STAT4188_Instacart_Kaggle_Modeling

May 8, 2019

1 Instacart Reorders

1.1 Package Import

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

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn import tree

import xgboost as xgb
import imblearn.over_sampling
from collections import OrderedDict

# Notebook display settings
sns.set()
color = sns.color_palette()
%matplotlib inline
jtplot.style(theme='grade3') # Make graphs have white background
pd.options.mode.chained_assignment = None  # default='warn'

# Data Tables:
#    aisles
#    departments
#    order_products_prior
#    order_products_train
#    orders
#    products
```

1.2 Data Import

Created a 50,000 order subset of the data (2,000,000 orders requires cloud computing for full processing)

```
In [4]: # Unzip Instacart data subset ~50,000 orders
        # Only run once to create unzipped file

        !unzip instacart_data_subset.zip
```

```
Archive:  instacart_data_subset.zip
   creating: instacart_data_subset/
  inflating: instacart_data_subset/.DS_Store
   creating: __MACOSX/
   creating: __MACOSX/instacart_data_subset/
  inflating: __MACOSX/instacart_data_subset/._.DS_Store
  inflating: instacart_data_subset/aisles.csv
  inflating: __MACOSX/instacart_data_subset/._aisles.csv
  inflating: instacart_data_subset/departments.csv
  inflating: __MACOSX/instacart_data_subset/._departments.csv
  inflating: instacart_data_subset/instacart_df_X_features.csv
  inflating: instacart_data_subset/order_products__prior_subset.csv
  inflating: instacart_data_subset/order_products__train_subset.csv
  inflating: instacart_data_subset/orders_subset.csv
  inflating: instacart_data_subset/products.csv
  inflating: __MACOSX/instacart_data_subset/._products.csv
```

```
In [5]: # Define path of unzipped file
        path = 'instacart_data_subset/'
```

1.2.1 Setting up Dataframes

```
In [6]: # Orders dataframe
        orders_df = pd.read_csv(path + 'orders_subset.csv')
        orders_df.head(3)
```

Out[6]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	1363380	50	prior	1	3	9	NaN
1	3131103	50	prior	2	6	12	10.0
2	2197066	50	prior	3	1	13	9.0

```
In [7]: # Order Products Prior dataframe
        order_products_prior_df = pd.read_csv(path + 'order_products__prior_subset.csv')
        order_products_prior_df.head(3)
```

Out[7]:

	order_id	product_id	add_to_cart_order	reordered
0	12	30597	1	1
1	12	15221	2	1
2	12	43772	3	1

In [9]:
```
# Order Products Train dataframe
order_products_train_df = pd.read_csv(path + 'order_products__train_subset.csv')
order_products_train_df.head(3)
```

Out[9]:

	order_id	product_id	add_to_cart_order	reordered
0	1077	13176	1	1
1	1077	39922	2	1
2	1077	5258	3	1

In [10]:
```
# Aisles dataframe
aisles_df = pd.read_csv(path + 'aisles.csv')
aisles_df.head(3)
```

Out[10]:

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars

In [11]:
```
# Departments dataframe
departments_df = pd.read_csv(path + 'departments.csv')
departments_df.head(3)
```

Out[11]:

	department_id	department
0	1	frozen
1	2	other
2	3	bakery

In [12]:
```
# Products dataframe
products_df = pd.read_csv(path + 'products.csv')
products_df.head(3)
```

Out[12]:

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7

1.2.2 Orders Merged with Products Prior and Products Train

In [13]:
```
# Adding Orders dataframe info to both products train and prior
order_products_train_df = order_products_train_df.merge(orders_df.drop('eval_set', ax
order_products_prior_df = order_products_prior_df.merge(orders_df.drop('eval_set', ax
order_products_train_df.head(3)
```

Out[13]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	1077	13176	1	1	173934	11	
1	1077	39922	2	1	173934	11	
2	1077	5258	3	1	173934	11	

	order_dow	order_hour_of_day	days_since_prior_order
0	6	9	10.0
1	6	9	10.0
2	6	9	10.0

In [14]:
```
# Adding Products dataframe info to both products train and prior
order_products_train_df = order_products_train_df.merge(products_df.drop('product_name
order_products_prior_df = order_products_prior_df.merge(products_df.drop('product_name
order_products_train_df.head(3)
```

Out[14]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	1077	13176	1	1	173934	11	
1	1342	13176	1	1	156818	32	
2	6286	13176	1	1	185494	14	

	order_dow	order_hour_of_day	days_since_prior_order	aisle_id	\
0	6	9	10.0	24	
1	3	8	30.0	24	
2	3	9	9.0	24	

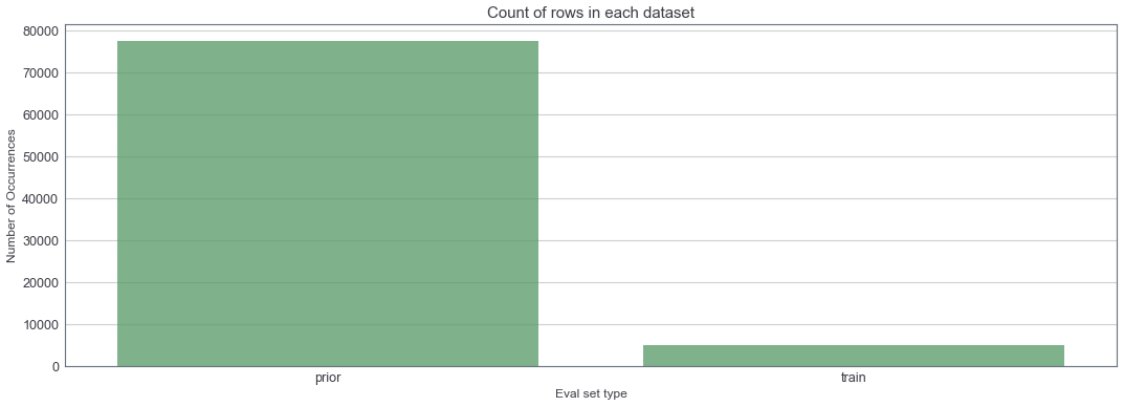
	department_id
0	4
1	4
2	4

In [15]:
```
order_products_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53704 entries, 0 to 53703
Data columns (total 11 columns):
order_id                  53704 non-null int64
product_id                53704 non-null int64
add_to_cart_order         53704 non-null int64
reordered                 53704 non-null int64
user_id                   53704 non-null int64
order_number              53704 non-null int64
order_dow                 53704 non-null int64
order_hour_of_day         53704 non-null int64
days_since_prior_order    53704 non-null float64
aisle_id                  53704 non-null int64
department_id             53704 non-null int64
dtypes: float64(1), int64(10)
memory usage: 4.9 MB
```

1.3 Baseline Data Visualization

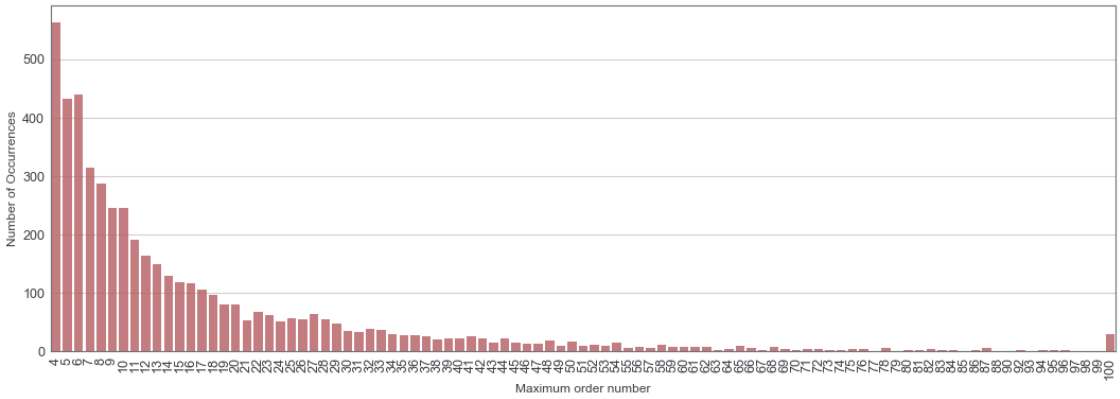
In [16]:
```
cnt_srs = orders_df.eval_set.value_counts()

plt.figure(figsize=(18,6))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Eval set type', fontsize=12)
plt.title('Count of rows in each dataset', fontsize=15)
plt.show()
```



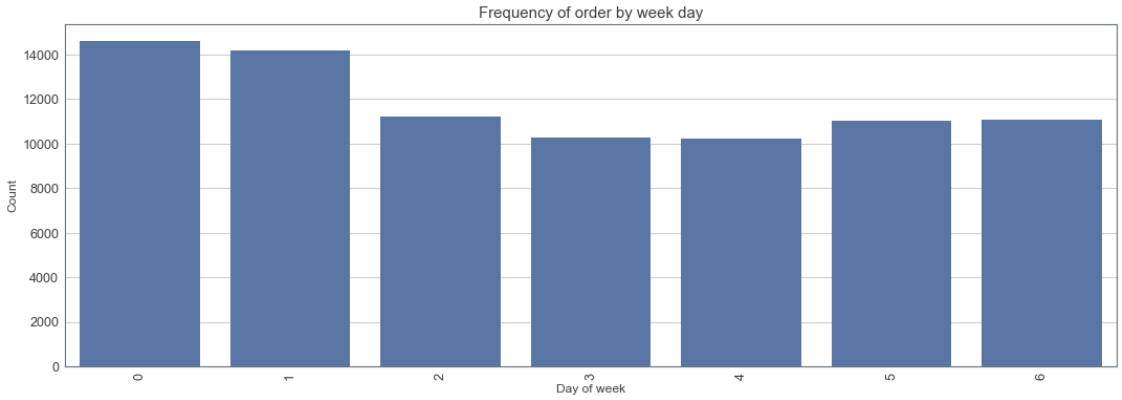
In [17]:
```
# Get unique count of number of customers
def get_unique_count(x):
    return len(np.unique(x))

cnt_srs = orders_df.groupby("eval_set")["user_id"].aggregate(get_unique_count)
cnt_srs
```

Out[17]:
```
eval_set
prior    5000
train    5000
Name: user_id, dtype: int64
```

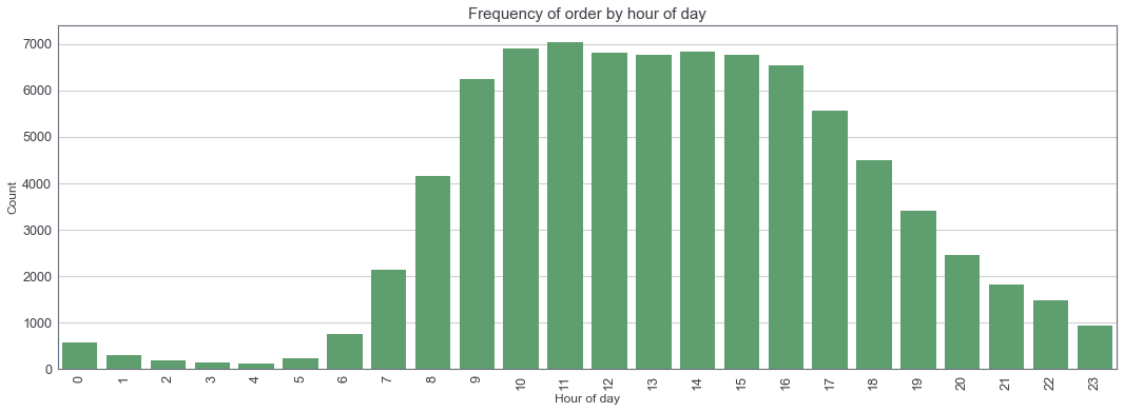
In [18]:
```
cnt_srs = orders_df.groupby("user_id")["order_number"].aggregate(np.max).reset_index()
cnt_srs = cnt_srs.order_number.value_counts()

plt.figure(figsize=(18,6))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8, color=color[2])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Maximum order number', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```



