Music_Store_App_User_Churn_Prediction

October 30, 2018

1 Introduction

In this report I present findings from an exploration of a Music Store App user dataset which records users behavior including playing , searching and downloading songs using the app. Of primary interest is user churn prediction, that means detecting which customers are likely to cancel a subscription to a service based on how they use it [1]. Typically, the customer churn is calculated as a relative number in percentage (i.e. the churn rate) [2]. There are several ways to calculate the churn rate. It is usually expressed as follows: * Fix a conventional period of time as a month or a year; * Count the number of customers lost in this period; * Divide this quantity by the number of customers that the firm had at the beginning of this period.

1.1 Dataset

The source data was provided by a star-up company in China, which includes play log, download log, and search log files from 3/30/2017 - 5/12/2017. Due to confidentiality, there is no user profile available. The data covers about 0.6 million users and 2 million songs.

2 Date preprocessing

Due to the large size of the data (~10GB), the files were first unpacked, and the log files were preprocessed using shell script. In addition, since the log files are recorded by each day, I also combined seperate log files to one file containing all the records between 3/30/2017 - 5/12/2017.

```
In [ ]: #### process play log files ####
        # unzip play log
        cd ../data/raw/
        for f in *_play.log.tar.gz
        do
         echo "Processing $f"
         tar -xvzf $f
        done

        mv *_play.log ../play/

        # append file_name to each row (date is added to the dataset)
        cd ../play/
```

```
for f in *.log
do
 echo "Processing $f"
 awk -v var="$f" '{print $0,"\t",substr(var,1,8)}' $f > ${f}.fn
done

# cat all log with filename to one file
cat *.log.fn > all_play_log
rm *.log
rm *.log.fn



#### process down log files ####
# unzip down log
cd ../data/raw/
for f in *_down.log.tar.gz
do
 echo "Processing $f"
 tar -xvzf $f && mv *_down.log  ../down/${f//".tar.gz"/""}
done

# append file_name to each row (date is added to the dataset)
cd ../down/
for f in *.log
do
 echo "Processing $f"
 awk -v var="$f" '{print $0,"\t",substr(var,1,8)}' $f > ${f}.fn
done

# cat all log with filename to one file
cat *.log.fn > all_down_log
rm *.log
rm *.log.fn



#### process search log files ####
# unzip search log
cd ../data/raw/
for f in *_search.log.tar.gz
do
 echo "Processing $f"
 tar -xvzf $f && mv *_search.log  ../search/${f//".tar.gz"/""}
done

# append file_name to each row (date is added to the dataset)
cd ../search/
```

```
for f in *.log
do
 echo "Processing $f"
 awk -v var="$f" '{print $0,"\t",substr(var,1,8)}' $f > ${f}.fn
done

# cat all log with filename to one file
cat *.log.fn > all_search_log
rm *.log
rm *.log.fn
```

2.1 Count unique ID

In [1]:
```
# use shell script to count plays group by each user id
import os
cmd="""
export LC_CTYPE=C
export LANG=C
# get uid field| sort | count unique ids | strip blank spaces | output to file
cat /Users/xuanou/Desktop/data/play/all_play_log| cut -f1 -d$'\t'| sort | uniq -c | sed
"""
os.system(cmd)
```

Out[1]: 0

In [18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import date2num
import seaborn as sns
from IPython.display import Image

%matplotlib inline
plt.style.use('ggplot')
```

I first counted how many times each distinct user played music in the play dataset, which is the largest dataset among three of those and found that there are 594734 users played music during that period of time. I also noted that there is 1 user id missing.

In [3]:
```
# read play counts data by each user ID
df = pd.read_csv('/Users/xuanou/Desktop/data/uid_count.csv',sep='\s+', names=['count',
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 594735 entries, 0 to 594734
Data columns (total 2 columns):
count     594735 non-null int64
uid       594734 non-null float64
dtypes: float64(1), int64(1)
```

```
memory usage: 9.1 MB
```

```
In [4]: df.describe()
```

```
Out[4]:            count           uid
        count  5.947350e+05  5.947340e+05
        mean   2.460525e+02  1.673628e+08
        std    1.526662e+04  1.047142e+07
        min    1.000000e+00  0.000000e+00
        25%    9.000000e+00  1.680262e+08
        50%    4.000000e+01  1.684782e+08
        75%    1.740000e+02  1.687685e+08
        max    7.501794e+06  1.692623e+08
```

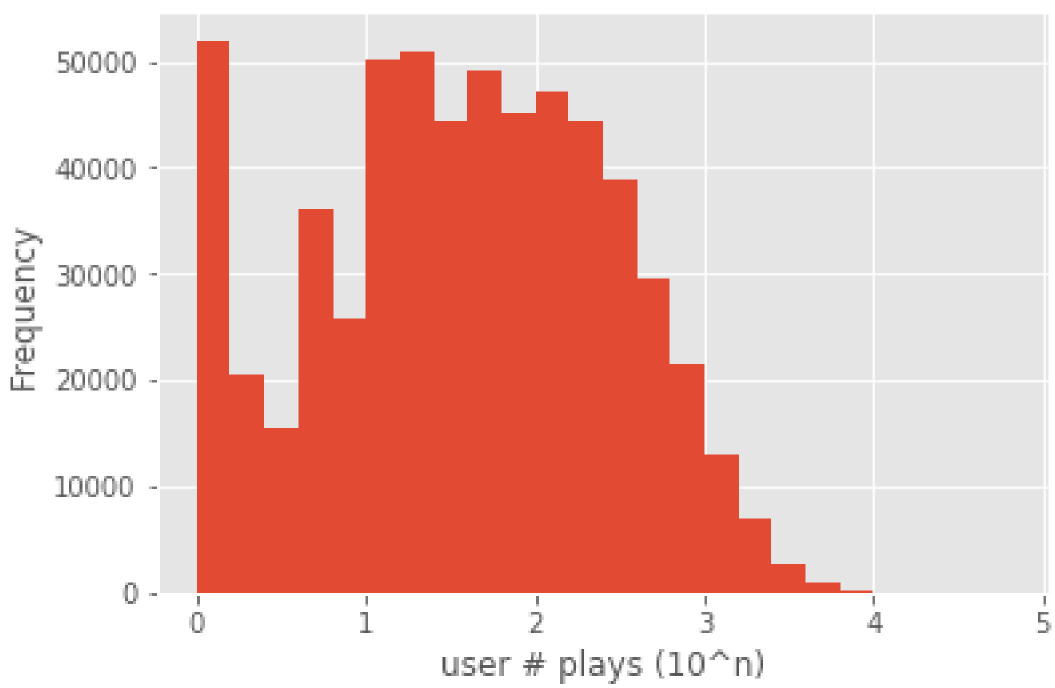
The above summary statistics shows us that 50% users played music less than 40 times between 3/30/2017 - 5/12/2017 (44 days) , which makes sense. I'd like to check whether the maximum of play counts is resonable or not.

2.2 Remove bots and outliers

Due to large variation between play times, I transformed the data by taking the log of the play counts and made a plot of log(play_counts)

```
In [5]: # make a plot of log(play_counts)
        np.log10(df['count']).plot.hist(bins=np.arange(0,5,0.2))
        plt.xlabel("user # plays (10^n)")
```

```
Out[5]: Text(0.5,0,'user # plays (10^n)')
```



```
In [7]: # 99.9% percentile of play_counts

        top_count_threshold = np.percentile(df['count'],99.9)
        print(top_count_threshold)
```

```
5195.394000000553
```

```
In [8]: # remove bots: get id with play counts < top_count_threshold
        id_list_bot_removed = np.array(df['uid'][df['count']<top_count_threshold].dropna())

        print("total number of users after bot removed:",len(id_list_bot_removed))
```

```
total number of users after bot removed: 594139
```

We removed 596 records (594735 - 594139 = 596) including both outliers and one missing value.

2.3 Apply downsmaple on uid level

The whole dataset is too large to handle in personal laptop, therefore I decided to downsampling the data at user level.

