# Data analysis and visualization

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from mpl_toolkits.basemap import Basemap
```

In [2]:
```
# Input data from "/data" director
os.listdir("data")
```

Out[2]:
```
['app_events.csv',
 'phone_brand_device_model.csv',
 'app_events.csv.zip',
 'app_labels.csv.zip',
 'app_labels.csv',
 'gender_age_test.csv.zip',
 'phone_brand_device_model.csv.zip',
 'gender_age_test.csv',
 'events.csv',
 'events.csv.zip',
 'sample_submission.csv.zip',
 'gender_age_train.csv.zip',
 'gender_age_train.csv',
 'label_categories.csv.zip',
 'sample_submission.csv',
 'label_categories.csv']
```

In [3]:
```
import seaborn as sns
sns.set(color_codes=True)
app_event=pd.read_csv("data/events.csv")
app_event.shape
```

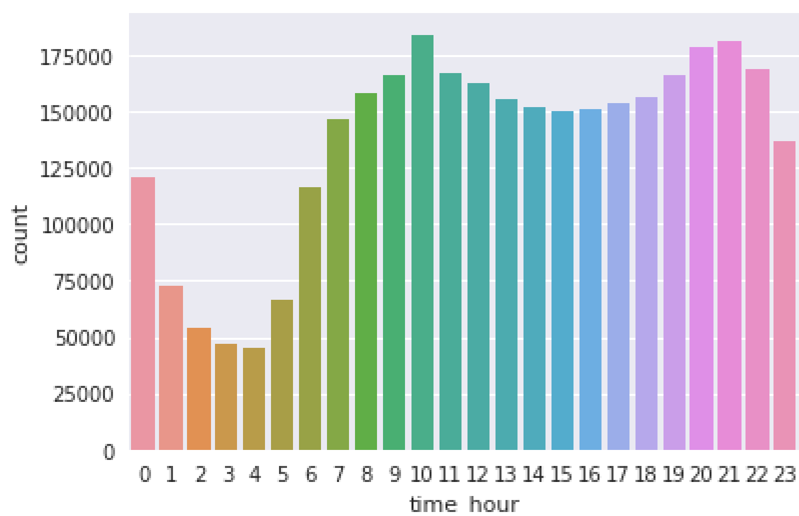
Out[3]:
```
(3252950, 5)
```

In [4]:
```