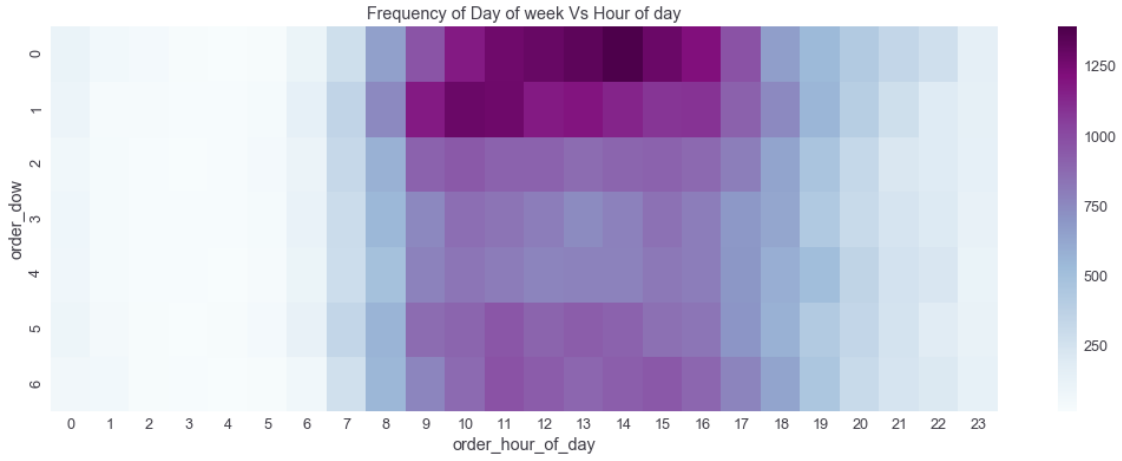

```
In [19]: plt.figure(figsize=(18,6))
         sns.countplot(x="order_dow", data=orders_df, color=color[0])
         plt.ylabel('Count', fontsize=12)
         plt.xlabel('Day of week', fontsize=12)
         plt.xticks(rotation='vertical')
         plt.title("Frequency of order by week day", fontsize=15)
         plt.show()
```




```
In [20]: plt.figure(figsize=(18,6))
         sns.countplot(x="order_hour_of_day", data=orders_df, color=color[1])
         plt.ylabel('Count', fontsize=12)
         plt.xlabel('Hour of day', fontsize=12)
         plt.xticks(rotation='vertical')
         plt.title("Frequency of order by hour of day", fontsize=15)
         plt.show()
```

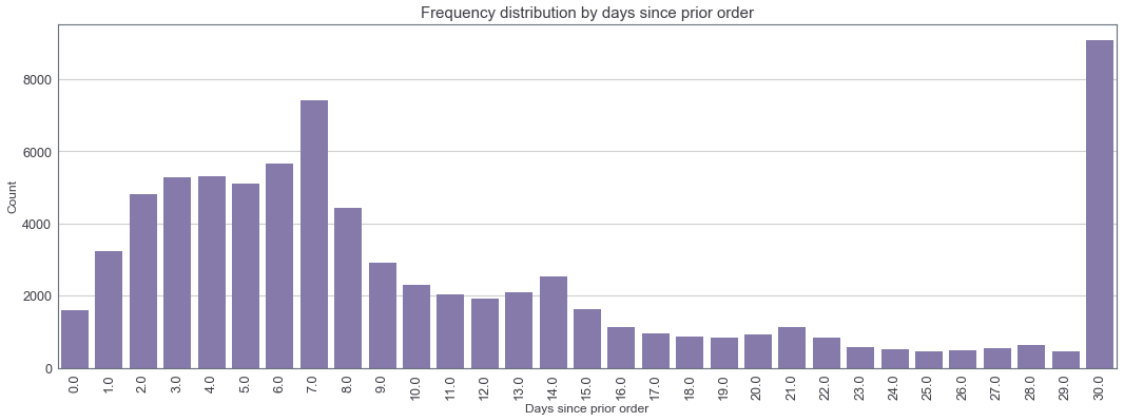


In [21]:
```
grouped_df = orders_df.groupby(["order_dow", "order_hour_of_day"])["order_number"].agg
grouped_df = grouped_df.pivot(index='order_dow', columns='order_hour_of_day', values=

plt.figure(figsize=(18,6))
sns.heatmap(grouped_df,cmap="BuPu")
plt.title("Frequency of Day of week Vs Hour of day")
plt.show()
```



In [22]:
```
plt.figure(figsize=(18,6))
sns.countplot(x="days_since_prior_order", data=orders_df, color=color[3])
plt.ylabel('Count', fontsize=12)
plt.xlabel('Days since prior order', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency distribution by days since prior order", fontsize=15)
plt.show()
```

```
# Reorders occur every most frequently once every 7 days or 30 days
# Also peaks at 14, 21, and 28 days
```



In [23]:
```
# percentage of re-orders in prior set
order_products_prior_df.reordered.sum() / order_products_prior_df.shape[0]
```

Out[23]:
```
0.584587441619097
```

In [24]:
```
# percentage of re-orders in train set
order_products_train_df.reordered.sum() / order_products_train_df.shape[0]
```

Out[24]:
```
0.6021711604349769
```

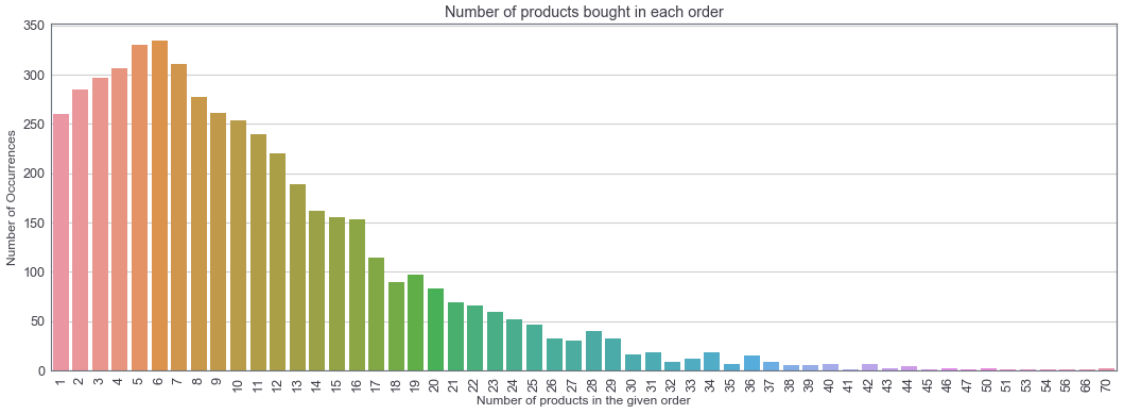
In [25]:
```
# Looking at orders with at least 1 reordered item in prior
grouped_df = order_products_prior_df.groupby("order_id")["reordered"].aggregate("sum")
grouped_df["reordered"].loc[grouped_df["reordered"]>1] = 1
grouped_df.reordered.value_counts() / grouped_df.shape[0]
# 12% of orders did not have a single reordered item
```

Out[25]:
```
1    0.881145
0    0.118855
Name: reordered, dtype: float64
```

In [26]:
```
# Looking at orders with at least 1 reordered item in train
grouped_df = order_products_train_df.groupby("order_id")["reordered"].aggregate("sum")
grouped_df["reordered"].loc[grouped_df["reordered"]>1] = 1
grouped_df.reordered.value_counts() / grouped_df.shape[0]
# 7% of orders did not have a single reordered item
```

Out[26]:
```
1    0.941
0    0.059
Name: reordered, dtype: float64
```

```
In [27]: grouped_df = order_products_train_df.groupby("order_id")["add_to_cart_order"].aggregat
         cnt_srs = grouped_df.add_to_cart_order.value_counts()

         plt.figure(figsize=(18,6))
         sns.barplot(cnt_srs.index, cnt_srs.values)
         plt.ylabel('Number of Occurrences', fontsize=12)
         plt.xlabel('Number of products in the given order', fontsize=12)
         plt.title('Number of products bought in each order')
         plt.xticks(rotation='vertical')
         plt.show()
```



1.4 Feature Engineering

1.4.1 Breakdown of Feature Categories

Product Features: general information about product purchase patterns across ALL users. The category of the product, its general popularity, how high priority the item tends to be, etc. * **product_total_orders**: how many times has a given product_id been ordered before across all orders and all customers * **product_avg_add_to_cart_order**: average order in which the product is being added to the cart across all orders and all customers * **product_aisle_id**: aisle_id of product * **product_department_id**: department_id of product * **product_avg_order_dow**: average day of the week (rounded) the product was ordered on * **product_avg_order_hour_of_day**: average hour of the day (rounded) the product was ordered during * **product_total_users**: total # of unique users that have ordered the product

User Features: information about specific user behavior. How many items do they tend to order, how long has it been since they've last ordered, what time of day do they usually order, etc. * **user_total_orders**: count of total prior orders by user id * **user_avg_cartsize**: average number of products across all carts by user id * **user_total_products**: count of unique product ids ordered across all orders by user id * **user_avg_days_since_prior_order**: average of days since prior order across all orders by user id

User-Product features: information about product-specific user behavior. How often have they ordered this product, how high-priority does it tend to be for them, how long has it

been since they've ordered this product, etc. * **user_product_avg_add_to_cart_order**: average order in which each product id is added to a cart across all orders by user id * **user_product_avg_order_dow**: average day of the week product is ordered on user-product level * **user_product_avg_order_hour_of_day**: average hour of the day the product is ordered on user-product level * **max_order_number_per_product**: max order number a product showed up in based on user_id * **user_product_order_freq**: how often the user bought the product out of the user's total orders * **user_product_orders_ago_ordered**: how many orders ago the user last ordered the product * **user_product_times_bought_in_last_five_orders**: how many times the user bought the product in their last 5 orders * **user_product_times_bought_in_last_ten_orders**: how many times the user bought the product in their last 10 orders * **weighted_reorder_sum**: a weighted sum of product reorders by user id (more weight the more recent the reorder) * **weighted_reorder_squared_sum**: a weighted sum of product reorders by user id (even more weight the more recent the reorder) * **consecutive_reorder_count**: counts how many times the product was consecutively reordered * **consecutive_reorder_percent**: percent of how many orders were consecutive as a percent of total orders for a product * **days_since_last_order**: count of days since the last time the product was orderd by the user

In [28]:
```
# Creating initial X_df to store product_id and user_id
user_product_df = (order_products_prior_df.groupby(['product_id','user_id'],as_index=
                                          .agg({'order_id':'count'})
                                          .rename(columns={'order_id':'user_product_t

train_ids = order_products_train_df['user_id'].unique()
X_df = user_product_df[user_product_df['user_id'].isin(train_ids)]
X_df = X_df.drop(['user_product_total_orders'],axis=1)
X_df.head()
```

Out[28]:

	product_id	user_id
0	1	21285
1	1	47549
2	1	54136
3	1	54240
4	1	95730

In [29]:
```
# Setting up latest_cart and in_cart to create target variable in X_df
train_carts = (order_products_train_df.groupby('user_id',as_index=False)
                                      .agg({'product_id':(lambda x: set(x))})
                                      .rename(columns={'product_id':'latest_cart'}))

X_df = X_df.merge(train_carts, on='user_id')
X_df['in_cart'] = (X_df.apply(lambda row: row['product_id'] in row['latest_cart'], ax
X_df.head()
```

Out[29]:

	product_id	user_id	latest_cart	\
0	1	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	
1	3298	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	
2	4920	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	
3	6066	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	
4	6184	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	

```
   in_cart
0        0
1        0
2        0
3        0
4        0
```

In [30]:
```
# Check proportion of train products that were reordered
X_df.in_cart.value_counts(normalize=True)
```

Out[30]:
```
0    0.901945
1    0.098055
Name: in_cart, dtype: float64
```

1.4.2 Product Features

In [31]:
```
# First category of features are product_id specific features

prod_features = ['product_total_orders','product_avg_add_to_cart_order','product_aisle
                 'product_avg_order_dow','product_avg_order_hour_of_day','product_tota

# Using max on aisle and dept id just as a trick to extract from groupby

prod_features_df = (order_products_prior_df.groupby(['product_id'],as_index=False)
                                           .agg(OrderedDict(
                                                   [('order_id','nunique'),
                                                    ('add_to_cart_order','mean'),
                                                    ('aisle_id','max'),
                                                    ('department_id','max'),
                                                    ('order_dow','mean'),
                                                    ('order_hour_of_day','mean'),
                                                    ('user_id','nunique')])))
prod_features_df.columns = ['product_id'] + prod_features

# Round back to get categorical days and binned hours
prod_features_df.product_avg_order_dow = prod_features_df.product_avg_order_dow.round
prod_features_df.product_avg_order_hour_of_day = prod_features_df.product_avg_order_ho

prod_features_df.head(3)
```