```
In [10]: # downsample ids
         np.random.seed = 1
         down_sample_ratio = 0.1
         id_subset = set(id_list_bot_removed[np.random.random(id_list_bot_removed.shape)<down_s

         print("total number of users after down sample:",len(id_subset))
```

```
total number of users after down sample: 59107
```

After downsampling, we have 59107 distinct user ids.

```
In [11]: # define date conversion function
         import datetime
         def convert_date(s):
             s = str(s).strip()  #leading spaces are removed
             try:
                 year = int(s[:4])
                 month = int(s[4:6])
                 day = int(s[6:8])
                 return datetime.date(year,month,day)
             except:
                 return None
```

```
In [13]: # downsample search dataset by uid
         import csv
         input_file = '/Users/xuanou/Desktop/data/search/all_search_log'
         output_file = '/Users/xuanou/Desktop/data/search_ds.csv'
         input_field_list = ['uid','device','time_stamp','search_query','date']
         output_field_list = ['uid','device','date']
         i=0
         with open(input_file,'r',encoding='latin-1') as fin, open(output_file,'w') as fout:
             csvin = csv.DictReader(fin,delimiter='\t',fieldnames=input_field_list,quoting=csv
             csvout = csv.writer(fout,delimiter=',')
             csvout.writerow(output_field_list) # write header
             for row in csvin:
                 i+=1
                 if i%1000000==0:
                     print("#row processed:",i)
                 try:
                     int(row['uid'])
                 except:
                     continue
                 if int(row['uid']) in id_subset:
                     row['date'] = convert_date(row['date'])
                     if row['date'] != None:
                         csvout.writerow([str(row[key]).strip() for key in output_field_list])
```

```
#row processed: 1000000
#row processed: 2000000
#row processed: 3000000
#row processed: 4000000
#row processed: 5000000
#row processed: 6000000
#row processed: 7000000
#row processed: 8000000
```

```
In [14]: # downsample download dataset by uid
         import csv
         input_file = '/Users/xuanou/Desktop/data/down/all_down_log'
         output_file = '/Users/xuanou/Desktop/data/down_ds.csv'
         input_field_list = ['uid','device','song_id','song_name','singer','paid_flag','date']
         output_field_list = ['uid','device','song_id','date']
         i=0
         with open(input_file,'r',encoding='latin-1') as fin, open(output_file,'w') as fout:
             csvin = csv.DictReader(fin,delimiter='\t',fieldnames=input_field_list,quoting=csv
             csvout = csv.writer(fout,delimiter=',')
             csvout.writerow(output_field_list) # write header
             for row in csvin:
                 i+=1
                 if i%1000000==0:
```

```
                    print("#row processed:",i)
                try:
                    int(row['uid'])
                except:
                    continue
                if int(row['uid']) in id_subset:
                    row['date'] = convert_date(row['date'])
                    if row['date'] != None:
                        csvout.writerow([str(row[key]).strip() for key in output_field_list])
```

```
#row processed: 1000000
#row processed: 2000000
#row processed: 3000000
#row processed: 4000000
#row processed: 5000000
#row processed: 6000000
#row processed: 7000000
```

In [15]:
```
# Take a quick look at three datasets after down-sampling
df_play = pd.read_csv('/Users/xuanou/Desktop/data/play_ds.csv')
df_play.info()
```

```
/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785: DtypeWarning: Co
  interactivity=interactivity, compiler=compiler, result=result)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10745731 entries, 0 to 10745730
Data columns (total 6 columns):
uid            int64
device         object
song_id        object
date           object
play_time      object
song_length    float64
dtypes: float64(1), int64(1), object(4)
memory usage: 491.9+ MB
```

In [17]:
```
df_play.head()
```

Out[17]:

	uid	device	song_id	date	play_time	song_length
0	168540348	ar	77260	2017-03-30	64528	0.0
1	168547857	ar	4.3563e+06	2017-03-30	3	0.0
2	168548101	ip	6.91318e+06	2017-03-30	40	198.0
3	168551487	ar	811133	2017-03-30	200	200.0
4	168532776	ip	2.06741e+07	2017-03-30	172	172.0

```
In [18]: df_search = pd.read_csv('/Users/xuanou/Desktop/data/search_ds.csv')
         df_search.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 763366 entries, 0 to 763365
Data columns (total 3 columns):
uid       763366 non-null int64
device    763366 non-null object
date      763366 non-null object
dtypes: int64(1), object(2)
memory usage: 17.5+ MB
```

```
In [27]: df_search.head()
```

Out[27]:

	uid	device	date
0	168040163	ar	2017-03-30
1	168045723	ar	2017-03-30
2	167780192	ar	2017-03-30
3	168045723	ar	2017-03-30
4	168021965	ar	2017-03-30

```
In [19]: df_down = pd.read_csv('/Users/xuanou/Desktop/data/down_ds.csv')
         df_down.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 623240 entries, 0 to 623239
Data columns (total 4 columns):
uid        623240 non-null int64
device     623240 non-null object
song_id    623169 non-null float64
date       623240 non-null object
dtypes: float64(1), int64(1), object(2)
memory usage: 19.0+ MB
```

```
In [28]: df_down.head()
```

Out[28]:

	uid	device	song_id	date
0	167580727	ip	6298828.0	2017-03-30
1	167819030	ip	365492.0	2017-03-30
2	167819030	ip	6854227.0	2017-03-30
3	167819030	ip	3626250.0	2017-03-30
4	167819030	ip	914553.0	2017-03-30

2.4 Create event table for feature generation

After getting three datasets after downsampling, I combined play, search and download datasets into one event table in order to generate new features.

In [20]:
```
play_file = '/Users/xuanou/Desktop/data/play_ds.csv'
down_file = '/Users/xuanou/Desktop/data/down_ds.csv'
search_file = '/Users/xuanou/Desktop/data/search_ds.csv'
output_file = '/Users/xuanou/Desktop/data/event_ds.csv'
play_field_list = ['uid','device','song_id','date','play_time','song_length']
down_field_list = ['uid','device','song_id','date']
search_field_list = ['uid','device','date']
output_field_list = ['uid','event','song_id','date']
with open(play_file,'r') as f_play, open(down_file,'r') as f_down, \
open(search_file,'r') as f_search,open(output_file,'w') as f_out:
    csvplay = csv.DictReader(f_play,delimiter=',')
    csvdown = csv.DictReader(f_down,delimiter=',')
    csvsearch = csv.DictReader(f_search,delimiter=',')
    csvout = csv.writer(f_out,delimiter=',')
    csvout.writerow(output_field_list) # write header
    print('Processing play ...')
    for row in csvplay:
        row['event'] = 'P'
        row['date']
        csvout.writerow([row[key] for key in output_field_list])
    print('Processing down ...')
    for row in csvdown:
        row['event'] = 'D'
        csvout.writerow([row[key] for key in output_field_list])
    print('Processing search ...')
    for row in csvsearch:
        row['event'] = 'S'
        csvout.writerow([row.get(key,'') for key in output_field_list])
```

