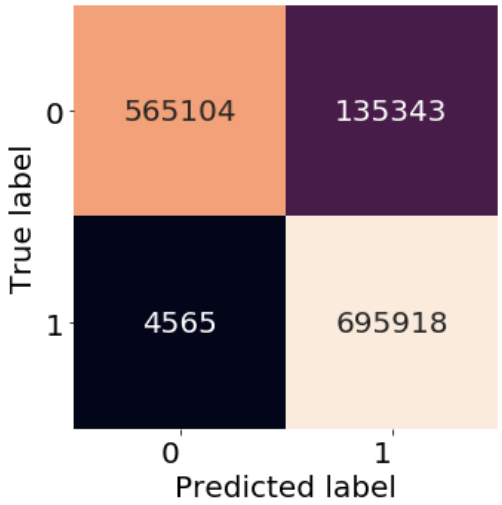
We can dive deeper by illustrating the confusion matrix. From the graph below, we know that our misclassification rate is 28.82%, which means that the probability that we classify mistakenly is 0.2882.



4.2 K-Nearest-Neighbor model

For k=3, both classification report and confusion matrix show that KNN classifier performs better than the logistic model. The accuracy increases to 90.01% and the misclassification rate decreases to 9.99%, which means the classifier can correctly classify 90.01% of the test dataset.

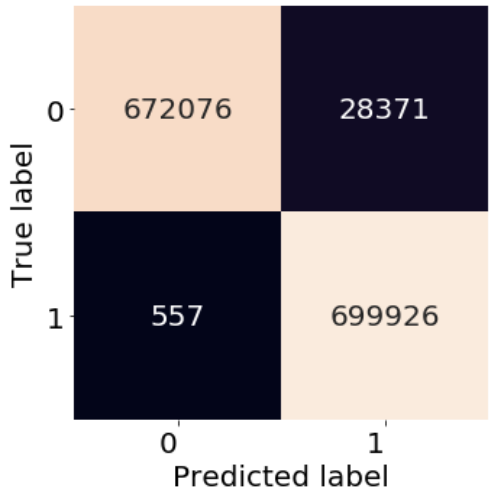
```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.81      0.89    700447
         1.0       0.84      0.99      0.91    700483

    accuracy                           0.90   1400930
   macro avg       0.91      0.90      0.90   1400930
weighted avg       0.91      0.90      0.90   1400930


Accuracy :  90.01320551348033


ROC_AUC :  99.97228979749896
```



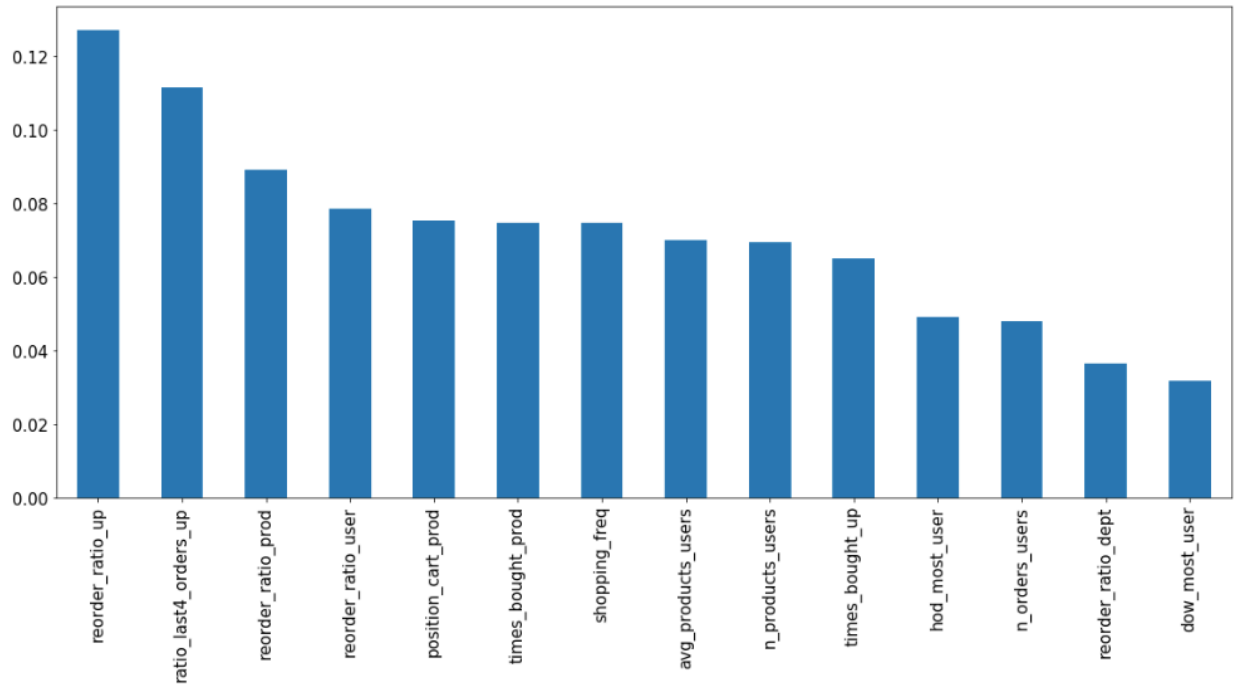
4.3 Random Forest Model

We can see from the classification report and confusion matrix below that the model performs excellent in the test dataset. The accuracy is 97.94% and the misclassification rate is 2.06%, which means the model can successfully predict 97.94% of the test dataset.

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98    700447
         1.0       0.96      1.00      0.98    700483

    accuracy                           0.98   1400930
   macro avg       0.98      0.98      0.98   1400930
weighted avg       0.98      0.98      0.98   1400930


Accuracy :  97.93508597859993


ROC_AUC :  99.97318652685728
```



Also, we plot feature importance distribution to illustrate which feature play an crucial role in the modeling. Here the most influential feature is ‘reorder_ratio_up’, which illustrates how often the user buy the product again after first purchase. The formula we use to calculate this feature is as follows: reorder_ratio_up = times a user buys a product/number of orders after the user buy the product for the first time. Similarly, feature ‘Ratio_last4_orders_up’ describes the users’ recent purchase preference by only considering the last four previous orders. On the other hand, feature ‘dow_most_user’ which illustrates the day in a week when a user tends to purchase has the least impact on the model.



5. Summary and Conclusions

Apart from the models we discuss before, we also develop other classification models, including decision tree, SVM, and Naïve Bayes. However, due to the limitation of our computers and the big size of data, SVM takes too long to run out. Naïve Bayes performs poorly in this case, and the performance of decision tree is similar to our baseline model. So, we do not discuss in details in terms of these models.

Our goal of this project is to predict whether a user will buy a product in the next order. First, we get familiar with the data through exploratory data analysis, and then we abstract 14 features from prior orders. We develop and compare different classification models and choose random forest as our final model with 97.94% accuracy. Also, we visualize the feature importance distribution to show how features influence the model separately.

We gain some practical experience during this project. We find that SVM may not be a good classifier for big data because it would take too long to see the result. On contrast, random forest performs best in this case, and it can also show the feature importance, which many other classifiers lack. However, one problem with random forest is that we cannot interpret it compared with k-nearest-neighbor. Although the accuracy of KNN is a bit lower than random forest, KNN has a more intuitive interpretation and is easy to understand.

6. Percentage of the code referenced

$$\frac{180\text{-}70}{180+20}=55\%$$

7. References

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. No. 10. New York: Springer series in statistics, 2001.