app_event.timestamp=pd.to_datetime(app_event.timestamp)
app_event['time_hour'] = app_event.timestamp.apply(lambda x: x.hour)
```

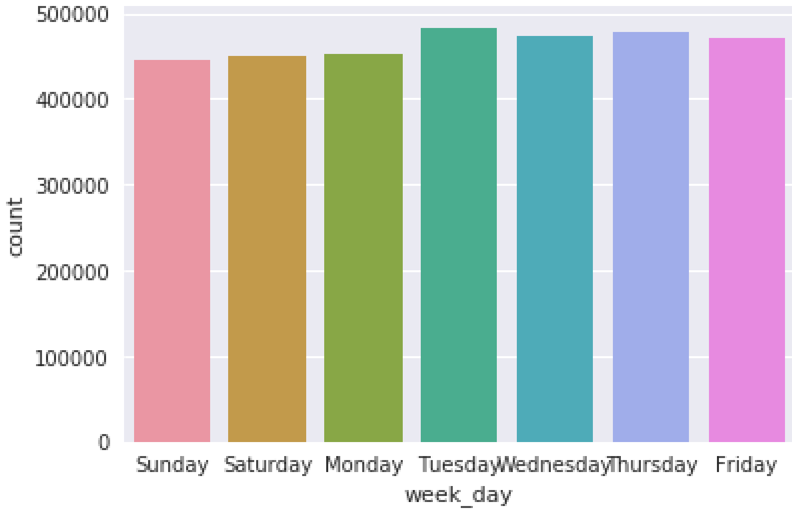
In [5]:
```
# Show frequency of events by hour
app_event['time_hour'].value_counts()
```

Out[5]:
```
10    183839
21    181175
20    178179
22    168246
11    167025
19    166160
9     166061
12    162745
8     157896
18    156209
13    155337
17    153516
14    151379
16    150732
15    149912
7     146667
23    136339
0     120512
6     116370
1      72671
5      66411
2      53764
3      47048
4      44757
Name: time_hour, dtype: int64
```

In [6]:
```
ax = sns.countplot(x="time_hour", data=app_event)
```

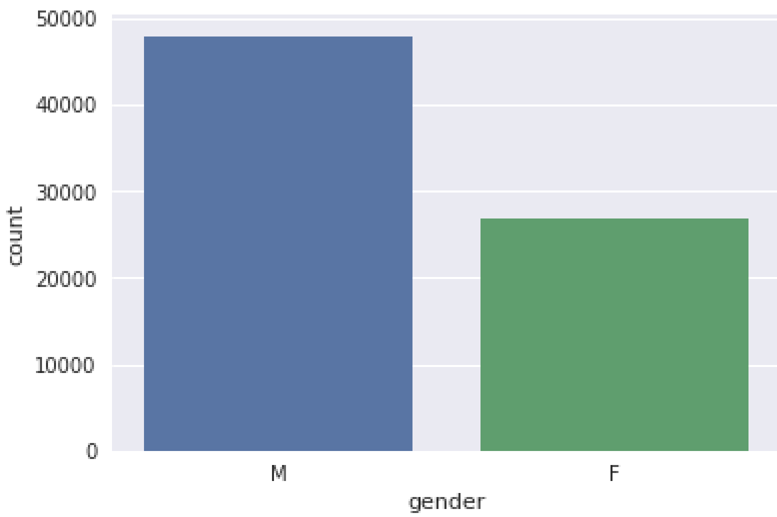


In [7]:
```
import calendar
app_event['week_day'] = app_event.timestamp.apply(lambda x: calendar.day_na
me[x.weekday()])
ax = sns.countplot(x="week_day", data=app_event)