```
Processing play ...
Processing down ...
Processing search ...
```

In [1]:
```
import pyspark.sql.functions as F
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
sc = SparkContext('local')
spark = SparkSession(sc)
```

In [63]:
```
# Load data into Spark DataFrame
df = spark.read.csv('/Users/xuanou/Desktop/data/event_ds.csv',header=True).cache()
df
```

Out[63]:
```
DataFrame[uid: string, event: string, song_id: string, date: string]
```

In [64]:
```
df.show(10)
```

```
+---------+-----+--------+----------+
|      uid|event| song_id|      date|
```

```
+---------+-----+--------+----------+
|168540348|    P|   77260|2017-03-30|
|168547857|    P| 4356304|2017-03-30|
|168548101|    P| 6913185|2017-03-30|
|168551487|    P|  811133|2017-03-30|
|168532776|    P|20674091|2017-03-30|
|168548099|    P| 4984002|2017-03-30|
|168543049|    P|  347730|2017-03-30|
|168550576|    P|  324249|2017-03-30|
|168551383|    P| 7149583|2017-03-30|
|168543348|    P|       0|2017-03-30|
+---------+-----+--------+----------+
only showing top 10 rows
```

3 Data Exploration

In [65]:
```
# create new or overwrite original field with withColumn
df = df.withColumn('date',F.col('date').cast('date'))
df
```

Out[65]: DataFrame[uid: string, event: string, song_id: string, date: date]

In [25]:
```
df.count()
```

Out[25]: 12132337

In [26]:
```
# select operation, count distinct rows
df.select('uid').distinct().count()
```

Out[26]: 59106

In [29]:
```
# group by aggregation
df.groupBy('event').count().show()
```

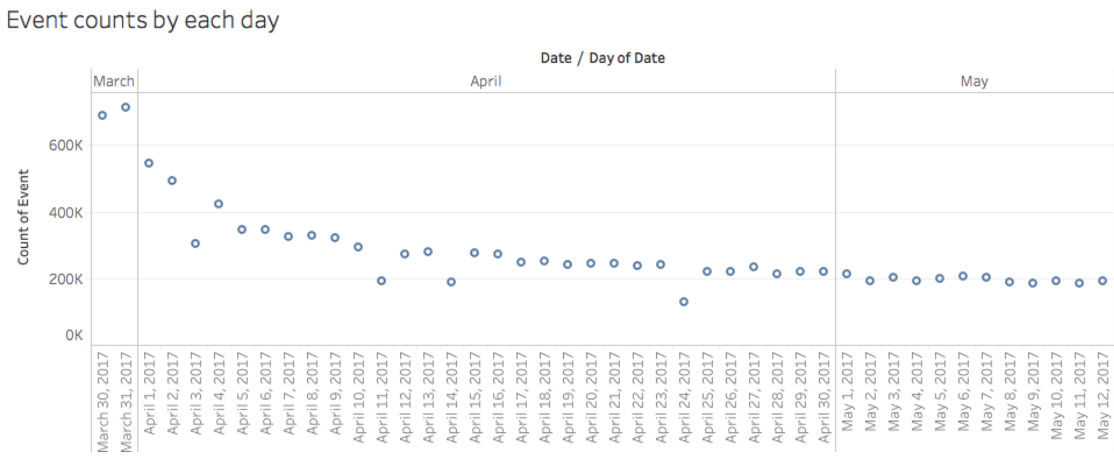
```
+-----+--------+
|event|   count|
+-----+--------+
|    D|  623240|
|    S|  763366|
|    P|10745731|
+-----+--------+
```

In [68]:
```
date_count = df.groupBy('date').count().toPandas()
date_count['date'] = date_count['date'].apply(date2num)
```

Let's take a look the trend when people like to use the app by each day and weekday.

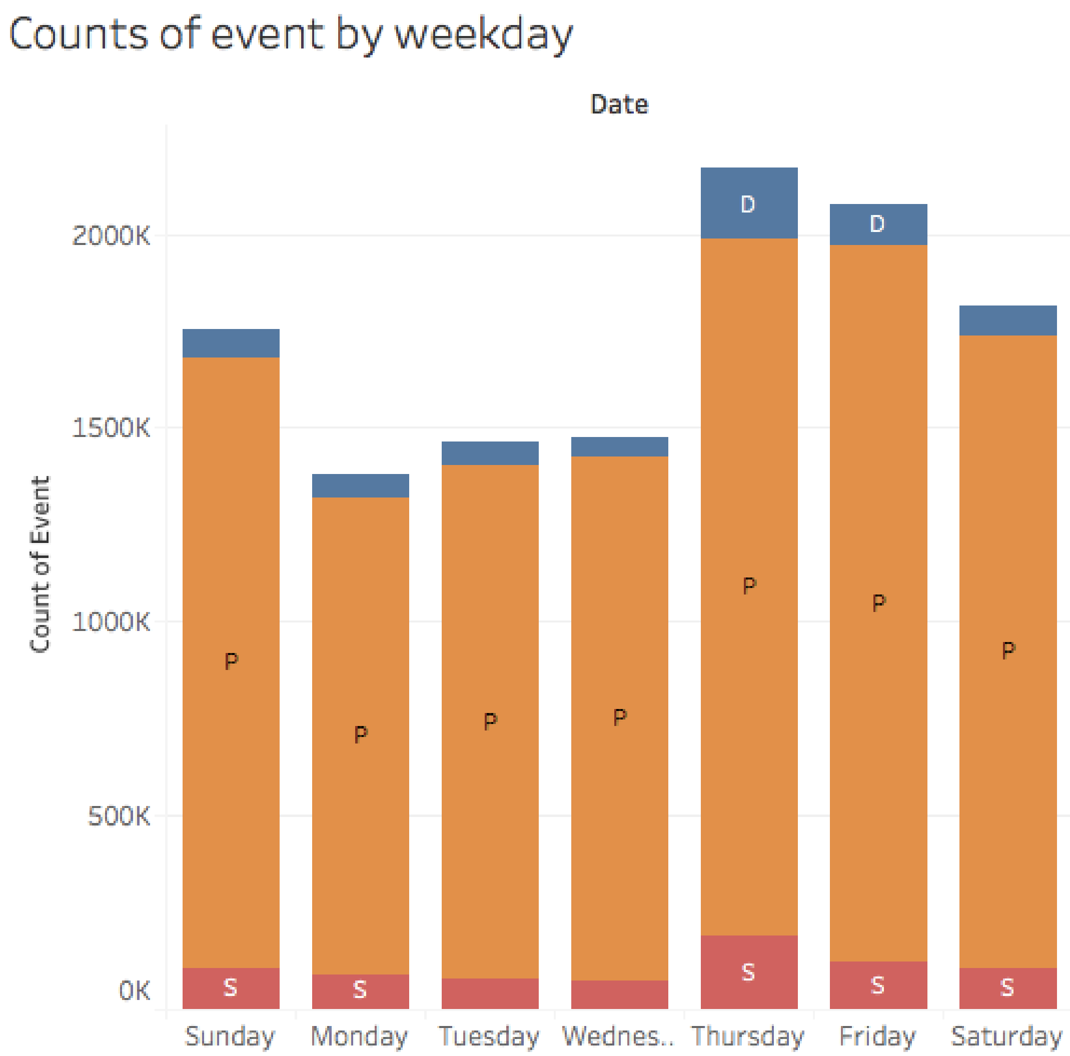
```
In [1]: Image(filename='/Users/xuanou/Desktop/data/Counts_eachday.png')
```

Out[1]:




```
In [2]: Image(filename='/Users/xuanou/Desktop/data/Counts_weekday.png')
```

Out[2]:



In the first week, the active users were at least 2X times more than the rest of days. It needed to investigate to get more insights. From the below barchart, it shows us that users used the app more frequently on Thursday, Friday, and the weekends then Monday, Tuesday, and Wednesday.

3.1 Churn Label

Here I defined user churn if users were inactive in outcome window of last two weeks (4/29-5/12). Then the population included all active users between 3/30-4/28, and excluded inactive users during the observation window.