Out[31]:
```
   product_id  product_total_orders  product_avg_add_to_cart_order  \
0           1                    26                       4.576923
1           2                     1                       3.000000
2           3                     1                       4.000000

   product_aisle_id  product_department_id  product_avg_order_dow  \
0                61                     19                    3.0
1               104                     13                    4.0
```

```
2                94                      7                    4.0

   product_avg_order_hour_of_day  product_total_users
0                           13.0                  12
1                           12.0                   1
2                           17.0                   1
```

In [32]:
```
# Merge in product features into X_df on product_id
X_df = X_df.merge(prod_features_df, on='product_id')
X_df = X_df.dropna()
X_df.head(1)
```

Out[32]:
```
   product_id  user_id                                        latest_cart  \
0           1    21285  {21573, 35561, 37710, 11759, 12341, 13176, 32478}

   in_cart  product_total_orders  product_avg_add_to_cart_order  \
0        0                    26                       4.576923

   product_aisle_id  product_department_id  product_avg_order_dow  \
0                61                     19                    3.0

   product_avg_order_hour_of_day  product_total_users
0                           13.0                  12
```

In [33]:
```
# There is a risk to overfitting user specific behavior, which would make having the
# test bad (since you'd have an overfit model testing on the same user in the test se
# on user-product

# Baseline with Product Features

# Sample 20% of users
np.random.seed(45)
total_users = X_df['user_id'].unique()
test_users = np.random.choice(total_users, size=int(total_users.shape[0] * .20), repla

# Split into train/test
X_train_df, X_test_df = X_df[~X_df['user_id'].isin(test_users)], X_df[X_df['user_id']

# Split into features/target
y_train, y_test = X_train_df['in_cart'], X_test_df['in_cart']
X_train, X_test = X_train_df.drop(['product_id','user_id','latest_cart','in_cart'],ax
                  X_test_df.drop(['product_id','user_id','latest_cart','in_cart'],axis

# Fit simple baseline model
lr = LogisticRegression()
lr.fit(X_train, y_train)
metrics.f1_score(lr.predict(X_test), y_test)
```

```
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/metrics/classification.py:113
```

```
  'recall', 'true', average, warn_for)
```

Out[33]: 0.0

1.4.3 User Features

In [34]:
```
# Second category of features are user specific features

user_features = ['user_total_orders','user_avg_cartsize','user_total_products','user_a

user_features_df = (order_products_prior_df.groupby(['user_id'],as_index=False)
                                           .agg(OrderedDict(
                                                  [('order_id',['nunique', (lambda x
                                                   ('product_id','nunique'),
                                                   ('days_since_prior_order','mean')]

user_features_df.columns = ['user_id'] + user_features

user_features_df.head(3)
```

Out[34]:

	user_id	user_total_orders	user_avg_cartsize	user_total_products	user_avg_days_since_prior_order
0	50	67	6.761194	89	5.691275
1	52	27	6.259259	51	9.134969
2	65	14	9.428571	80	13.594828

In [35]:
```
# Merge user features into X_df on user_id
X_df = X_df.merge(user_features_df, on='user_id')
X_df = X_df.dropna()
X_df.head(1)
```

Out[35]:

	product_id	user_id	latest_cart	in_cart	product_total_orders	product_avg_add_to_cart_order	product_aisle_id	product_department_id	product_avg_order_dow	product_avg_order_hour_of_day	product_total_users	user_total_orders	user_avg_cartsize	user_total_products	user_avg_days_since_prior_order
0	1	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	0	26	4.576923	61	19	3.0	13.0	12	48	6.604167	46	8.044872

In [36]:
```
# Baseline with Product & User Features (using same user_id 20/80 split defined in Pr

# Split into train/test
X_train_df, X_test_df = X_df[~X_df['user_id'].isin(test_users)], X_df[X_df['user_id']

# Split into features/target
y_train, y_test = X_train_df['in_cart'], X_test_df['in_cart']
X_train, X_test = X_train_df.drop(['product_id','user_id','latest_cart','in_cart'],ax
                  X_test_df.drop(['product_id','user_id','latest_cart','in_cart'],axis

# Fit simple baseline model
lr = LogisticRegression()
lr.fit(X_train, y_train)
metrics.f1_score(lr.predict(X_test), y_test)
```

Out[36]:
```
0.014392991239048813
```

1.4.4 User-Product Features

User-Product Features Setup

In [37]:
```
# Creating a cart for each prior order (to be used to create User-Product features)
prior_carts = (order_products_prior_df.groupby(['user_id','order_id'],as_index=False)
                                      .agg({'product_id':(lambda x: set(x))})
                                      .rename(columns={'product_id':'order_id_cart'}))
prior_carts.head(2)
```

Out[37]:

	user_id	order_id	order_id_cart
0	50	103726	{39877, 6182, 31720, 22825, 34409, 42701, 2190...
1	50	131991	{7014, 6182, 47209, 47018, 19678}

In [38]:
```
# Adding cart for each order id to prior_products to set up for reorder comparison fe
order_products_prior_df = order_products_prior_df.merge(prior_carts.drop('user_id', ax
order_products_prior_df.head(2)
```

Out[38]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	12	30597	1	1	152610	22	
1	12	15221	2	1	152610	22	

	order_dow	order_hour_of_day	days_since_prior_order	aisle_id	\
0	6	8	10.0	53	
1	6	8	10.0	84	

	department_id	order_id_cart
0	16	{38050, 30597, 11175, 38888, 34335, 3164, 2339...
1	16	{38050, 30597, 11175, 38888, 34335, 3164, 2339...

In [39]:
```
# Creating a set that contains past order numbers in which the product_id was ordered
order_number_per_product = (order_products_prior_df.groupby(['product_id','user_id'],a
```

```
                                                .agg({'order_number':(lambda x: set(x))})
                                                .rename(columns={'order_number':'order_number_pe
order_number_per_product.head(2)
```

Out[39]:

	product_id	user_id	order_number_per_product
0	1	21285	{2}
1	1	47549	{8, 17, 11, 21}

```
In [40]: # Adding order_number set for each product_id & user_id to prior_products to set up fe
order_products_prior_df = order_products_prior_df.merge(order_number_per_product, on=
order_products_prior_df.head(2)
```

Out[40]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	12	30597	1	1	152610	22	
1	138033	30597	3	0	152610	2	

	order_dow	order_hour_of_day	days_since_prior_order	aisle_id	\
0	6	8	10.0	53	
1	4	13	13.0	53	

	department_id	order_id_cart	\
0	16	{38050, 30597, 11175, 38888, 34335, 3164, 2339...	
1	16	{44162, 46979, 30597, 28553, 49683, 27796, 348...	

	order_number_per_product
0	{2, 4, 5, 7, 22}
1	{2, 4, 5, 7, 22}

```
In [41]: # Add max order column to order_products_prior_df
# (max_order_number_per_product will also be used as a feature in X_df, as shown in U
order_products_prior_df["max_order_number_per_product"] = order_products_prior_df["ord
order_products_prior_df.head(2)
```

Out[41]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	12	30597	1	1	152610	22	
1	138033	30597	3	0	152610	2	

	order_dow	order_hour_of_day	days_since_prior_order	aisle_id	\
0	6	8	10.0	53	
1	4	13	13.0	53	

	department_id	order_id_cart	\
0	16	{38050, 30597, 11175, 38888, 34335, 3164, 2339...	
1	16	{44162, 46979, 30597, 28553, 49683, 27796, 348...	

	order_number_per_product	max_order_number_per_product
0	{2, 4, 5, 7, 22}	22
1	{2, 4, 5, 7, 22}	22

In [42]:
```
# Creating user_total_orders_minus_five column for user_product_times_bought_in_last_
user_total_orders = user_features_df.drop(['user_avg_cartsize','user_total_products',
user_total_orders['user_total_orders_minus_five'] = user_total_orders['user_total_orde
user_total_orders['user_total_orders_minus_ten'] = user_total_orders['user_total_order
order_products_prior_df = order_products_prior_df.merge(user_total_orders, on='user_id
order_products_prior_df.head(2)
```

Out[42]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	12	30597	1	1	152610	22	
1	138033	30597	3	0	152610	2	

	order_dow	order_hour_of_day	days_since_prior_order	aisle_id	\
0	6	8	10.0	53	
1	4	13	13.0	53	

	department_id	order_id_cart	\
0	16	{38050, 30597, 11175, 38888, 34335, 3164, 2339...	
1	16	{44162, 46979, 30597, 28553, 49683, 27796, 348...	

	order_number_per_product	max_order_number_per_product	user_total_orders	\
0	{2, 4, 5, 7, 22}	22	26	
1	{2, 4, 5, 7, 22}	22	26	

	user_total_orders_minus_five	user_total_orders_minus_ten
0	21	16
1	21	16

In [43]:
```
# Add last five orders feature using function
def last_five_orders(user_total_orders_minus_five,order_set):
    times_bought_in_last_five_order = 0
    for order_num in order_set:
        if order_num >= user_total_orders_minus_five:
            times_bought_in_last_five_order += 1
    return times_bought_in_last_five_order

# Add last ten orders feature using function
def last_ten_orders(user_total_orders_minus_ten,order_set):
    times_bought_in_last_ten_order = 0
    for order_num in order_set:
        if order_num >= user_total_orders_minus_ten:
            times_bought_in_last_ten_order += 1
    return times_bought_in_last_ten_order

order_products_prior_df["user_product_times_bought_in_last_five_orders"] = order_produ
order_products_prior_df["user_product_times_bought_in_last_ten_orders"] = order_produc
order_products_prior_df.head(2)
```

Out[43]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	12	30597	1	1	152610	22	

```
1    138033       30597                 3          0   152610             2

   order_dow  order_hour_of_day  days_since_prior_order  aisle_id  \
0          6                  8                    10.0        53
1          4                 13                    13.0        53

   department_id                                    order_id_cart  \
0             16  {38050, 30597, 11175, 38888, 34335, 3164, 2339...
1             16  {44162, 46979, 30597, 28553, 49683, 27796, 348...

  order_number_per_product  max_order_number_per_product  user_total_orders  \
0         {2, 4, 5, 7, 22}                            22                 26
1         {2, 4, 5, 7, 22}                            22                 26

   user_total_orders_minus_five  user_total_orders_minus_ten  \
0                            21                           16
1                            21                           16

   user_product_times_bought_in_last_five_orders  \
0                                              1
1                                              1

   user_product_times_bought_in_last_ten_orders
0                                             1
1                                             1
```

In [44]:
```
# Weighted reorder feature (Sum(order_number/user_total_orders))
def weighted_reorder_sum(order_set,user_total_orders):
    weighted_reorder_sum = 0
    for order_num in order_set:
        x = order_num / user_total_orders
        weighted_reorder_sum += x
    return weighted_reorder_sum

# Weighted reorder feature (Sum(squared(order_number/user_total_orders)))
def weighted_reorder_squared_sum(order_set,user_total_orders):
    weighted_reorder_squared_sum = 0
    for order_num in order_set:
        x = (order_num / user_total_orders)**2
        weighted_reorder_squared_sum += x
    return weighted_reorder_squared_sum

order_products_prior_df["weighted_reorder_sum"] = order_products_prior_df.apply(lambda
order_products_prior_df["weighted_reorder_squared_sum"] = order_products_prior_df.appl
order_products_prior_df.head(2)
```

Out[44]:
```
   order_id  product_id  add_to_cart_order  reordered  user_id  order_number  \
0        12       30597                  1          1   152610            22
```

```
1    138033       30597                  3          0   152610             2

   order_dow  order_hour_of_day  days_since_prior_order  aisle_id  \
0          6                  8                    10.0        53
1          4                 13                    13.0        53

               ...              \
0              ...
1              ...

                                        order_id_cart order_number_per_product  \
0  {38050, 30597, 11175, 38888, 34335, 3164, 2339...         {2, 4, 5, 7, 22}
1  {44162, 46979, 30597, 28553, 49683, 27796, 348...         {2, 4, 5, 7, 22}

  max_order_number_per_product  user_total_orders  \
0                           22                26
1                           22                26

   user_total_orders_minus_five  user_total_orders_minus_ten  \
0                            21                           16
1                            21                           16

   user_product_times_bought_in_last_five_orders  \
0                                              1
1                                              1

   user_product_times_bought_in_last_ten_orders  weighted_reorder_sum  \
0                                             1              1.538462
1                                             1              1.538462

   weighted_reorder_squared_sum
0                       0.85503
1                       0.85503

[2 rows x 21 columns]
```