```

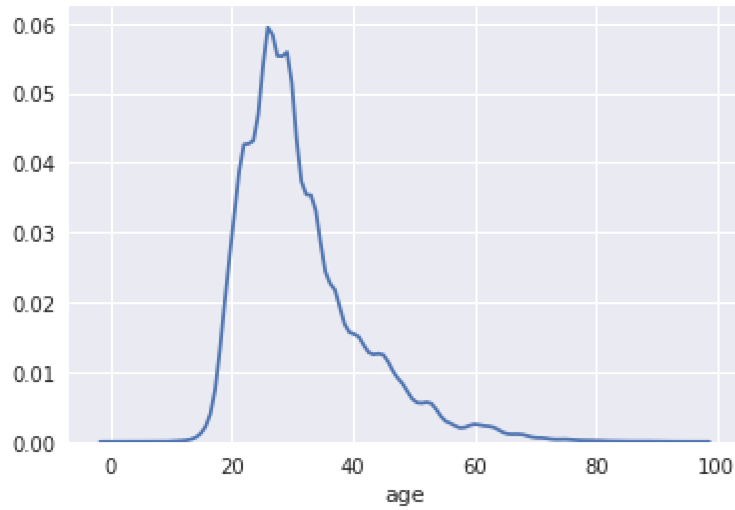



In [8]:
```
gender=pd.read_csv("data/gender_age_train.csv")
print(gender.gender.value_counts())
ax = sns.countplot(x="gender", data=gender)
```

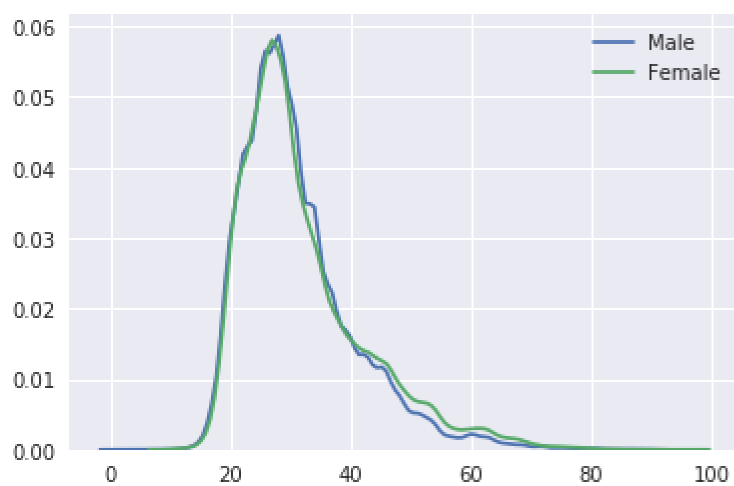
```
M    47904
F    26741
Name: gender, dtype: int64
```



In [9]:
```
# Distribution by age
sns.distplot(gender.age, hist=False);
```



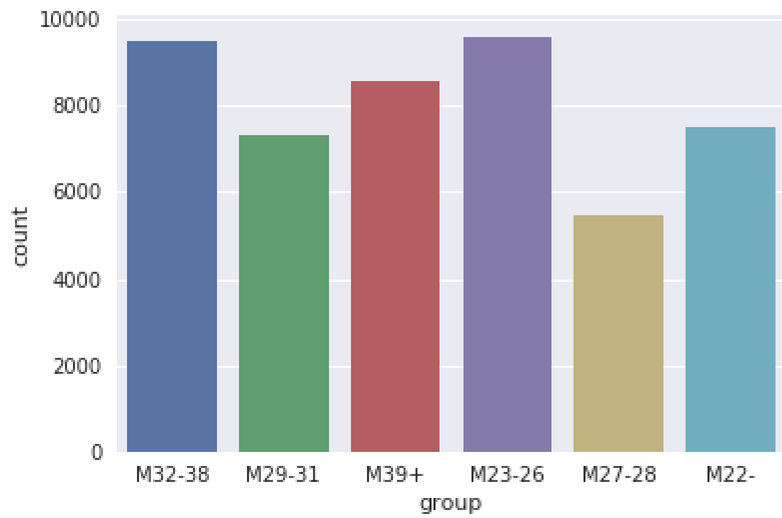

In [10]:
```
# Distribution by sex
sns.kdeplot(gender.age[gender.gender=="M"], label="Male")
sns.kdeplot(gender.age[gender.gender=="F"],  label="Female")
plt.legend();
```




Female at old age are using mobiles little bit more than males at old age

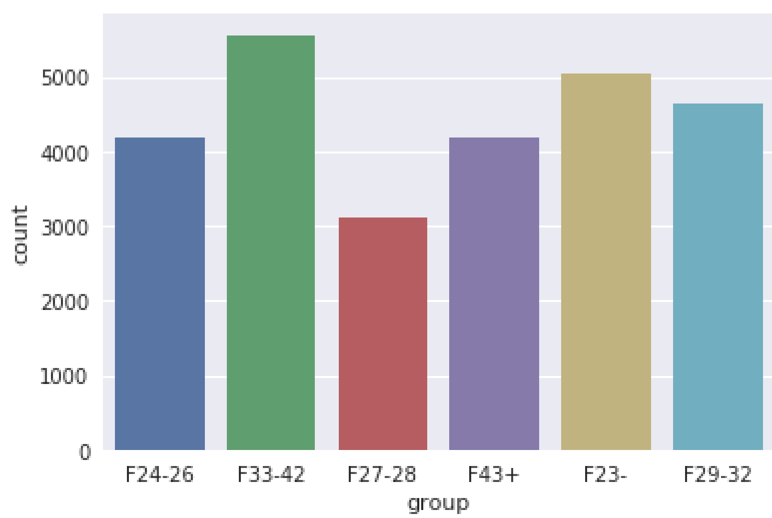
In [11]:
```
print("Male mobile usage count by age")
ax = sns.countplot(x="group", data=gender[gender.gender=="M"])
```