```
In [10]: # Create label window (2017-04-29 ~ 2017-05-12) and feature windown (2017-03-30 ~ 201
         import datetime
         from dateutil import parser

         label_window_size = 14
```

```
label_window_end_date = parser.parse('2017-05-12').date()
label_window_start_date = label_window_end_date - datetime.timedelta(label_window_size
print('label window:',label_window_start_date,'~',label_window_end_date,'days:',label_

feature_window_size = 30
feature_window_end_date = label_window_start_date - datetime.timedelta(1)
feature_window_start_date = feature_window_end_date  - datetime.timedelta(feature_winc
print('feature window:',feature_window_start_date,'~',feature_window_end_date,'days:'
```

```
label window: 2017-04-29 ~ 2017-05-12 days: 14
feature window: 2017-03-30 ~ 2017-04-28 days: 30
```

In [11]:
```
# Include all the users in feature window
df_model_uid = df.filter((F.col('date')>=feature_window_start_date) & (F.col('date')<=
                    .select('uid').distinct()
# active in label window (active label=0)
df_active_uid_in_label_window = df.filter((F.col('date')>=label_window_start_date) &
                            .select('uid').distinct().withColumn('label',F.lit(0))
```

If a user existed in feature window but left in label window, that user was defined as ’churner’, and labeled as 1, otherwise the user was labeled as 0.

In [12]:
```
# prepare label data (churn label=1; active label=0)
df_label = df_model_uid.join(df_active_uid_in_label_window,on=['uid'],how='left')
df_label = df_label.fillna(1)
```

In [13]:
```
df_label.show(5)
```

```
+---------+-----+
|      uid|label|
+---------+-----+
|136858556|    1|
|159183409|    0|
|166855134|    0|
|167581827|    1|
|167590080|    1|
+---------+-----+
only showing top 5 rows
```

In [55]:
```
df_label.groupBy('label').count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|    1|35847|
|    0|21939|
+-----+-----+
```

3.2 Feature generation

15 features were generated consisting of frequencies of events over observation window. * Number of days: 1, 3, 7, 14, 30 * Type of events: play, search, download * 15 features

```
In [7]: df.show(5)
```

```
+---------+-----+--------+----------+
|      uid|event| song_id|      date|
+---------+-----+--------+----------+
|168540348|    P|   77260|2017-03-30|
|168547857|    P| 4356304|2017-03-30|
|168548101|    P| 6913185|2017-03-30|
|168551487|    P|  811133|2017-03-30|
|168532776|    P|20674091|2017-03-30|
+---------+-----+--------+----------+
only showing top 5 rows
```

```
In [14]: # event_data in feature_window
         df_feature_window = df.filter((F.col('date')>=feature_window_start_date) & (F.col('da
```

3.2.1 Frequency features generation

```
In [15]: # define a function to generate frequency features for a list of time windows
         # using when().otherwise(), and list comprehension trick!
         def frequency_feature_generation_time_windows(df,event,time_window_list,snapshot_date
             """
             generate frequency features for one event type and a list of time windows
             """
             df_feature = df \
                 .filter(F.col('event')==event) \
                 .groupBy('uid') \
                 .agg(*[F.sum(F.when((F.col('date')>=snapshot_date-datetime.timedelta(time_win
                         for time_window in time_window_list]
                     )# *[] opens list and make them comma separated
             return df_feature
```

```
In [16]: # generate one event type, all time windows
         event = 'S'
         time_window_list = [1,3,7,14,30]
         snapshot_date = feature_window_end_date
         df_feature = frequency_feature_generation_time_windows(df_feature_window,event,time_wi
         df_feature.show(5)
```

```
+---------+-------------+-------------+-------------+--------------+--------------+
|      uid|freq_S_last_1|freq_S_last_3|freq_S_last_7|freq_S_last_14|freq_S_last_30|
+---------+-------------+-------------+-------------+--------------+--------------+
```

```
|167718831|           8|           8|          14|           29|          129|
|167810312|           0|           0|           6|            9|           19|
|167935507|           6|          10|          33|           91|          222|
|167878077|           0|           0|           0|            0|           15|
|167787875|           0|           0|           0|            2|           21|
+---------+------------+------------+------------+-------------+-------------+
only showing top 5 rows
```

```
In [23]: # generate frequency features for all event_list, time_window_list
         event_list = ['P','D','S']
         time_window_list = [1,3,7,14,30]
         df_feature_list = []
         for event in event_list:
             df_feature_list.append(frequency_feature_generation_time_windows(df_feature_windo
```

```
In [ ]: df_feature_list
```

3.2.2 Profile features

```
In [27]: df_play = spark.read.csv('/Users/xuanou/Desktop/data/play_ds.csv',header=True)
         df_play.show(5)
```

```
+---------+------+--------+----------+---------+-----------+
|      uid|device| song_id|      date|play_time|song_length|
+---------+------+--------+----------+---------+-----------+
|168540348|    ar|   77260|2017-03-30|    64528|          0|
|168547857|    ar| 4356304|2017-03-30|        3|          0|
|168548101|    ip| 6913185|2017-03-30|       40|        198|
|168551487|    ar|  811133|2017-03-30|      200|        200|
|168532776|    ip|20674091|2017-03-30|      172|        172|
+---------+------+--------+----------+---------+-----------+
only showing top 5 rows
```

```
In [29]: df_play_feature_window = df_play.filter((F.col('date')>=feature_window_start_date) &
         df_profile_tmp = df_play_feature_window.select('uid','device').distinct()
```

```
In [30]: df_profile_tmp = df_profile_tmp.withColumn('device_type',F.when(F.col('device')=='ip'
         df_profile_tmp.groupBy('device_type').count().show()
```

```
+-----------+-----+
|device_type|count|
+-----------+-----+
|          1| 7297|
|          0|50373|
+-----------+-----+
```

```
In [79]: df_profile = df_label.select('uid').join(df_profile_tmp.select('uid','device_type'),o
```

3.3 Create final training data

```
In [39]: def join_feature_data(df_master,df_feature_list):
             for df_feature in df_feature_list:
                 df_master = df_master.join(df_feature,on='uid',how='left')
                 #df_master.persist() # uncomment if number of joins is too many
             return df_master
```

```
In [40]: # join all behavior features
         df_model_final = join_feature_data(df_label,df_feature_list)
```

```
In [41]: # join all profile features
         df_model_final = join_feature_data(df_model_final,[df_profile])
```

```
In [42]: df_model_final.fillna(0).toPandas().to_csv('/Users/xuanou/Desktop/data/df_model_final
```

4 Train Model

```
In [19]: # Load data from file
         df = pd.read_csv('/Users/xuanou/Desktop/data/df_model_final.csv')
```

```
In [4]: df.head(10)
```

```
Out[4]:          uid  label  freq_P_last_1  freq_P_last_3  freq_P_last_7  \
        0  136858556      1              0              0              0
        1  159183409      0              0              0              1
        2  166855134      0              8              8             90
        3  167581827      1              0              0              0
        4  167590080      1              0              0              0
        5  167594294      1              0              0              0
        6  167598799      1              0              0              0
        7  167611704      1              0              0              0
        8  167625105      1              0              0              0
        9  167631789      1              0              0              0

           freq_P_last_14  freq_P_last_30  freq_D_last_1  freq_D_last_3  \
        0               0               3              0              0
        1              79             774              0              0
        2             159             231              0              0
        3              15             163              0              0
        4              97             257              0              0
        5               0              17              0              0
        6               0              13              0              0
        7               2               6              0              0
        8               0              11              0              0
        9               0              17              0              0
```