In [45]:
```python
# Consecutive reorders count feature
def consecutive_reorder_count(order_set):
    consecutive_reorder_count = 0
    order_set = sorted(order_set)
    order_set = list(order_set)
    if len(order_set) != 1:
        for idx,order_num in enumerate(order_set[:-1]):
            x1 = order_num
            x2 = order_set[idx+1]
            if x1+1 == x2:
                consecutive_reorder_count += 1
        return consecutive_reorder_count
```

```
        else:
            return 0

    # Consecutive reorders as a percent of total orders feature
    def consecutive_reorder_percent(order_set):
        consecutive_reorder_count = 0
        order_set = sorted(order_set)
        order_set = list(order_set)
        if len(order_set) != 1:
            for idx,order_num in enumerate(order_set[:-1]):
                x1 = order_num
                x2 = order_set[idx+1]
                if x1+1 == x2:
                    consecutive_reorder_count += 1
            return consecutive_reorder_count / len(order_set)
        else:
            return 0

    order_products_prior_df["consecutive_reorder_count"] = order_products_prior_df.apply(
    order_products_prior_df["consecutive_reorder_percent"] = order_products_prior_df.apply
    order_products_prior_df.head(2)
```

Out[45]:

	order_id	product_id	add_to_cart_order	reordered	user_id	order_number	\
0	12	30597	1	1	152610	22	
1	138033	30597	3	0	152610	2	

	order_dow	order_hour_of_day	days_since_prior_order	aisle_id	\
0	6	8	10.0	53	
1	4	13	13.0	53	

	...	max_order_number_per_product	\
0	...	22	
1	...	22	

	user_total_orders	user_total_orders_minus_five	user_total_orders_minus_ten	\
0	26	21	16	
1	26	21	16	

	user_product_times_bought_in_last_five_orders	\
0	1	
1	1	

	user_product_times_bought_in_last_ten_orders	weighted_reorder_sum	\
0	1	1.538462	
1	1	1.538462	

	weighted_reorder_squared_sum	consecutive_reorder_count	\
0	0.85503	1	

```
1                       0.85503                          1

   consecutive_reorder_percent
0                          0.2
1                          0.2

[2 rows x 23 columns]
```

In [46]:
```
# Creating tuple for each user_id which stores sorted days_since_prior_order values
user_order_df = order_products_prior_df.sort_values(['user_id','order_number'])

user_order_df = (user_order_df.groupby(['user_id','order_number'],as_index=False)
                              .agg({'days_since_prior_order':'max'}))

user_days_since_prior_order = (user_order_df.groupby(['user_id'],as_index=False)
                                            .agg({'days_since_prior_order':(lambda x:
                                            .rename(columns={'days_since_prior_order'

order_products_prior_df = order_products_prior_df.merge(user_days_since_prior_order, o
order_products_prior_df.head(2)
```

Out[46]:
```
   order_id  product_id  add_to_cart_order  reordered  user_id  order_number  \
0        12       30597                  1          1   152610            22
1    138033       30597                  3          0   152610             2

   order_dow  order_hour_of_day  days_since_prior_order  aisle_id  \
0          6                  8                    10.0        53
1          4                 13                    13.0        53

                         ...                          user_total_orders  \
0                        ...                                         26
1                        ...                                         26

  user_total_orders_minus_five user_total_orders_minus_ten  \
0                           21                          16
1                           21                          16

   user_product_times_bought_in_last_five_orders  \
0                                              1
1                                              1

   user_product_times_bought_in_last_ten_orders  weighted_reorder_sum  \
0                                             1              1.538462
1                                             1              1.538462

   weighted_reorder_squared_sum  consecutive_reorder_count  \
0                       0.85503                          1
1                       0.85503                          1
```

```
   consecutive_reorder_percent  \
0                          0.2
1                          0.2

                        days_since_prior_order_tuple
0  (nan, 13.0, 30.0, 8.0, 16.0, 4.0, 19.0, 10.0, ...
1  (nan, 13.0, 30.0, 8.0, 16.0, 4.0, 19.0, 10.0, ...

[2 rows x 24 columns]
```

In [48]:
```
# Days since last order feature
def days_since_last_order(max_order_number_per_product,days_since_prior_order_tuple):
    days_since_prior_order_tuple = np.nan_to_num(days_since_prior_order_tuple)
    return sum(days_since_prior_order_tuple[max_order_number_per_product-1:])

# Max days between orders feature
def max_days_between_orders(order_number_per_product,days_since_prior_order_tuple,max_
    days_since_prior_order_tuple = np.nan_to_num(days_since_prior_order_tuple)
    order_number_per_product = list(order_number_per_product)
    if len(order_number_per_product) != 1:
        list_of_sums = []
        for idx,order_num in enumerate(order_number_per_product[:-1]):
            start_pos = order_num - 1
            length = order_number_per_product[idx+1] - order_num
            list_of_sums.append(sum(days_since_prior_order_tuple[start_pos:start_pos+l
        return max(list_of_sums)
    else:
        return sum(days_since_prior_order_tuple[max_order_number_per_product-1:])

order_products_prior_df['days_since_last_order'] = order_products_prior_df.apply(lambd
order_products_prior_df['max_days_between_orders'] = order_products_prior_df.apply(lam
order_products_prior_df.head(2)

# DEBUG + Subtract the two features to create a third
```

Out[48]:
```
   order_id  product_id  add_to_cart_order  reordered  user_id  order_number  \
0        12       30597                  1          1   152610            22
1    138033       30597                  3          0   152610             2

   order_dow  order_hour_of_day  days_since_prior_order  aisle_id  \
0          6                  8                    10.0        53
1          4                 13                    13.0        53

             ...            user_total_orders_minus_ten  \
0            ...                                     16
1            ...                                     16
```

```
  user_product_times_bought_in_last_five_orders  \
0                                             1
1                                             1

  user_product_times_bought_in_last_ten_orders  weighted_reorder_sum  \
0                                            1              1.538462
1                                            1              1.538462

   weighted_reorder_squared_sum  consecutive_reorder_count  \
0                       0.85503                          1
1                       0.85503                          1

   consecutive_reorder_percent  \
0                          0.2
1                          0.2

                        days_since_prior_order_tuple  days_since_last_order  \
0  (nan, 13.0, 30.0, 8.0, 16.0, 4.0, 19.0, 10.0, ...                   40.0
1  (nan, 13.0, 30.0, 8.0, 16.0, 4.0, 19.0, 10.0, ...                   40.0

   max_days_between_orders
0                    166.0
1                    166.0

[2 rows x 26 columns]
```

User-Product Features

```
In [49]: # Third category of features are user-product specific features

         user_prod_features = ['user_product_avg_add_to_cart_order','user_product_total_orders
                               'user_product_avg_order_hour_of_day','max_order_number_per_produ
                               'user_product_times_bought_in_last_ten_orders','weighted_reorder
                               'consecutive_reorder_count','consecutive_reorder_percent','days_

         user_prod_features_df = (order_products_prior_df.groupby(['product_id','user_id'],as_i
                                                          .agg(OrderedDict(
                                                               [('add_to_cart_order','mean'),
                                                                ('order_id','count'),
                                                                ('order_dow','mean'),
                                                                ('order_hour_of_day','mean'),
                                                                ('max_order_number_per_product'
                                                                ('user_product_times_bought_in_l
                                                                ('user_product_times_bought_in_l
                                                                ('weighted_reorder_sum','max'),
                                                                ('weighted_reorder_squared_sum'
                                                                ('consecutive_reorder_count','ma
                                                                ('consecutive_reorder_percent',
```

```
                                                        ('days_since_last_order','min')
                                                        ('max_days_between_orders','min

user_prod_features_df.columns = ['product_id','user_id'] + user_prod_features

# Round back to get categorical days and binned hours
user_prod_features_df.user_product_avg_order_dow = user_prod_features_df.user_product_
user_prod_features_df.user_product_avg_order_hour_of_day = user_prod_features_df.user_

user_prod_features_df.head(10)
```

Out[49]:

	product_id	user_id	user_product_avg_add_to_cart_order	\
0	1	21285	3.0	
1	1	47549	4.0	
2	1	54136	3.0	
3	1	54240	2.0	
4	1	95730	1.0	
5	1	96682	4.0	
6	1	109784	8.0	
7	1	110040	2.5	
8	1	118812	11.0	
9	1	126451	3.5	

	user_product_total_orders	user_product_avg_order_dow	\
0	1	0.0	
1	4	4.0	
2	1	2.0	
3	1	2.0	
4	1	1.0	
5	8	3.0	
6	1	3.0	
7	2	1.0	
8	1	3.0	
9	2	4.0	

	user_product_avg_order_hour_of_day	max_order_number_per_product	\
0	15.0	2	
1	18.0	21	
2	13.0	18	
3	16.0	1	
4	14.0	7	
5	12.0	23	
6	7.0	5	
7	16.0	28	
8	19.0	11	
9	8.0	7	

	user_product_times_bought_in_last_five_orders	\

```
0                                                 0
1                                                 1
2                                                 1
3                                                 1
4                                                 1
5                                                 1
6                                                 1
7                                                 0
8                                                 0
9                                                 2

   user_product_times_bought_in_last_ten_orders  weighted_reorder_sum  \
0                                             0              0.041667
1                                             2              2.192308
2                                             1              0.782609
3                                             1              0.200000
4                                             1              0.777778
5                                             1              2.888889
6                                             1              1.000000
7                                             0              0.414634
8                                             1              0.647059
9                                             2              1.857143

   weighted_reorder_squared_sum  consecutive_reorder_count  \
0                      0.001736                          0
1                      1.353550                          0
2                      0.612476                          0
3                      0.040000                          0
4                      0.604938                          0
5                      1.448560                          3
6                      1.000000                          0
7                      0.121951                          0
8                      0.418685                          0
9                      1.734694                          1

   consecutive_reorder_percent  days_since_last_order  max_days_between_orders
0                        0.000                  356.0                    356.0
1                        0.000                  110.0                    100.0
2                        0.000                   97.0                     97.0
3                        0.000                  120.0                    120.0
4                        0.000                   74.0                     74.0
5                        0.375                   64.0                    148.0
6                        0.000                   30.0                     30.0
7                        0.000                  259.0                      0.0
8                        0.000                  145.0                    145.0
9                        0.500                   22.0                     30.0
```

In [50]: *# Merge user-product features into X_df based on user_id & product_id*

```
X_df = X_df.merge(user_prod_features_df, on=['user_id','product_id'])

# Creating user_product_order_freq outide of user_prod_features cell because feature
X_df['user_product_order_freq'] = X_df['user_product_total_orders'] / X_df['user_total
X_df['user_product_orders_ago_ordered'] = (X_df['user_total_orders'] + 1) - X_df['max_

X_df = X_df.dropna()
X_df.head(1)
```

Out[50]:

	product_id	user_id	latest_cart	\
0	1	21285	{21573, 35561, 37710, 11759, 12341, 13176, 32478}	

	in_cart	product_total_orders	product_avg_add_to_cart_order	\
0	0	26	4.576923	

	product_aisle_id	product_department_id	product_avg_order_dow	\
0	61	19	3.0	

	product_avg_order_hour_of_day	...	\
0	13.0	...	

	user_product_times_bought_in_last_five_orders	\
0	0	

	user_product_times_bought_in_last_ten_orders	weighted_reorder_sum	\
0	0	0.041667	

	weighted_reorder_squared_sum	consecutive_reorder_count	\
0	0.001736	0	

	consecutive_reorder_percent	days_since_last_order	\
0	0.0	356.0	

	max_days_between_orders	user_product_order_freq	\
0	356.0	0.020833	

	user_product_orders_ago_ordered
0	47

[1 rows x 30 columns]