```
Male mobile usage count by age
```




In [12]:
```
print("Female mobile usage count by age")
ax = sns.countplot(x="group", data=gender[gender.gender=="F"])
```

```
Female mobile usage count by age
```



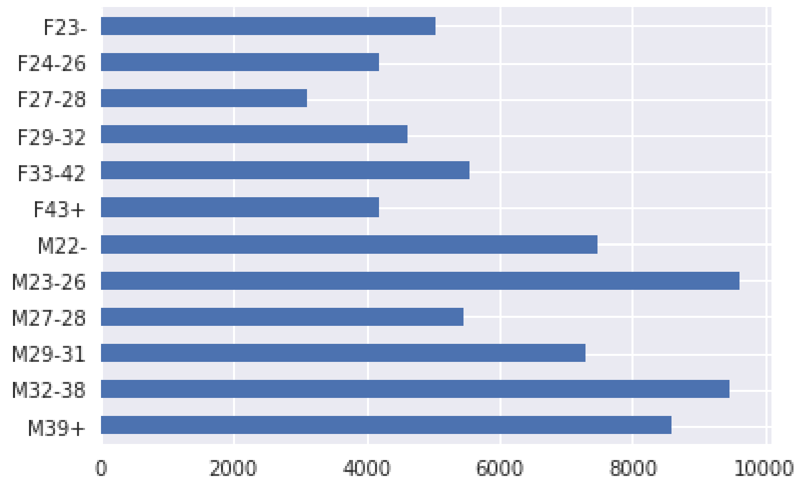

In [13]:
```
appscategories=pd.read_csv("data/label_categories.csv")
print(appscategories.head())
print(appscategories.shape)
```