```
   freq_D_last_7  freq_D_last_14  freq_D_last_30  freq_S_last_1  \
0              0               0               0              0
1              0               0               1              0
2              0               0               2              2
3              0               0               0              0
4              0               0               0              0
5              0               0               0              0
6              0               0               0              0
7              0               0               0              0
8              0               0               0              0
9              0               0               0              0

   freq_S_last_3  freq_S_last_7  freq_S_last_14  freq_S_last_30  device_type
0              0              0               0               0            0
1              0              2               2               9            0
2              2             13              23              32            0
3              0              0               0               0            0
4              0              0               0               0            0
5              0              0               0               0            0
6              0              0               0               0            0
7              0              0               0               0            1
8              0              0               0               0            0
9              0              0               0               0            0
```

In [5]:
```
df.isnull().sum().sum()
```

Out[5]:
```
0
```

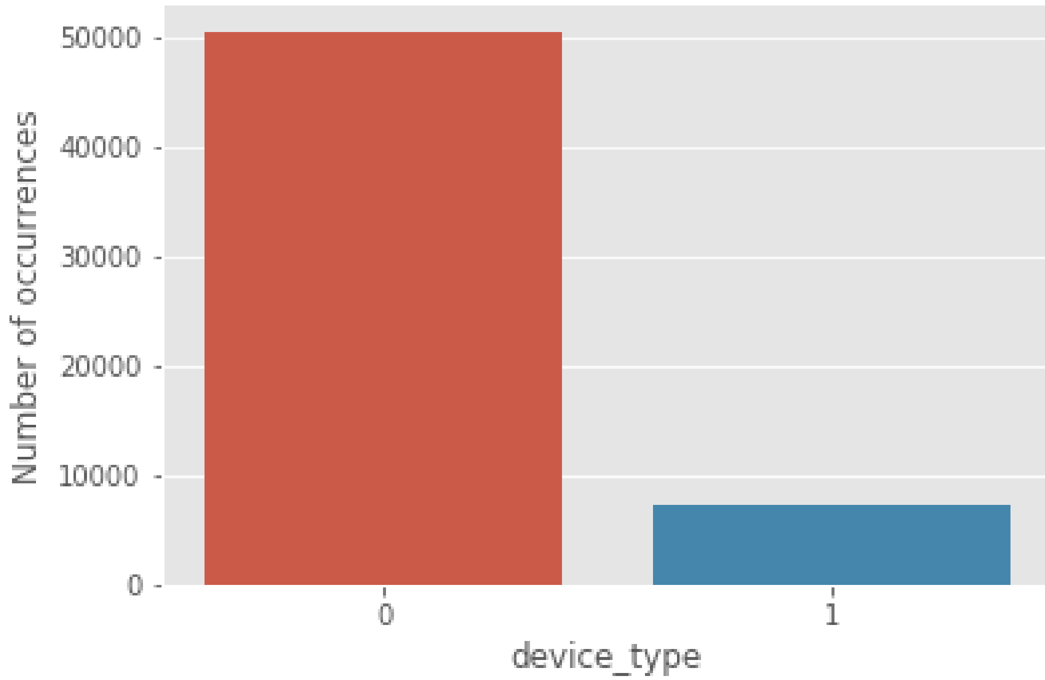
In [47]:
```
# Show summary stats
df.describe()
```

Out[47]:
```
                uid         label  freq_P_last_1  freq_P_last_3  \
count  5.780400e+04  57804.000000   57804.000000   57804.000000
mean   1.674472e+08      0.620355       3.402636      10.846845
std    1.008130e+07      0.485303      16.539069      42.379305
min    1.039280e+05      0.000000       0.000000       0.000000
25%    1.680336e+08      0.000000       0.000000       0.000000
50%    1.684944e+08      1.000000       0.000000       0.000000
75%    1.687749e+08      1.000000       0.000000       0.000000
max    1.692432e+08      1.000000    1012.000000    2411.000000

       freq_P_last_7  freq_P_last_14  freq_P_last_30  freq_D_last_1  \
count   57804.000000    57804.000000    57804.000000   57804.000000
mean       23.896651       51.878642      141.916788       0.144402
std        76.739165      144.055433      294.326629       4.466437
min         0.000000        0.000000        0.000000       0.000000
25%         0.000000        0.000000        8.000000       0.000000
50%         0.000000        0.000000       35.000000       0.000000
```

```
75%         9.000000      33.000000     140.000000      0.000000
max      2791.000000    3046.000000    4558.000000    642.000000

       freq_D_last_3  freq_D_last_7  freq_D_last_14  freq_D_last_30  \
count   57804.000000   57804.000000    57804.000000    57804.000000
mean        0.394021       0.912324        2.006643        9.184070
std         7.607693      12.167230       21.350712       56.329045
min         0.000000       0.000000        0.000000        0.000000
25%         0.000000       0.000000        0.000000        0.000000
50%         0.000000       0.000000        0.000000        0.000000
75%         0.000000       0.000000        0.000000        3.000000
max       670.000000    1119.000000     1932.000000     6427.000000

       freq_S_last_1  freq_S_last_3  freq_S_last_7  freq_S_last_14  \
count   57804.000000   57804.000000   57804.000000    57804.000000
mean        0.140181       0.338783       1.141011        2.818767
std         1.402440       2.275486       5.662774       11.047602
min         0.000000       0.000000       0.000000        0.000000
25%         0.000000       0.000000       0.000000        0.000000
50%         0.000000       0.000000       0.000000        0.000000
75%         0.000000       0.000000       0.000000        1.000000
max       101.000000     173.000000     383.000000      578.000000