In [51]:
```
# Full feature and target set
X_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 329806 entries, 0 to 329805
Data columns (total 30 columns):
product_id                                  329806 non-null int64
```

```
user_id                                              329806 non-null int64
latest_cart                                          329806 non-null object
in_cart                                              329806 non-null int64
product_total_orders                                 329806 non-null int64
product_avg_add_to_cart_order                        329806 non-null float64
product_aisle_id                                     329806 non-null int64
product_department_id                                329806 non-null int64
product_avg_order_dow                                329806 non-null float64
product_avg_order_hour_of_day                        329806 non-null float64
product_total_users                                  329806 non-null int64
user_total_orders                                    329806 non-null int64
user_avg_cartsize                                    329806 non-null float64
user_total_products                                  329806 non-null int64
user_avg_days_since_prior_order                      329806 non-null float64
user_product_avg_add_to_cart_order                   329806 non-null float64
user_product_total_orders                            329806 non-null int64
user_product_avg_order_dow                           329806 non-null float64
user_product_avg_order_hour_of_day                   329806 non-null float64
max_order_number_per_product                         329806 non-null int64
user_product_times_bought_in_last_five_orders        329806 non-null int64
user_product_times_bought_in_last_ten_orders         329806 non-null int64
weighted_reorder_sum                                 329806 non-null float64
weighted_reorder_squared_sum                         329806 non-null float64
consecutive_reorder_count                            329806 non-null int64
consecutive_reorder_percent                          329806 non-null float64
days_since_last_order                                329806 non-null float64
max_days_between_orders                              329806 non-null float64
user_product_order_freq                              329806 non-null float64
user_product_orders_ago_ordered                      329806 non-null int64
dtypes: float64(14), int64(15), object(1)
memory usage: 78.0+ MB
```

In [52]:
```
# Baseline with Product, User, & User-Product Features  (using same user_id 20/80 spl

# Split into train/test
X_train_df, X_test_df = X_df[~X_df['user_id'].isin(test_users)], X_df[X_df['user_id']

# Split into features/target
y_train, y_test = X_train_df['in_cart'], X_test_df['in_cart']
X_train, X_test = X_train_df.drop(['product_id','user_id','latest_cart','in_cart',],ax
                  X_test_df.drop(['product_id','user_id','latest_cart','in_cart'],axis

# Fit simple baseline model
lr = LogisticRegression()
lr.fit(X_train, y_train)
metrics.f1_score(y_test, lr.predict(X_test))
```

Out[52]:
```
0.2670993509735397
```

1.5 Modeling

1.5.1 Splitting up data

In [53]:
```
# Sample 20% of users for holdout (test)
np.random.seed(45)
total_users = X_df['user_id'].unique()
test_users = np.random.choice(total_users, size=int(total_users.shape[0] * .20), repla

# Split into train/test on 80/20 split of users
X_train_df, X_test_df = X_df[~X_df['user_id'].isin(test_users)], X_df[X_df['user_id']

# Split train into 5 folds for cross-validation
train_users = X_train_df['user_id'].unique()
np.random.shuffle(train_users)
train_users_5folds = np.split(train_users,5)

# Test sets for cross-validation
X_valid_df_fold1 = X_train_df[X_train_df['user_id'].isin(train_users_5folds[0])]
X_valid_df_fold2 = X_train_df[X_train_df['user_id'].isin(train_users_5folds[1])]
X_valid_df_fold3 = X_train_df[X_train_df['user_id'].isin(train_users_5folds[2])]
X_valid_df_fold4 = X_train_df[X_train_df['user_id'].isin(train_users_5folds[3])]
X_valid_df_fold5 = X_train_df[X_train_df['user_id'].isin(train_users_5folds[4])]

# Train sets for cross-validation
X_train_df_fold1 = X_train_df[X_train_df['user_id'].isin(np.concatenate(train_users_5
X_train_df_fold2 = X_train_df[X_train_df['user_id'].isin(np.concatenate((train_users_
X_train_df_fold3 = X_train_df[X_train_df['user_id'].isin(np.concatenate((train_users_
X_train_df_fold4 = X_train_df[X_train_df['user_id'].isin(np.concatenate((train_users_
X_train_df_fold5 = X_train_df[X_train_df['user_id'].isin(np.concatenate(train_users_5

# Split into features/target for test set and cross validation folds
y_test = X_test_df['in_cart']
X_test = X_test_df.drop(['product_id','user_id','latest_cart','in_cart'],axis=1)

y_train_fold1 = X_train_df_fold1['in_cart']
y_train_fold2 = X_train_df_fold2['in_cart']
y_train_fold3 = X_train_df_fold3['in_cart']
y_train_fold4 = X_train_df_fold4['in_cart']
y_train_fold5 = X_train_df_fold5['in_cart']

X_train_fold1 = X_train_df_fold1.drop(['product_id','user_id','latest_cart','in_cart'
X_train_fold2 = X_train_df_fold2.drop(['product_id','user_id','latest_cart','in_cart'
X_train_fold3 = X_train_df_fold3.drop(['product_id','user_id','latest_cart','in_cart'
X_train_fold4 = X_train_df_fold4.drop(['product_id','user_id','latest_cart','in_cart'
X_train_fold5 = X_train_df_fold5.drop(['product_id','user_id','latest_cart','in_cart'

y_valid_fold1 = X_valid_df_fold1['in_cart']
```

```
y_valid_fold2 = X_valid_df_fold2['in_cart']
y_valid_fold3 = X_valid_df_fold3['in_cart']
y_valid_fold4 = X_valid_df_fold4['in_cart']
y_valid_fold5 = X_valid_df_fold5['in_cart']

X_valid_fold1 = X_valid_df_fold1.drop(['product_id','user_id','latest_cart','in_cart'
X_valid_fold2 = X_valid_df_fold2.drop(['product_id','user_id','latest_cart','in_cart'
X_valid_fold3 = X_valid_df_fold3.drop(['product_id','user_id','latest_cart','in_cart'
X_valid_fold4 = X_valid_df_fold4.drop(['product_id','user_id','latest_cart','in_cart'
X_valid_fold5 = X_valid_df_fold5.drop(['product_id','user_id','latest_cart','in_cart'
```

In [54]:
```
# Checking columns
for x in zip(list(X_test.columns),list(X_train_fold1.columns)):
    print(x)
```

```
('product_total_orders', 'product_total_orders')
('product_avg_add_to_cart_order', 'product_avg_add_to_cart_order')
('product_aisle_id', 'product_aisle_id')
('product_department_id', 'product_department_id')
('product_avg_order_dow', 'product_avg_order_dow')
('product_avg_order_hour_of_day', 'product_avg_order_hour_of_day')
('product_total_users', 'product_total_users')
('user_total_orders', 'user_total_orders')
('user_avg_cartsize', 'user_avg_cartsize')
('user_total_products', 'user_total_products')
('user_avg_days_since_prior_order', 'user_avg_days_since_prior_order')
('user_product_avg_add_to_cart_order', 'user_product_avg_add_to_cart_order')
('user_product_total_orders', 'user_product_total_orders')
('user_product_avg_order_dow', 'user_product_avg_order_dow')
('user_product_avg_order_hour_of_day', 'user_product_avg_order_hour_of_day')
('max_order_number_per_product', 'max_order_number_per_product')
('user_product_times_bought_in_last_five_orders', 'user_product_times_bought_in_last_five_orde
('user_product_times_bought_in_last_ten_orders', 'user_product_times_bought_in_last_ten_orders
('weighted_reorder_sum', 'weighted_reorder_sum')
('weighted_reorder_squared_sum', 'weighted_reorder_squared_sum')
('consecutive_reorder_count', 'consecutive_reorder_count')
('consecutive_reorder_percent', 'consecutive_reorder_percent')
('days_since_last_order', 'days_since_last_order')
('max_days_between_orders', 'max_days_between_orders')
('user_product_order_freq', 'user_product_order_freq')
('user_product_orders_ago_ordered', 'user_product_orders_ago_ordered')
```

1.5.2 Oversampling Validation Folds

In [55]:
```
# Randomly oversample positive samples to be 40% of targets
ROS = imblearn.over_sampling.RandomOverSampler(ratio=.4, random_state=42)
```

```
    # Note oversampling converts to arrays
    X_tr_os1, y_tr_os1 = ROS.fit_sample(X_train_fold1, y_train_fold1)
    X_tr_os2, y_tr_os2 = ROS.fit_sample(X_train_fold2, y_train_fold2)
    X_tr_os3, y_tr_os3 = ROS.fit_sample(X_train_fold3, y_train_fold3)
    X_tr_os4, y_tr_os4 = ROS.fit_sample(X_train_fold4, y_train_fold4)
    X_tr_os5, y_tr_os5 = ROS.fit_sample(X_train_fold5, y_train_fold5)

    # Convert X_train folds to dataframes
    X_tr_os1 = pd.DataFrame(X_tr_os1)
    X_tr_os1.columns = X_train_fold1.columns
    X_tr_os2 = pd.DataFrame(X_tr_os2)
    X_tr_os2.columns = X_train_fold2.columns
    X_tr_os3 = pd.DataFrame(X_tr_os3)
    X_tr_os3.columns = X_train_fold3.columns
    X_tr_os4 = pd.DataFrame(X_tr_os4)
    X_tr_os4.columns = X_train_fold4.columns
    X_tr_os5 = pd.DataFrame(X_tr_os5)
    X_tr_os5.columns = X_train_fold5.columns

    # Convert y_train folds to series
    y_tr_os1 = pd.Series(y_tr_os1)
    y_tr_os2 = pd.Series(y_tr_os2)
    y_tr_os3 = pd.Series(y_tr_os3)
    y_tr_os4 = pd.Series(y_tr_os4)
    y_tr_os5 = pd.Series(y_tr_os5)
```

```
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/utils/deprecation.py:77: Depre
  warnings.warn(msg, category=DeprecationWarning)
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/utils/deprecation.py:77: Depre
  warnings.warn(msg, category=DeprecationWarning)
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/utils/deprecation.py:77: Depre
  warnings.warn(msg, category=DeprecationWarning)
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/utils/deprecation.py:77: Depre
  warnings.warn(msg, category=DeprecationWarning)
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/utils/deprecation.py:77: Depre
  warnings.warn(msg, category=DeprecationWarning)
```

1.5.3 Fitting Models

```
In [56]: # Setup for modeling functions
         X_tr_os_list = [X_tr_os1,X_tr_os2,X_tr_os3,X_tr_os4,X_tr_os5]
         y_tr_os_list = [y_tr_os1,y_tr_os2,y_tr_os3,y_tr_os4,y_tr_os5]
         X_valid_list = [X_valid_fold1,X_valid_fold2,X_valid_fold3,X_valid_fold4,X_valid_fold5]
         y_valid_list = [y_valid_fold1,y_valid_fold2,y_valid_fold3,y_valid_fold4,y_valid_fold5]
```

Logistic Regression

In [57]:
```
# Logistic Regression cross validation function

def log_reg_cross_val(x_train,y_train,x_val,y_val):
    """Take training and validation data in folds and fit logistic regression and cal
    f1_scores = []
    auc_scores = []
    for x in range(len(x_train)):
        lr_os = LogisticRegression()
        lr_os.fit(x_train[x], y_train[x])
        log_reg_f1, log_reg_auc = metrics.f1_score(y_val[x], lr_os.predict(x_val[x]))
        f1_scores.append(log_reg_f1)
        auc_scores.append(log_reg_auc)
        print('Logistic Regression on Oversampled Train Data Fold %.0f  Test F1: %.3f
    print('Logistic Regression on Oversampled Train Data; Mean F1: %.3f' %np.mean(f1_s
    print('Logistic Regression on Oversampled Train Data; Mean AUC: %.3f' %np.mean(auc

log_reg_cross_val(X_tr_os_list,y_tr_os_list,X_valid_list,y_valid_list)
```

```
Logistic Regression on Oversampled Train Data Fold 1  Test F1: 0.430, Test AUC: 0.828
Logistic Regression on Oversampled Train Data Fold 2  Test F1: 0.413, Test AUC: 0.813
Logistic Regression on Oversampled Train Data Fold 3  Test F1: 0.443, Test AUC: 0.823
Logistic Regression on Oversampled Train Data Fold 4  Test F1: 0.419, Test AUC: 0.813
Logistic Regression on Oversampled Train Data Fold 5  Test F1: 0.432, Test AUC: 0.825
Logistic Regression on Oversampled Train Data; Mean F1: 0.427
Logistic Regression on Oversampled Train Data; Mean AUC: 0.820
```