```
   label_id           category
0         1                NaN
1         2     game-game type
2         3   game-Game themes
3         4     game-Art Style
4         5  game-Leisure time
(930, 2)
```

## Joint visualisation - male and female

In [14]:
```
gender.group.value_counts().sort_index(ascending=False).plot(kind='barh')
```

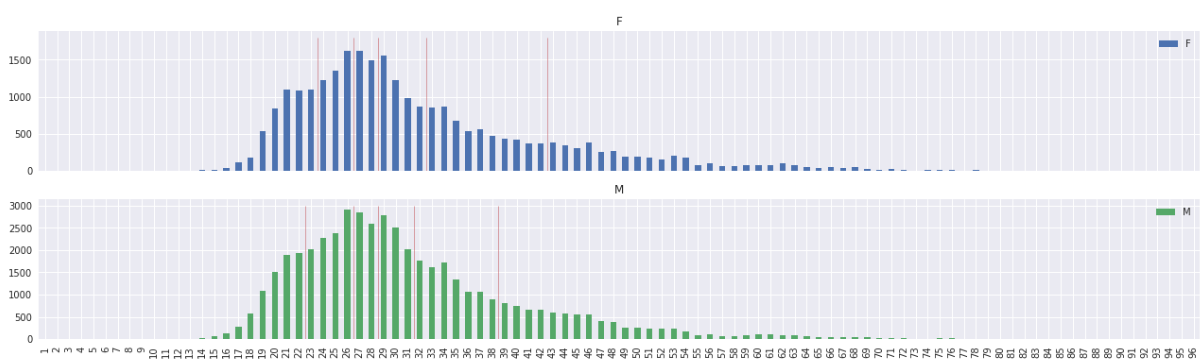
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f31e9ea9a58>




Further break down by age

In [15]:
```
c = gender.groupby(['age','gender']).size().unstack().reindex(index=np.aran
ge(gender.age.min(), gender.age.max()+1)).fillna(0)
ax1, ax2 = c.plot(kind='bar',figsize=(22,6),subplots=True);
ax1.vlines(np.array([23,26,28,32,42])-0.5,0,1800,alpha=0.5,linewidth=1,colo
r='r')
ax2.vlines(np.array([22,26,28,31,38])-0.5,0,3000,alpha=0.5,linewidth=1,colo
r='r')
```

Out[15]: <matplotlib.collections.LineCollection at 0x7f31e9dfa438>




# Locations visualization

In [16]:
```
df_events = pd.read_csv("data/events.csv", dtype={'device_id': np.str})
df_events.head()
```

Out[16]:

| | event_id | device_id | timestamp | longitude | latitude |
|---|---|---|---|---|---|
| **0** | 1 | 29182687948017175 | 2016-05-01 00:55:25 | 121.38 | 31.24 |
| **1** | 2 | -6401643145415154744 | 2016-05-01 00:54:12 | 103.65 | 30.97 |
| **2** | 3 | -4833982096941402721 | 2016-05-01 00:08:05 | 106.60 | 29.70 |
| **3** | 4 | -6815121365017318426 | 2016-05-01 00:06:40 | 104.27 | 23.28 |
| **4** | 5 | -5373797595892518570 | 2016-05-01 00:07:18 | 115.88 | 28.66 |

## Locations of events - World map

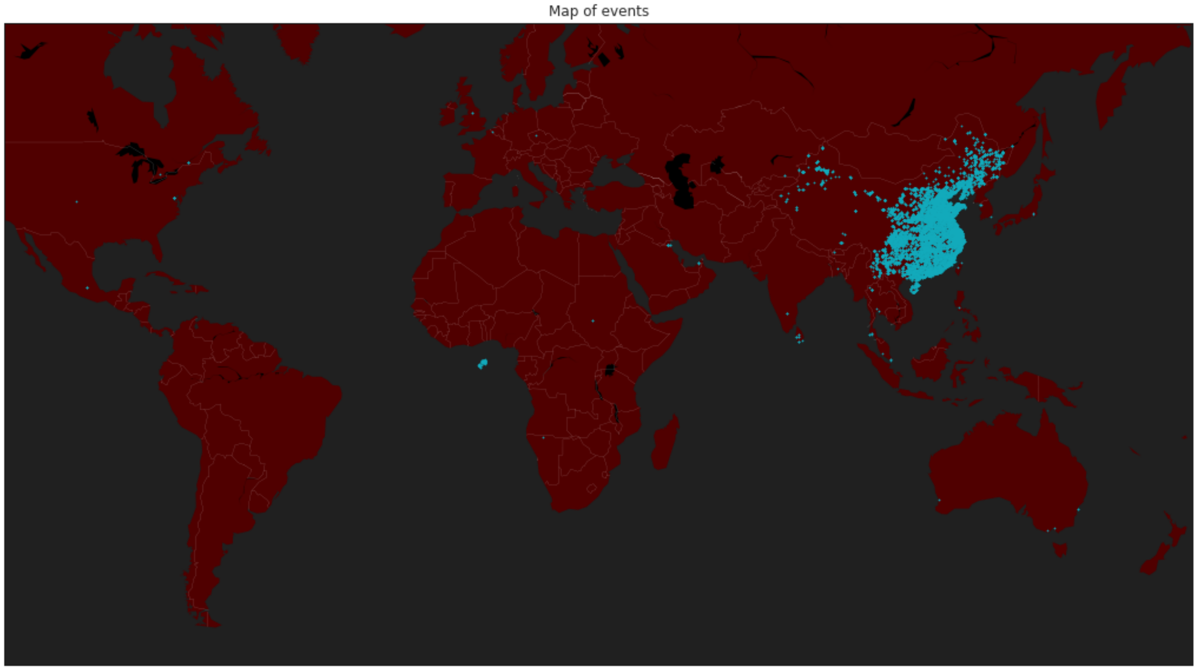
Expand size of the screen so larger map can nicely fit without truncated window with scroller.

In [17]:
```
%%html
<style>
.output_wrapper, .output {
    height:auto !important;
    max-height:1000px;  /* your desired max-height here */
}
.output_scroll {
    box-shadow:none !important;
    webkit-box-shadow:none !important;
}
</style>
```

In [18]:
```
# Set plot
df_events_sample = df_events.sample(n=90000)
plt.figure(1, figsize=(20,10))
pd.set_option('display.max_colwidth', -1)

# Map of World
map = Basemap(projection='merc',
             llcrnrlat=-60,
             urcrnrlat=65,
             llcrnrlon=-120,
             urcrnrlon=180,
             lat_ts=0,
             resolution='c')

map.fillcontinents(color='#500000',lake_color='#000000') # grey land, black
 lakes
map.drawmapboundary(fill_color='#202020')                # black background
map.drawcountries(linewidth=0.1, color="w")              # white line of co
untry borders