       freq_S_last_30   device_type
count    57804.000000  57804.000000
mean        10.516919      0.126237
std         28.328271      0.332119
min          0.000000      0.000000
25%          0.000000      0.000000
50%          1.000000      0.000000
75%          9.000000      0.000000
max       1365.000000      1.000000
```

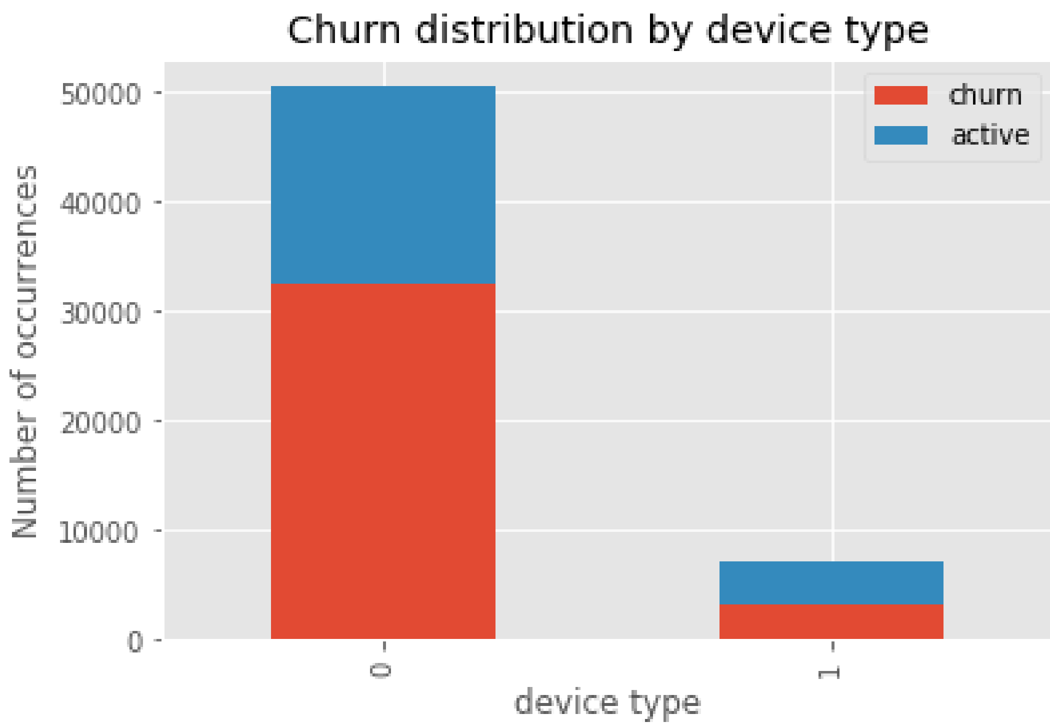
There are 5X times more users using andriod than iphone.

In [5]:
```
sns.countplot(df.device_type);
plt.xlabel('device_type');
plt.ylabel('Number of occurrences');
```



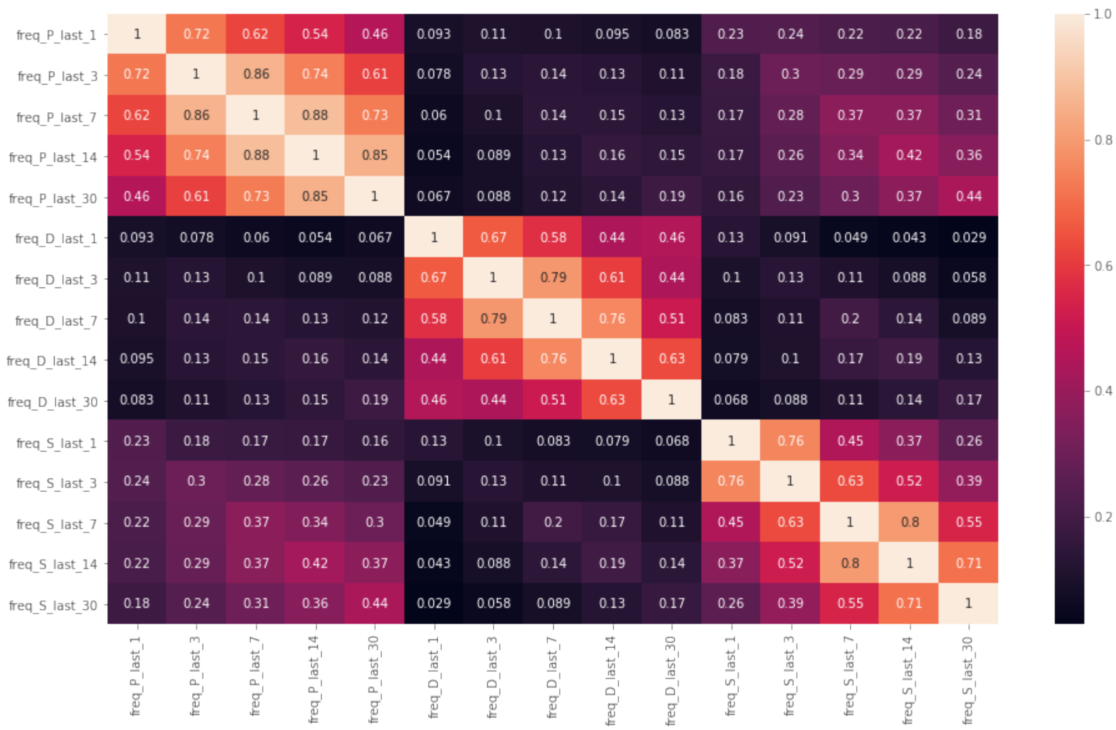

```
In [10]: fig = plt.figure()
         fig.set(alpha=0.2)

         label_0 = df.device_type[df.label == 0].value_counts()
         label_1 = df.device_type[df.label == 1].value_counts()
         df_tmp = pd.DataFrame({u'churn':label_1, u'active':label_0})
         df_tmp.plot(kind='bar', stacked=True)
         plt.title(u"Churn distribution by device type")
         plt.xlabel(u"device type")
         plt.ylabel(u"Number of occurrences")
         plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



From the above plot, there seems to have higher churn/active ratio of iphone users than android users.

```
In [16]: plt.subplots(figsize=(16,9))
         correlation_mat = df[selected_features].corr()
         sns.heatmap(correlation_mat, annot=True)

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b8c9be0>
```



We also calculate the Pearson correlation coefficent to quantify correlations between the features (variables). This is a measure of the strength and direction of a linear relationship between two variables: a value of -1 means the two variables are perfectly negatively linearly correlated and a value of +1 means the two variables are perfectly positively linearly correlated. The above figure shows different values of the correlation coeffient and how they appear graphically.

We see a high correlation among a few features, for example, freq_P_last14 and freq_P_last7. We should be very careful while implementing linear regression models on the dataset.

4.1 Define features and target

```
In [5]: selected_features = list(df.columns.values)
```

```
In [6]: selected_features.remove('uid')
        selected_features.remove('label')
        selected_features
```

```
Out[6]: ['freq_P_last_1',
         'freq_P_last_3',
         'freq_P_last_7',
         'freq_P_last_14',
         'freq_P_last_30',
         'freq_D_last_1',
         'freq_D_last_3',
         'freq_D_last_7',
         'freq_D_last_14',
```

```
          'freq_D_last_30',
          'freq_S_last_1',
          'freq_S_last_3',
          'freq_S_last_7',
          'freq_S_last_14',
          'freq_S_last_30',
          'device_type']
```

```
In [7]: X = df[selected_features]
        y = df['label']
```

```
In [51]: X.shape
```

```
Out[51]: (57804, 16)
```

```
In [52]: y[:10]
```

```
Out[52]: 0    1
         1    0
         2    0
         3    1
         4    1
         5    1
         6    1
         7    1
         8    1
         9    1
         Name: label, dtype: int64
```

4.2 Split dataset to train and test data

First, we split the data into training (80%) and testing sets (20%):

```
In [9]: # import train test split function from sklearn
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=(
```

4.3 Train model using sklearn

```
In [10]: # define function to perform train, test, and get model performance
         def train_test_model(clf, X_train, y_train, X_test, y_test):
             # Fit a model by providing X and y from training set
             clf.fit(X_train, y_train)

             # Make prediction on the training data
             y_train_pred = clf.predict(X_train)
             p_train_pred = clf.predict_proba(X_train)[:,1]

             # Make predictions on test data
```

```
    y_test_pred = clf.predict(X_test)
    p_test_pred = clf.predict_proba(X_test)[:,1]

    # print model results
    get_performance_metrics(y_train, p_train_pred, y_test, p_test_pred)
    plot_roc_curve(y_train, p_train_pred, y_test, p_test_pred)
```