Decision Tree

In [58]:
```
# Test of Decision Tree
dtr_os = tree.DecisionTreeClassifier(min_samples_split=10, max_depth=6, min_samples_le
dtr_os.fit(X_tr_os_list[0], y_tr_os_list[0])
dtr_f1, dtr_auc = metrics.f1_score(y_valid_list[0], dtr_os.predict(X_valid_list[0])),
print(dtr_f1)
print(dtr_auc)
```

```
0.42600253271422545
0.8229352239032384
```

In [59]:
```
# Decision Tree cross validation function

def decision_tree_cross_val(x_train,y_train,x_val,y_val,min_samples_split=10, max_dept
    """Take training and validation data in folds and fit decision tree and calculate
    f1_scores = []
    auc_scores = []
    for x in range(len(x_train)):
        dtr_os = tree.DecisionTreeClassifier(min_samples_split=min_samples_split,max_
```

```
            dtr_os.fit(x_train[x], y_train[x])
            dtr_f1, dtr_auc = metrics.f1_score(y_val[x], dtr_os.predict(x_val[x])), metric
            f1_scores.append(dtr_f1)
            auc_scores.append(dtr_auc)
            print('Decision Tree on Oversampled Train Data Fold %.0f  Test F1: %.3f, Test
        print('Decision Tree on Oversampled Train Data; Mean F1: %.3f' %np.mean(f1_scores)
        print('Decision Tree on Oversampled Train Data; Mean AUC: %.3f' %np.mean(auc_score

    decision_tree_cross_val(X_tr_os_list,y_tr_os_list,X_valid_list,y_valid_list)
```

```
Decision Tree on Oversampled Train Data Fold 1  Test F1: 0.426, Test AUC: 0.823
Decision Tree on Oversampled Train Data Fold 2  Test F1: 0.414, Test AUC: 0.805
Decision Tree on Oversampled Train Data Fold 3  Test F1: 0.433, Test AUC: 0.818
Decision Tree on Oversampled Train Data Fold 4  Test F1: 0.408, Test AUC: 0.806
Decision Tree on Oversampled Train Data Fold 5  Test F1: 0.427, Test AUC: 0.819
Decision Tree on Oversampled Train Data; Mean F1: 0.422
Decision Tree on Oversampled Train Data; Mean AUC: 0.814
```

Random Forest

```
In [60]: # Test of Random Forest
         rfor = RandomForestClassifier(n_estimators = 100, max_features = 3, min_samples_leaf =
         rfor.fit(X_tr_os_list[0],y_tr_os_list[0])
         rfor_f1, rfor_auc = metrics.f1_score(y_valid_list[0], rfor.predict(X_valid_list[0])),
         print(rfor_f1)
         print(rfor_auc)
```

```
0.39229504345783417
0.8224829676777085
```

```
In [61]: # Random Forest cross validation function

         def random_forest_cross_val(x_train,y_train,x_val,y_val,n_estimators = 100, max_featur
             """Take training and validation data in folds and fit random forest and calculate
             f1_scores = []
             auc_scores = []
             for x in range(len(x_train)):
                 rfor_os = RandomForestClassifier(n_estimators=n_estimators,max_features=max_fe
                 rfor_os.fit(x_train[x], y_train[x])
                 rfor_f1, rfor_auc = metrics.f1_score(y_val[x], rfor_os.predict(x_val[x])), met
                 f1_scores.append(dtr_f1)
                 auc_scores.append(dtr_auc)
                 print('Random Forest on Oversampled Train Data Fold %.0f  Test F1: %.3f, Test
             print('Random Forest on Oversampled Train Data; Mean F1: %.3f' %np.mean(f1_scores)
             print('Random Forest on Oversampled Train Data; Mean AUC: %.3f' %np.mean(auc_score

         random_forest_cross_val(X_tr_os_list,y_tr_os_list,X_valid_list,y_valid_list)
```

```
Random Forest on Oversampled Train Data Fold 1  Test F1: 0.393, Test AUC: 0.823
Random Forest on Oversampled Train Data Fold 2  Test F1: 0.373, Test AUC: 0.808
Random Forest on Oversampled Train Data Fold 3  Test F1: 0.399, Test AUC: 0.819
Random Forest on Oversampled Train Data Fold 4  Test F1: 0.373, Test AUC: 0.808
Random Forest on Oversampled Train Data Fold 5  Test F1: 0.395, Test AUC: 0.819
Random Forest on Oversampled Train Data; Mean F1: 0.426
Random Forest on Oversampled Train Data; Mean AUC: 0.823
```

XGBoost

```
In [62]: gbm = xgb.XGBClassifier(
                                    n_estimators=1000, #arbitrary large number
                                    max_depth=3,
                                    objective="binary:logistic",
                                    learning_rate=.1,
                                    subsample=1,
                                    min_child_weight=1,
                                    colsample_bytree=.8
                                   )

         gbm.fit(X_valid_list[0], y_valid_list[0])

         metrics.f1_score(y_valid_list[0], gbm.predict(X_valid_list[0]))
```

```
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

```
Out[62]: 0.43112727272727275
```

```
In [63]: gbm = xgb.XGBClassifier(
                                    n_estimators=10000, #arbitrary large number
                                    max_depth=3,
                                    objective="binary:logistic",
                                    learning_rate=.1,
                                    subsample=1,
                                    min_child_weight=1,
                                    colsample_bytree=.8
                                   )

         eval_set=[(X_tr_os_list[0],y_tr_os_list[0]),(X_valid_list[0],y_valid_list[0])] #track
         fit_model = gbm.fit(
                              X_tr_os_list[0],y_tr_os_list[0],
                              eval_set=eval_set,
                              eval_metric='auc',
                              early_stopping_rounds=50,
                              verbose=True #gives output log as below
                             )
```

```
[0]        validation_0-auc:0.798679        validation_1-auc:0.80845
Multiple eval metrics have been passed: 'validation_1-auc' will be used for early stopping.

Will train until validation_1-auc hasn't improved in 50 rounds.
[1]        validation_0-auc:0.80929        validation_1-auc:0.81861
[2]        validation_0-auc:0.809741        validation_1-auc:0.81933
[3]        validation_0-auc:0.811828        validation_1-auc:0.821081
[4]        validation_0-auc:0.811447        validation_1-auc:0.820744
[5]        validation_0-auc:0.811806        validation_1-auc:0.821358
[6]        validation_0-auc:0.812495        validation_1-auc:0.822168
[7]        validation_0-auc:0.812753        validation_1-auc:0.822604
[8]        validation_0-auc:0.814048        validation_1-auc:0.823887
[9]        validation_0-auc:0.814881        validation_1-auc:0.824339
[10]        validation_0-auc:0.815164        validation_1-auc:0.824472
[11]        validation_0-auc:0.815865        validation_1-auc:0.824956
[12]        validation_0-auc:0.816197        validation_1-auc:0.825301
[13]        validation_0-auc:0.816782        validation_1-auc:0.82593
[14]        validation_0-auc:0.817132        validation_1-auc:0.826013
[15]        validation_0-auc:0.817392        validation_1-auc:0.826375
[16]        validation_0-auc:0.817614        validation_1-auc:0.826588
[17]        validation_0-auc:0.818094        validation_1-auc:0.827088
[18]        validation_0-auc:0.818405        validation_1-auc:0.827253
[19]        validation_0-auc:0.818748        validation_1-auc:0.827552
[20]        validation_0-auc:0.819194        validation_1-auc:0.827969
[21]        validation_0-auc:0.819458        validation_1-auc:0.828184
[22]        validation_0-auc:0.819797        validation_1-auc:0.828368
[23]        validation_0-auc:0.820052        validation_1-auc:0.828593
[24]        validation_0-auc:0.820334        validation_1-auc:0.828839
[25]        validation_0-auc:0.820684        validation_1-auc:0.82909
[26]        validation_0-auc:0.820927        validation_1-auc:0.829307
[27]        validation_0-auc:0.821085        validation_1-auc:0.829397
[28]        validation_0-auc:0.821235        validation_1-auc:0.829476
[29]        validation_0-auc:0.821452        validation_1-auc:0.829569
[30]        validation_0-auc:0.821645        validation_1-auc:0.829577
[31]        validation_0-auc:0.821772        validation_1-auc:0.829718
[32]        validation_0-auc:0.821932        validation_1-auc:0.829712
[33]        validation_0-auc:0.822051        validation_1-auc:0.829842
[34]        validation_0-auc:0.822162        validation_1-auc:0.82991
[35]        validation_0-auc:0.822243        validation_1-auc:0.829955
[36]        validation_0-auc:0.822459        validation_1-auc:0.830141
[37]        validation_0-auc:0.822682        validation_1-auc:0.830242
[38]        validation_0-auc:0.822841        validation_1-auc:0.830224
[39]        validation_0-auc:0.822983        validation_1-auc:0.830158
[40]        validation_0-auc:0.823195        validation_1-auc:0.830166
[41]        validation_0-auc:0.823357        validation_1-auc:0.830191
[42]        validation_0-auc:0.823466        validation_1-auc:0.830308
[43]        validation_0-auc:0.823641        validation_1-auc:0.830354
[44]        validation_0-auc:0.823823        validation_1-auc:0.830449
```

```
[45]    validation_0-auc:0.82392    validation_1-auc:0.830553
[46]    validation_0-auc:0.824127    validation_1-auc:0.830532
[47]    validation_0-auc:0.824283    validation_1-auc:0.830568
[48]    validation_0-auc:0.824386    validation_1-auc:0.830489
[49]    validation_0-auc:0.824513    validation_1-auc:0.830526
[50]    validation_0-auc:0.824643    validation_1-auc:0.830544
[51]    validation_0-auc:0.824725    validation_1-auc:0.83059
[52]    validation_0-auc:0.824811    validation_1-auc:0.830584
[53]    validation_0-auc:0.824894    validation_1-auc:0.830605
[54]    validation_0-auc:0.82501    validation_1-auc:0.830612
[55]    validation_0-auc:0.825183    validation_1-auc:0.830638
[56]    validation_0-auc:0.825291    validation_1-auc:0.830657
[57]    validation_0-auc:0.825433    validation_1-auc:0.830634
[58]    validation_0-auc:0.8255    validation_1-auc:0.830579
[59]    validation_0-auc:0.825602    validation_1-auc:0.830639
[60]    validation_0-auc:0.825673    validation_1-auc:0.830675
[61]    validation_0-auc:0.825785    validation_1-auc:0.83071
[62]    validation_0-auc:0.825969    validation_1-auc:0.830837
[63]    validation_0-auc:0.826085    validation_1-auc:0.830943
[64]    validation_0-auc:0.826145    validation_1-auc:0.830967
[65]    validation_0-auc:0.826231    validation_1-auc:0.830987
[66]    validation_0-auc:0.826305    validation_1-auc:0.831006
[67]    validation_0-auc:0.826408    validation_1-auc:0.831029
[68]    validation_0-auc:0.826495    validation_1-auc:0.83102
[69]    validation_0-auc:0.826533    validation_1-auc:0.831012
[70]    validation_0-auc:0.826639    validation_1-auc:0.83098
[71]    validation_0-auc:0.826725    validation_1-auc:0.830996
[72]    validation_0-auc:0.826762    validation_1-auc:0.830971
[73]    validation_0-auc:0.826859    validation_1-auc:0.831091
[74]    validation_0-auc:0.826941    validation_1-auc:0.831053
[75]    validation_0-auc:0.827044    validation_1-auc:0.831031
[76]    validation_0-auc:0.827123    validation_1-auc:0.831078
[77]    validation_0-auc:0.827223    validation_1-auc:0.8311
[78]    validation_0-auc:0.827305    validation_1-auc:0.831093
[79]    validation_0-auc:0.827383    validation_1-auc:0.831103
[80]    validation_0-auc:0.827478    validation_1-auc:0.831092
[81]    validation_0-auc:0.827568    validation_1-auc:0.831178
[82]    validation_0-auc:0.827651    validation_1-auc:0.831146
[83]    validation_0-auc:0.827668    validation_1-auc:0.831135
[84]    validation_0-auc:0.827738    validation_1-auc:0.831102
[85]    validation_0-auc:0.827804    validation_1-auc:0.8311
[86]    validation_0-auc:0.827857    validation_1-auc:0.831062
[87]    validation_0-auc:0.827898    validation_1-auc:0.831058
[88]    validation_0-auc:0.827974    validation_1-auc:0.831078
[89]    validation_0-auc:0.828057    validation_1-auc:0.831102
[90]    validation_0-auc:0.828102    validation_1-auc:0.831113
[91]    validation_0-auc:0.828138    validation_1-auc:0.83111
[92]    validation_0-auc:0.828217    validation_1-auc:0.831229
```