# Plot the data
mxy = map(df_events_sample["longitude"].tolist(), df_events_sample["latitud
e"].tolist())
map.scatter(mxy[0], mxy[1], s=3, c="#12AABB", lw=0, alpha=1, zorder=5)

plt.title("Map of events")
plt.show()
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1767: MatplotlibDeprecationWarning: The get_axis_bgcolor functio
n was deprecated in version 2.0. Use get_facecolor instead.
  axisbgc = ax.get_axis_bgcolor()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1698: MatplotlibDeprecationWarning: The axesPatch function was d
eprecated in version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3222: MatplotlibDeprecationWarning: The ishold function was depr
ecated in version 2.0.
  b = ax.ishold()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3231: MatplotlibDeprecationWarning: axes.hold is deprecated.
    See the API Changes document (http://matplotlib.org/api/api_changes.htm
l)
    for more details.
  ax.hold(b)
```

Map of events



As we can see some events have (lat,lon) = (0,0) which probably means that location couldn't be determined. We can find all event on that location and all events that have longitud and latitud less than 1 which means they are really close to that location. That is location is not probable since it's points to sea area close to African cost.

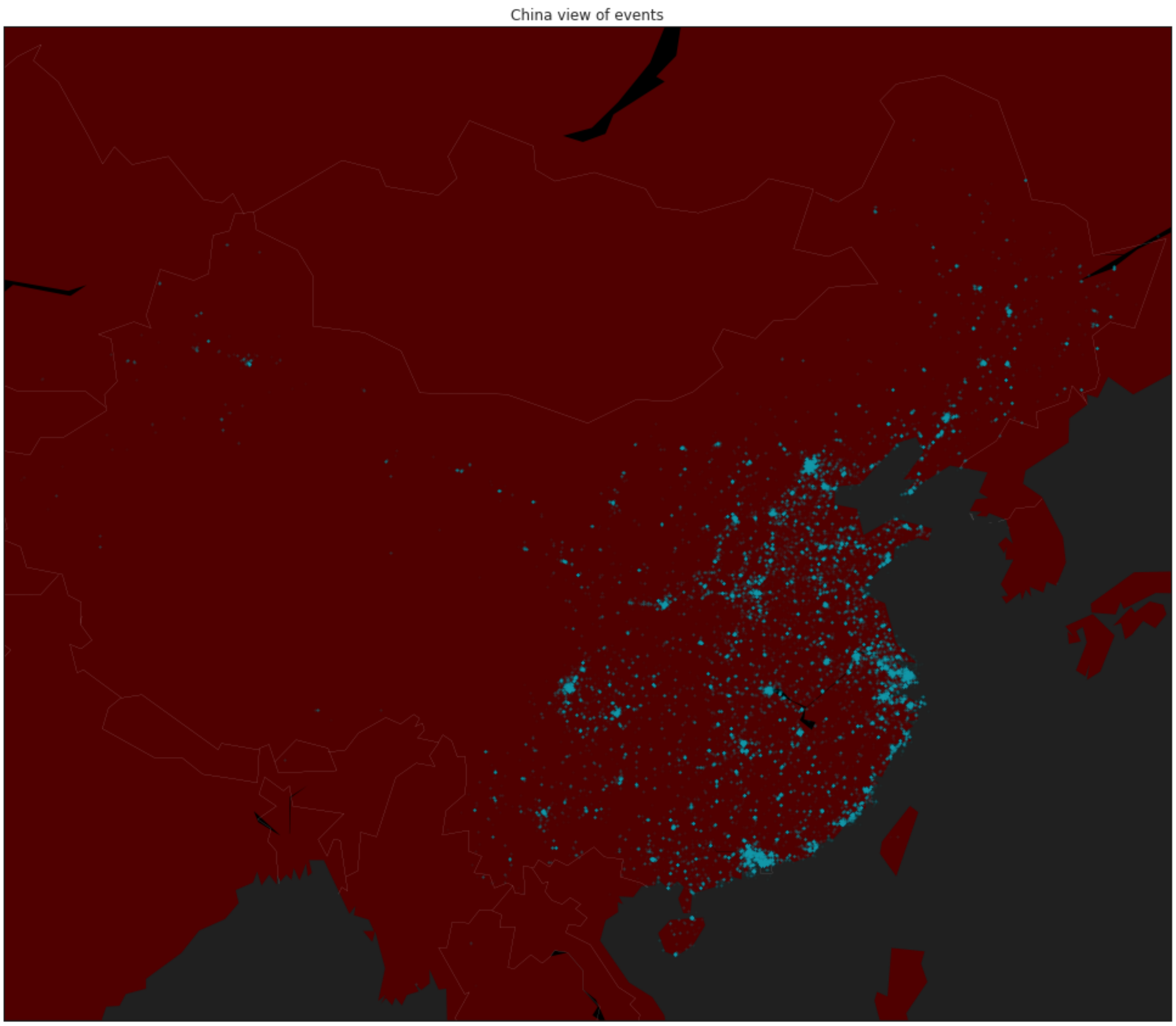
In [19]:
```
df_at0 = df_events[(df_events["longitude"]==0) & (df_events["latitude"]==0
)]
df_near0 = df_events[(df_events["longitude"]>-1) &\
                     (df_events["longitude"]<1) &\
                     (df_events["latitude"]>-1) &\
                     (df_events["latitude"]<1)]