4.3.1 Calculate the metric scores for the model

```
In [11]: %matplotlib inline
         import matplotlib.pyplot as plt
         from sklearn.metrics import roc_curve, auc

         def plot_roc_curve(y_train, y_train_pred, y_test, y_test_pred):
             roc_auc_train = roc_auc_score(y_train, y_train_pred)
             fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)

             roc_auc_test = roc_auc_score(y_test, y_test_pred)
             fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
             plt.figure()
             lw = 2
             plt.plot(fpr_train, tpr_train, color='green',
                      lw=lw, label='ROC Train (AUC = %0.4f)' % roc_auc_train)
             plt.plot(fpr_test, tpr_test, color='darkorange',
                      lw=lw, label='ROC Test (AUC = %0.4f)' % roc_auc_test)
             plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
             plt.xlim([0.0, 1.0])
             plt.ylim([0.0, 1.05])
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('Receiver operating characteristic example')
             plt.legend(loc="lower right")
             plt.show()
```

```
In [12]: # Import metrics functions from sklearn
         from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score, r
```

```
In [13]: # Helper method to print metric scores
         def get_performance_metrics(y_train, y_train_pred, y_test, y_test_pred, threshold=0.5)
             metric_names = ['AUC','Accuracy','Precision','Recall','f1-score']
             metric_values_train = [roc_auc_score(y_train, y_train_pred),
                             accuracy_score(y_train, y_train_pred>threshold),
                             precision_score(y_train, y_train_pred>threshold),
                             recall_score(y_train, y_train_pred>threshold),
                             f1_score(y_train, y_train_pred>threshold)
                            ]
             metric_values_test = [roc_auc_score(y_test, y_test_pred),
                             accuracy_score(y_test, y_test_pred>threshold),
```

```
                    precision_score(y_test, y_test_pred>threshold),
                    recall_score(y_test, y_test_pred>threshold),
                    f1_score(y_test, y_test_pred>threshold)
                   ]
all_metrics = pd.DataFrame({'metrics':metric_names,
                            'train':metric_values_train,
                            'test':metric_values_test},columns=['metrics','train'
print(all_metrics)
```

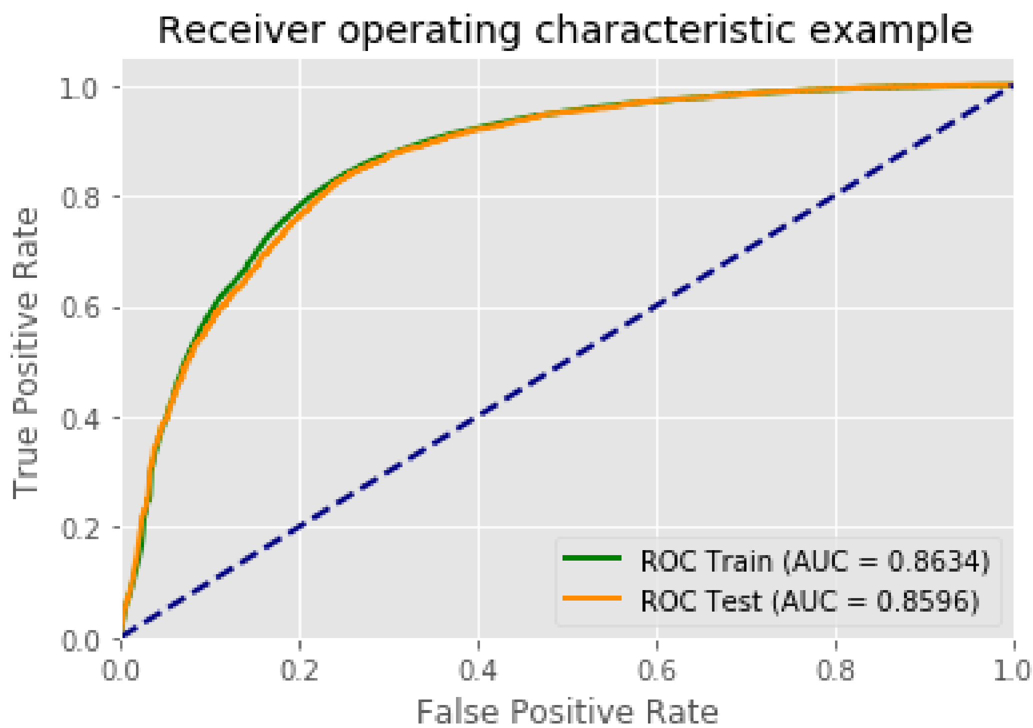
Since it is a binary classification problem, the data were fitted with * logistic regression * Random forest

And I started to train a logistic regression model as a baseline.

4.3.2 Fitting logstic regression model

```
In [58]: # Import logistic regression from sklearn
         from sklearn.linear_model import LogisticRegression

         # Initialize model by providing parameters
         # http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegre
         clf = LogisticRegression(C=1.0, penalty='l2')
         # Fit a model by providing X and y from training set
         clf.fit(X_train, y_train)

         # Train test model
         train_test_model(clf, X_train, y_train, X_test, y_test)
```

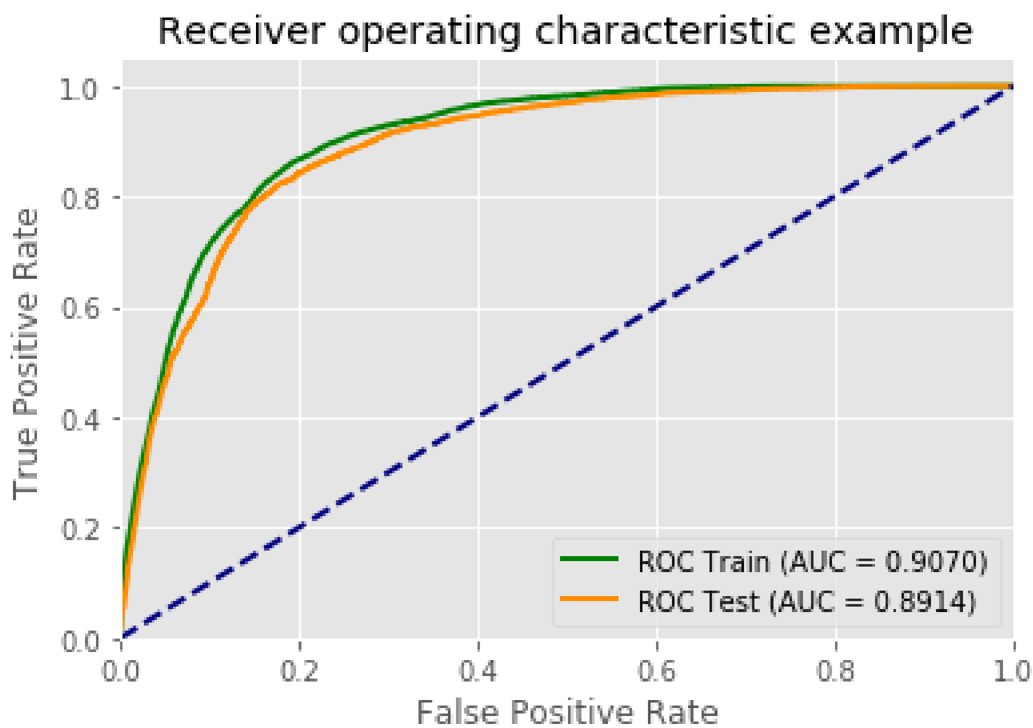
```
              train      test
metrics
AUC        0.863385  0.859576
Accuracy   0.778323  0.777528
Precision  0.753910  0.753248
Recall     0.954814  0.950932
f1-score   0.842551  0.840625
```