```
[93]	validation_0-auc:0.828321	validation_1-auc:0.831262
[94]	validation_0-auc:0.828372	validation_1-auc:0.831266
[95]	validation_0-auc:0.828458	validation_1-auc:0.831289
[96]	validation_0-auc:0.828531	validation_1-auc:0.831288
[97]	validation_0-auc:0.82864	validation_1-auc:0.831367
[98]	validation_0-auc:0.828703	validation_1-auc:0.831374
[99]	validation_0-auc:0.828725	validation_1-auc:0.831347
[100]	validation_0-auc:0.828818	validation_1-auc:0.831313
[101]	validation_0-auc:0.828885	validation_1-auc:0.831389
[102]	validation_0-auc:0.828978	validation_1-auc:0.831387
[103]	validation_0-auc:0.829035	validation_1-auc:0.831394
[104]	validation_0-auc:0.829097	validation_1-auc:0.831389
[105]	validation_0-auc:0.829177	validation_1-auc:0.831485
[106]	validation_0-auc:0.829294	validation_1-auc:0.831513
[107]	validation_0-auc:0.829353	validation_1-auc:0.831558
[108]	validation_0-auc:0.829377	validation_1-auc:0.83155
[109]	validation_0-auc:0.829414	validation_1-auc:0.831524
[110]	validation_0-auc:0.829465	validation_1-auc:0.831551
[111]	validation_0-auc:0.829555	validation_1-auc:0.831583
[112]	validation_0-auc:0.829592	validation_1-auc:0.831586
[113]	validation_0-auc:0.829682	validation_1-auc:0.83161
[114]	validation_0-auc:0.829753	validation_1-auc:0.831653
[115]	validation_0-auc:0.829814	validation_1-auc:0.83169
[116]	validation_0-auc:0.829855	validation_1-auc:0.831731
[117]	validation_0-auc:0.829929	validation_1-auc:0.831777
[118]	validation_0-auc:0.82997	validation_1-auc:0.831808
[119]	validation_0-auc:0.830033	validation_1-auc:0.83179
[120]	validation_0-auc:0.830098	validation_1-auc:0.831812
[121]	validation_0-auc:0.830161	validation_1-auc:0.831812
[122]	validation_0-auc:0.830234	validation_1-auc:0.83183
[123]	validation_0-auc:0.830295	validation_1-auc:0.831879
[124]	validation_0-auc:0.830362	validation_1-auc:0.831839
[125]	validation_0-auc:0.830387	validation_1-auc:0.831845
[126]	validation_0-auc:0.830469	validation_1-auc:0.831881
[127]	validation_0-auc:0.830547	validation_1-auc:0.831882
[128]	validation_0-auc:0.830586	validation_1-auc:0.831887
[129]	validation_0-auc:0.830647	validation_1-auc:0.831867
[130]	validation_0-auc:0.8307	validation_1-auc:0.831936
[131]	validation_0-auc:0.830767	validation_1-auc:0.83192
[132]	validation_0-auc:0.830826	validation_1-auc:0.831905
[133]	validation_0-auc:0.830881	validation_1-auc:0.831912
[134]	validation_0-auc:0.830929	validation_1-auc:0.831937
[135]	validation_0-auc:0.830997	validation_1-auc:0.831885
[136]	validation_0-auc:0.831085	validation_1-auc:0.831901
[137]	validation_0-auc:0.831152	validation_1-auc:0.831941
[138]	validation_0-auc:0.831182	validation_1-auc:0.831927
[139]	validation_0-auc:0.831255	validation_1-auc:0.831937
[140]	validation_0-auc:0.831291	validation_1-auc:0.831946
```

```
[141]	validation_0-auc:0.831324	validation_1-auc:0.831928
[142]	validation_0-auc:0.831338	validation_1-auc:0.83194
[143]	validation_0-auc:0.831419	validation_1-auc:0.831935
[144]	validation_0-auc:0.831482	validation_1-auc:0.831946
[145]	validation_0-auc:0.831524	validation_1-auc:0.831939
[146]	validation_0-auc:0.831602	validation_1-auc:0.831945
[147]	validation_0-auc:0.831682	validation_1-auc:0.831965
[148]	validation_0-auc:0.831752	validation_1-auc:0.831929
[149]	validation_0-auc:0.831834	validation_1-auc:0.832047
[150]	validation_0-auc:0.831904	validation_1-auc:0.832044
[151]	validation_0-auc:0.831933	validation_1-auc:0.832048
[152]	validation_0-auc:0.831992	validation_1-auc:0.832031
[153]	validation_0-auc:0.83201	validation_1-auc:0.832039
[154]	validation_0-auc:0.83206	validation_1-auc:0.832032
[155]	validation_0-auc:0.832116	validation_1-auc:0.832053
[156]	validation_0-auc:0.832163	validation_1-auc:0.832058
[157]	validation_0-auc:0.832214	validation_1-auc:0.832062
[158]	validation_0-auc:0.832285	validation_1-auc:0.832072
[159]	validation_0-auc:0.832306	validation_1-auc:0.832105
[160]	validation_0-auc:0.832397	validation_1-auc:0.832121
[161]	validation_0-auc:0.832434	validation_1-auc:0.832151
[162]	validation_0-auc:0.832497	validation_1-auc:0.832248
[163]	validation_0-auc:0.832582	validation_1-auc:0.832201
[164]	validation_0-auc:0.832628	validation_1-auc:0.832153
[165]	validation_0-auc:0.83274	validation_1-auc:0.832228
[166]	validation_0-auc:0.832797	validation_1-auc:0.832225
[167]	validation_0-auc:0.832862	validation_1-auc:0.832195
[168]	validation_0-auc:0.832929	validation_1-auc:0.832238
[169]	validation_0-auc:0.832991	validation_1-auc:0.832205
[170]	validation_0-auc:0.833023	validation_1-auc:0.832237
[171]	validation_0-auc:0.833064	validation_1-auc:0.832259
[172]	validation_0-auc:0.833123	validation_1-auc:0.832251
[173]	validation_0-auc:0.833166	validation_1-auc:0.832259
[174]	validation_0-auc:0.83325	validation_1-auc:0.832306
[175]	validation_0-auc:0.833301	validation_1-auc:0.832299
[176]	validation_0-auc:0.833317	validation_1-auc:0.832301
[177]	validation_0-auc:0.833378	validation_1-auc:0.832269
[178]	validation_0-auc:0.833416	validation_1-auc:0.832283
[179]	validation_0-auc:0.833465	validation_1-auc:0.832249
[180]	validation_0-auc:0.833488	validation_1-auc:0.83227
[181]	validation_0-auc:0.833562	validation_1-auc:0.832303
[182]	validation_0-auc:0.8336	validation_1-auc:0.832262
[183]	validation_0-auc:0.83367	validation_1-auc:0.832265
[184]	validation_0-auc:0.833745	validation_1-auc:0.832222
[185]	validation_0-auc:0.833805	validation_1-auc:0.832248
[186]	validation_0-auc:0.833858	validation_1-auc:0.832223
[187]	validation_0-auc:0.8339	validation_1-auc:0.832222
[188]	validation_0-auc:0.833956	validation_1-auc:0.832315
```

```
[189]	validation_0-auc:0.833973	validation_1-auc:0.832325
[190]	validation_0-auc:0.833999	validation_1-auc:0.832321
[191]	validation_0-auc:0.834087	validation_1-auc:0.832325
[192]	validation_0-auc:0.834165	validation_1-auc:0.832392
[193]	validation_0-auc:0.834201	validation_1-auc:0.83238
[194]	validation_0-auc:0.834265	validation_1-auc:0.832386
[195]	validation_0-auc:0.8343	validation_1-auc:0.832403
[196]	validation_0-auc:0.834363	validation_1-auc:0.832373
[197]	validation_0-auc:0.834418	validation_1-auc:0.83235
[198]	validation_0-auc:0.834452	validation_1-auc:0.832354
[199]	validation_0-auc:0.834478	validation_1-auc:0.832318
[200]	validation_0-auc:0.834492	validation_1-auc:0.832298
[201]	validation_0-auc:0.834534	validation_1-auc:0.832323
[202]	validation_0-auc:0.834605	validation_1-auc:0.83229
[203]	validation_0-auc:0.834684	validation_1-auc:0.832259
[204]	validation_0-auc:0.834718	validation_1-auc:0.832246
[205]	validation_0-auc:0.834738	validation_1-auc:0.832262
[206]	validation_0-auc:0.834797	validation_1-auc:0.832274
[207]	validation_0-auc:0.834862	validation_1-auc:0.832255
[208]	validation_0-auc:0.834895	validation_1-auc:0.832271
[209]	validation_0-auc:0.834973	validation_1-auc:0.832318
[210]	validation_0-auc:0.835011	validation_1-auc:0.832348
[211]	validation_0-auc:0.835066	validation_1-auc:0.832339
[212]	validation_0-auc:0.835113	validation_1-auc:0.832314
[213]	validation_0-auc:0.835189	validation_1-auc:0.832305
[214]	validation_0-auc:0.835258	validation_1-auc:0.832324
[215]	validation_0-auc:0.835309	validation_1-auc:0.832303
[216]	validation_0-auc:0.835363	validation_1-auc:0.832298
[217]	validation_0-auc:0.835449	validation_1-auc:0.832266
[218]	validation_0-auc:0.835485	validation_1-auc:0.832263
[219]	validation_0-auc:0.835543	validation_1-auc:0.832291
[220]	validation_0-auc:0.835562	validation_1-auc:0.832274
[221]	validation_0-auc:0.835587	validation_1-auc:0.832276
[222]	validation_0-auc:0.835629	validation_1-auc:0.832247
[223]	validation_0-auc:0.835682	validation_1-auc:0.83225
[224]	validation_0-auc:0.835708	validation_1-auc:0.83221
[225]	validation_0-auc:0.835731	validation_1-auc:0.832206
[226]	validation_0-auc:0.835782	validation_1-auc:0.832207
[227]	validation_0-auc:0.835845	validation_1-auc:0.832173
[228]	validation_0-auc:0.83591	validation_1-auc:0.832158
[229]	validation_0-auc:0.835947	validation_1-auc:0.832159
[230]	validation_0-auc:0.835981	validation_1-auc:0.832136
[231]	validation_0-auc:0.836035	validation_1-auc:0.832182
[232]	validation_0-auc:0.836069	validation_1-auc:0.832189
[233]	validation_0-auc:0.836143	validation_1-auc:0.8322
[234]	validation_0-auc:0.836195	validation_1-auc:0.832189
[235]	validation_0-auc:0.836231	validation_1-auc:0.832203
[236]	validation_0-auc:0.836288	validation_1-auc:0.832192
```

```
[237]	validation_0-auc:0.83636	validation_1-auc:0.832195
[238]	validation_0-auc:0.836405	validation_1-auc:0.832258
[239]	validation_0-auc:0.836478	validation_1-auc:0.832267
[240]	validation_0-auc:0.836499	validation_1-auc:0.832267
[241]	validation_0-auc:0.836514	validation_1-auc:0.832269
[242]	validation_0-auc:0.836582	validation_1-auc:0.83224
[243]	validation_0-auc:0.836622	validation_1-auc:0.832221
[244]	validation_0-auc:0.836659	validation_1-auc:0.832281
[245]	validation_0-auc:0.836725	validation_1-auc:0.832301
Stopping. Best iteration:
[195]	validation_0-auc:0.8343	validation_1-auc:0.832403
```

In [64]:
```
# XGBoost cross validation function

def xgboost_cross_val(x_train,y_train,x_val,y_val,n_estimators=10000,max_depth=3,learn
    """Take training and validation data in folds and fit XGBoost and calculate metri
    best_ntree_limit_list = []
    f1_scores = []
    auc_scores = []
    for x in range(len(x_train)):
        eval_set=[(X_tr_os_list[x],y_tr_os_list[x]),(X_valid_list[x],y_valid_list[x])]
        gbm = xgb.XGBClassifier(
                                n_estimators=n_estimators, #arbitrary large number
                                max_depth=max_depth,
                                objective="binary:logistic",
                                learning_rate=learning_rate,
                                subsample=subsample,
                                min_child_weight=min_child_weight,
                                colsample_bytree=.8
                               )
        fit_model = gbm.fit(
                            X_tr_os_list[x],y_tr_os_list[x],
                            eval_set=eval_set,
                            eval_metric='auc',
                            early_stopping_rounds=50,
                            verbose=False
                           )
        xgb_ntree = gbm.best_ntree_limit
        xgb_f1 = metrics.f1_score(y_val[x],gbm.predict(x_val[x], ntree_limit=gbm.best_
        xgb_auc = metrics.roc_auc_score(y_val[x],gbm.predict(x_val[x], ntree_limit=gbm
        best_ntree_limit_list.append(xgb_ntree)
        f1_scores.append(xgb_f1)
        auc_scores.append(xgb_auc)
        print('XGBoost on Oversampled Train Data Fold %.0f  Test nTree: %.0f, Test F1
    print('\nXGBoost on Oversampled Train Data; Mean Best nTree: %.0f' %round(np.mean
    print('XGBoost on Oversampled Train Data; Mean F1: %.3f' %np.mean(f1_scores))
```