print("Total number of events:", len(df_events))
print("Number of events at (0,0):", len(df_at0))
print("Number of events near (0,0):", len(df_near0))
```

```
Total number of events: 3252950
Number of events at (0,0): 968675
Number of events near (0,0): 969871
```

## Locations of events - zooming in region of China

In [20]:
```
# Locate China region
lon_min, lon_max = 75, 135
lat_min, lat_max = 15, 55

idx_china = (df_events["longitude"]>lon_min) &\
            (df_events["longitude"]<lon_max) &\
            (df_events["latitude"]>lat_min) &\
            (df_events["latitude"]<lat_max)

df_events_china = df_events[idx_china].sample(n=100000)

# China
plt.figure(2, figsize=(20,15))

map_zoom = Basemap(projection='merc',
             llcrnrlat=lat_min,
             urcrnrlat=lat_max,
             llcrnrlon=lon_min,
             urcrnrlon=lon_max,
             lat_ts=35,
             resolution='c')

map_zoom.fillcontinents(color='#500000',lake_color='#000000') # dark grey l
and, black lakes
map_zoom.drawmapboundary(fill_color='#202020')                # black backg
round
map_zoom.drawcountries(linewidth=0.1, color="w")              # thin white
 line for country borders


# Plot the data
mxy = map_zoom(df_events_china["longitude"].tolist(), df_events_china["lati
tude"].tolist())
map_zoom.scatter(mxy[0], mxy[1], s=5, c="#12AABB", lw=0, alpha=0.05, zorder
=5)