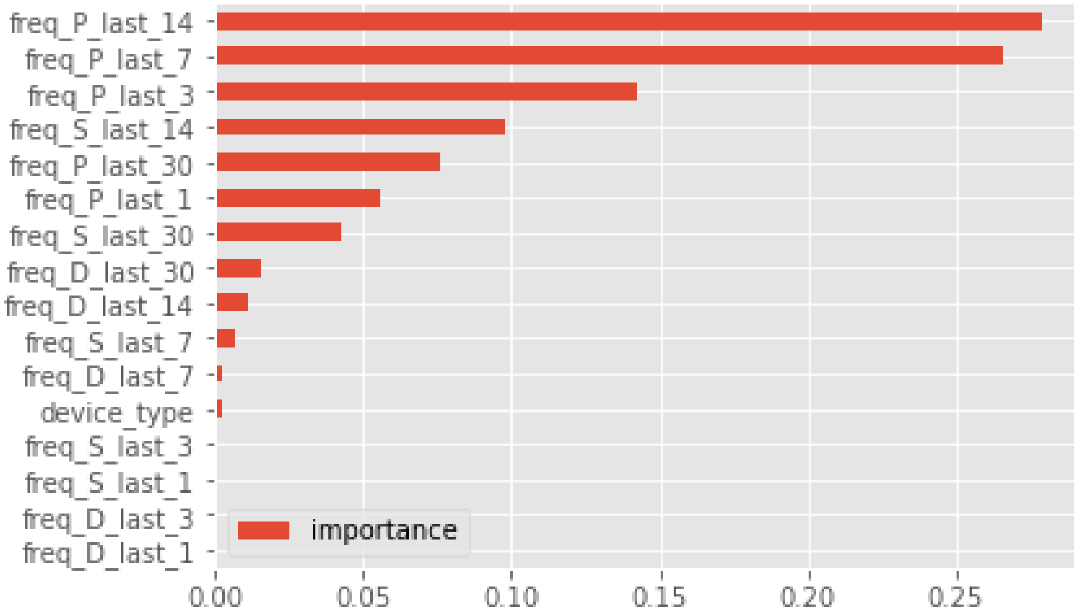
4.3.3 Random Forest

In [60]:
```
# fitting random forest
from sklearn.ensemble import RandomForestClassifier

# Choose some parameter combinations to try
parameters = {'n_estimators': 50,
              'max_features': 'auto',
              'criterion': 'gini',
              'max_depth': 20,
              'min_samples_split': 2,
              'min_samples_leaf': 20,
              'random_state': 0,
              'n_jobs': -1
              }

clf = RandomForestClassifier(**parameters)

# Fit a model by providing X and y from training set
clf.fit(X_train, y_train)

# Train test model
train_test_model(clf, X_train, y_train, X_test, y_test)
```

```
             train      test
metrics
AUC        0.906953  0.891396
Accuracy   0.846247  0.829081
Precision  0.854756  0.841750
Recall     0.906531  0.890369
f1-score   0.879882  0.865377
```




```
In [61]: df_feature_importance = pd.DataFrame()
         df_feature_importance['feature'] = selected_features
         df_feature_importance['importance'] = clf.feature_importances_
         df_feature_importance.sort_values('importance',inplace=True)

         ax = df_feature_importance.plot.barh()
         t = np.arange(len(df_feature_importance['feature']))
         ax.set_yticks(t)
         ax.set_yticklabels(df_feature_importance['feature'])
         plt.show()
```



4.3.4 HyperParameter Tuning: Grid Search

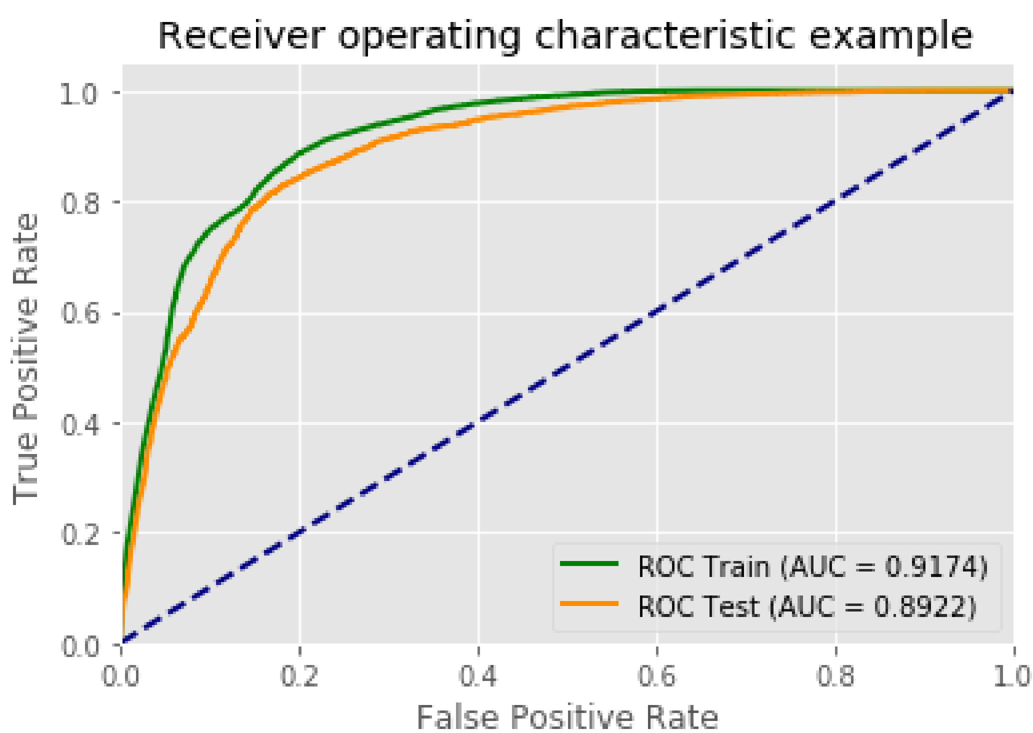
In [14]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, roc_auc_score, accuracy_score
from sklearn.model_selection import GridSearchCV

# Choose the type of classifier.
clf = RandomForestClassifier()

# Choose some parameter combinations to try
param_grid = {'n_estimators': [100,200],
              'max_features': ['auto'],
              'criterion': ['gini'],
              'max_depth': [15,20,25],
              'min_samples_split': [2],
              'min_samples_leaf': [2,10,20],
              'n_jobs':[-1]
             }

# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(roc_auc_score)

# Run the grid search
# read theory
grid_obj = GridSearchCV(clf, param_grid, cv=5, scoring=acc_scorer)
grid_obj = grid_obj.fit(X_train, y_train)
```

```
# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_

# Fit the best algorithm to the data.
clf.fit(X_train, y_train)
```

```
Out[14]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=20, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=10, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=-1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

```
In [15]: # Train test model
         train_test_model(clf, X_train, y_train, X_test, y_test)
```

```
              train      test
metrics
AUC        0.917397  0.892220
Accuracy   0.857124  0.831243
Precision  0.864507  0.845007
Recall     0.913110  0.889668
f1-score   0.888144  0.866762
```



5 Summary

From the above analysis, we can see that Random Forest outperforms logistic regression model for user churn prediction based on AUC score after hperparameter tuning using grid search and cross validation. Also, we might improve logistic regression performance by log transformation of features and doing more feature selection.

Throughout the analysis, I have learned several important things:

- Features such as frequency of users playing music in the window period of 14 days or 7 days appear to play an important role in user churn. It provides us a timeline when we need to take actions to stop churn. For example, during thisperiod of time, we can send users reminder and recommend songs they might like.
- On the other hand, users behavior of downloading songs plays less role in this churn classification task.
- There does not seem to be a relationship between dervice type and churn.

6 References

1. https://medium.com/@InDataLabs/effective-customer-churn-analysis-prediction-6fce3626f2c2
2. http://tesi.cab.unipd.it/53212/1/Valentino_Avon_-_1104319.pdf