```
            print('XGBoost on Oversampled Train Data; Mean AUC: %.3f' %np.mean(auc_scores))

        xgboost_cross_val(X_tr_os_list,y_tr_os_list,X_valid_list,y_valid_list)
```

```
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:

XGBoost on Oversampled Train Data Fold 1  Test nTree: 196, Test F1: 0.437, Test AUC: 0.706

/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:

XGBoost on Oversampled Train Data Fold 2  Test nTree: 307, Test F1: 0.418, Test AUC: 0.691

/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:

XGBoost on Oversampled Train Data Fold 3  Test nTree: 166, Test F1: 0.446, Test AUC: 0.706

/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:

XGBoost on Oversampled Train Data Fold 4  Test nTree: 176, Test F1: 0.419, Test AUC: 0.692

/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:

XGBoost on Oversampled Train Data Fold 5  Test nTree: 161, Test F1: 0.437, Test AUC: 0.706

XGBoost on Oversampled Train Data; Mean Best nTree: 201
XGBoost on Oversampled Train Data; Mean F1: 0.431
XGBoost on Oversampled Train Data; Mean AUC: 0.700

/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

1.5.4 XGBoost Parameter Tuning

In [65]:
```
# Setting up a manual gridsearch

# This is ugly, clean it up later!

max_depth_list = [3,5,7,9]

def xgboost_gridsearch(x_train,y_train,x_val,y_val,max_depth_list=max_depth_list):
    for max_dep in max_depth_list:
        print('\n\n\nGrid search with Max Depth: %.0f' %max_dep)
        xgboost_cross_val(x_train,y_train,x_val,y_val,max_depth=max_dep,learning_rate=

xgboost_gridsearch(X_tr_os_list,y_tr_os_list,X_valid_list,y_valid_list,max_depth_list)
```

```
Grid search with Max Depth: 3


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 1  Test nTree: 388, Test F1: 0.437, Test AUC: 0.706


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 2  Test nTree: 387, Test F1: 0.419, Test AUC: 0.692


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 3  Test nTree: 398, Test F1: 0.446, Test AUC: 0.706


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

```
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 4  Test nTree: 349, Test F1: 0.418, Test AUC: 0.691


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 5  Test nTree: 315, Test F1: 0.439, Test AUC: 0.707

XGBoost on Oversampled Train Data; Mean Best nTree: 367
XGBoost on Oversampled Train Data; Mean F1: 0.432
XGBoost on Oversampled Train Data; Mean AUC: 0.701



Grid search with Max Depth: 5


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 1  Test nTree: 255, Test F1: 0.437, Test AUC: 0.705


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 2  Test nTree: 139, Test F1: 0.420, Test AUC: 0.693


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 3  Test nTree: 359, Test F1: 0.447, Test AUC: 0.705
```

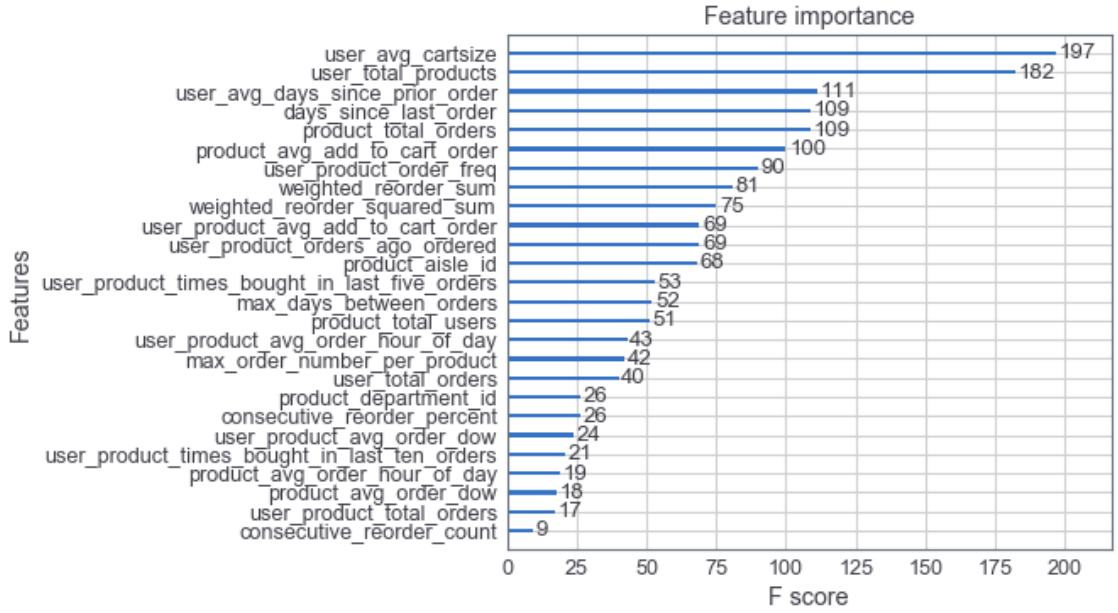
```
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 4  Test nTree: 107, Test F1: 0.421, Test AUC: 0.691


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 5  Test nTree: 210, Test F1: 0.437, Test AUC: 0.706

XGBoost on Oversampled Train Data; Mean Best nTree: 214
XGBoost on Oversampled Train Data; Mean F1: 0.433
XGBoost on Oversampled Train Data; Mean AUC: 0.700



Grid search with Max Depth: 7


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 1  Test nTree: 134, Test F1: 0.436, Test AUC: 0.702


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 2  Test nTree: 125, Test F1: 0.419, Test AUC: 0.690


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

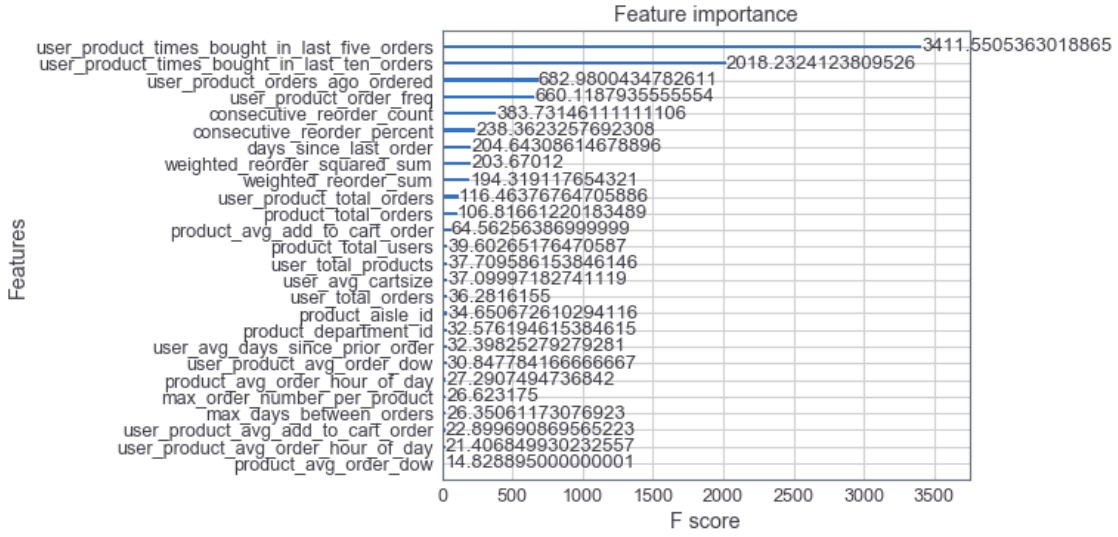
```
XGBoost on Oversampled Train Data Fold 3  Test nTree: 99, Test F1: 0.444, Test AUC: 0.703


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 4  Test nTree: 109, Test F1: 0.423, Test AUC: 0.690


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 5  Test nTree: 138, Test F1: 0.437, Test AUC: 0.702

XGBoost on Oversampled Train Data; Mean Best nTree: 121
XGBoost on Oversampled Train Data; Mean F1: 0.432
XGBoost on Oversampled Train Data; Mean AUC: 0.697



Grid search with Max Depth: 9


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 1  Test nTree: 80, Test F1: 0.434, Test AUC: 0.696


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 2  Test nTree: 72, Test F1: 0.415, Test AUC: 0.683


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

```
XGBoost on Oversampled Train Data Fold 3  Test nTree: 74, Test F1: 0.442, Test AUC: 0.695


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 4  Test nTree: 68, Test F1: 0.417, Test AUC: 0.683


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


XGBoost on Oversampled Train Data Fold 5  Test nTree: 91, Test F1: 0.436, Test AUC: 0.697

XGBoost on Oversampled Train Data; Mean Best nTree: 77
XGBoost on Oversampled Train Data; Mean F1: 0.429
XGBoost on Oversampled Train Data; Mean AUC: 0.691


/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

1.6 Model Evaluation

```
In [67]: metrics.f1_score(y_test,gbm.predict(X_test, ntree_limit=gbm.best_ntree_limit))

/Users/EdwardLeardi/anaconda/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:


Out[67]: 0.4242380366321581

In [68]: xgb.plot_importance(gbm);
             # Frequency, of all of the trees we trained, AveOccup was used 6526
         xgb.plot_importance(gbm, importance_type='gain');
             # Information gain calculation (go back to tree notebook)
```


Feature importance
user_avg_cartsize 197
user_total_products 182
user_avg_days_since_prior_order 111
days_since_last_order 109
product_total_orders 109
product_avg_add_to_cart_order 100
user_product_order_freq 90
weighted_reorder_sum 81
weighted_reorder_squared_sum 75
user_product_avg_add_to_cart_order 69
user_product_orders_ago_ordered 69
product_aisle_id 68
user_product_times_bought_in_last_five_orders 53
max_days_between_orders 52
product_total_users 51
user_product_avg_order_hour_of_day 43
max_order_number_per_product 42
user_total_orders 40
product_department_id 26
consecutive_reorder_percent 26
user_product_avg_order_dow 24
user_product_times_bought_in_last_ten_orders 21
product_avg_order_hour_of_day 19
product_avg_order_dow 18
user_product_total_orders 17
consecutive_reorder_count 9
Features
0 25 50 75 100 125 150 175 200
F score


Feature importance
user_product_times_bought_in_last_five_orders 3411.5505363018865
user_product_times_bought_in_last_ten_orders 2018.2324123809526
user_product_orders_ago_ordered 682.9800434782611
user_product_order_freq 660.1187935555554
consecutive_reorder_count 383.73146111111106
consecutive_reorder_percent 238.3623257692308
days_since_last_order 204.64308614678896
weighted_reorder_squared_sum 203.67012
weighted_reorder_sum 194.319117654321
user_product_total_orders 116.46376764705886
product_total_orders 106.81661220183489
product_avg_add_to_cart_order 64.56256386999999
product_total_users 39.60265176470587
user_total_products 37.709586153846146
user_avg_cartsize 37.09997182741119
user_total_orders 36.2816155
product_aisle_id 34.650672610294116
product_department_id 32.576194615384615
user_avg_days_since_prior_order 32.39825279279281
user_product_avg_order_dow 30.847784166666667
product_avg_order_hour_of_day 27.2907494736842
max_order_number_per_product 26.623175
max_days_between_orders 26.35061173076923
user_product_avg_add_to_cart_order 22.899690869565223
user_product_avg_order_hour_of_day 21.406849930232557
product_avg_order_dow 14.828895000000001
Features
0 500 1000 1500 2000 2500 3000 3500
F score