plt.title("China view of events")
plt.show()
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1767: MatplotlibDeprecationWarning: The get_axis_bgcolor functio
n was deprecated in version 2.0. Use get_facecolor instead.
  axisbgc = ax.get_axis_bgcolor()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1698: MatplotlibDeprecationWarning: The axesPatch function was d
eprecated in version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3222: MatplotlibDeprecationWarning: The ishold function was depr
ecated in version 2.0.
  b = ax.ishold()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3231: MatplotlibDeprecationWarning: axes.hold is deprecated.
    See the API Changes document (http://matplotlib.org/api/api_changes.htm
l)
    for more details.
  ax.hold(b)
```

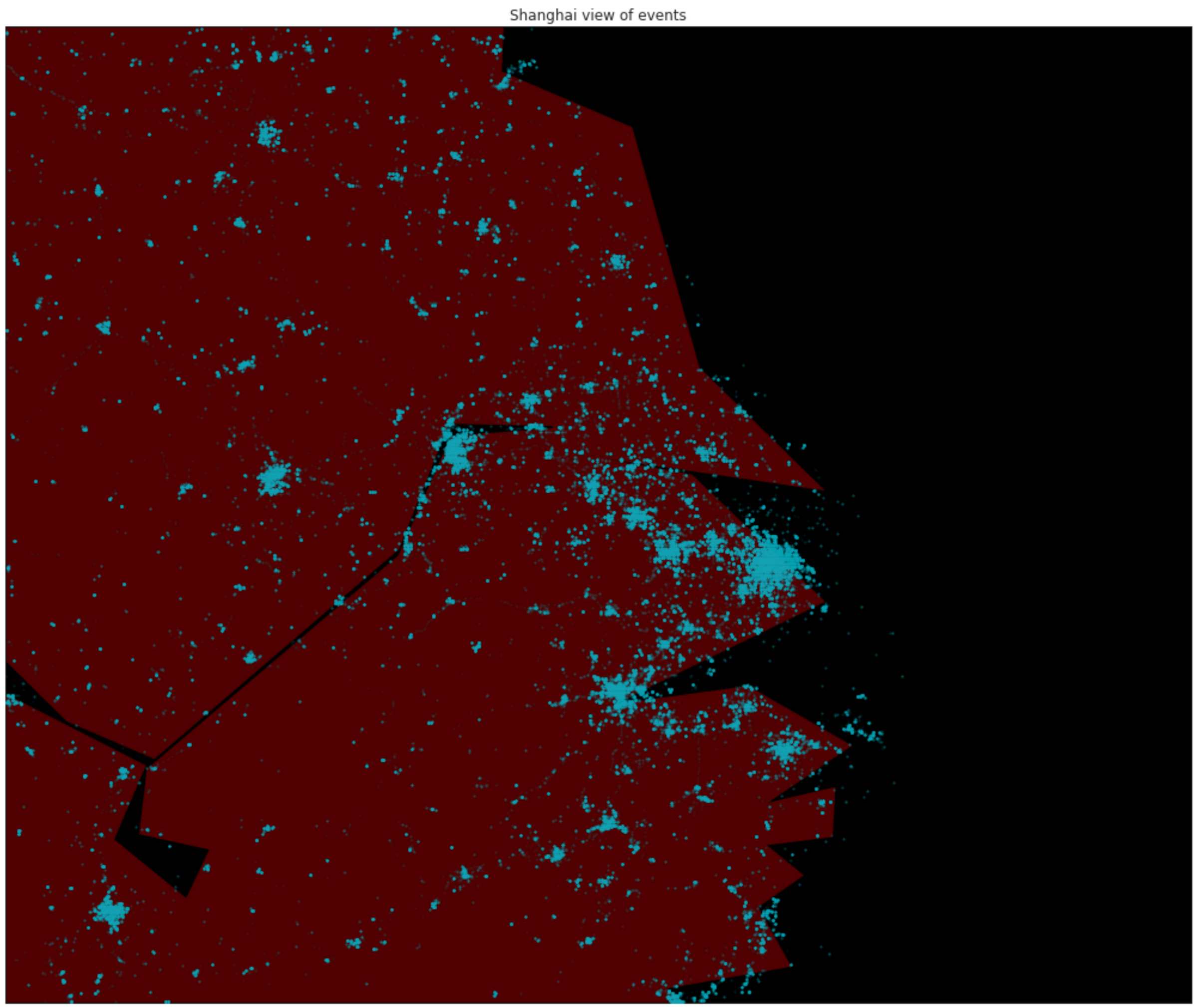



This map nicely shows population density of China. We'll analyze one city region in this case we'll take Shanghai. What follows are maps that are showing longitude and latitude areas. In same way, we can zoom on any area on earth for showing more details in that area.

## Locations of applications events - region of Shanghai

In [21]:
```
# Shanghai sits on 31.2304° N, 121.4737° E
# Sampling wider Shanghai region
lon_min, lon_max = 115, 125
lat_min, lat_max = 28, 35

idx_shanghai = (df_events["longitude"]>lon_min) &\
               (df_events["longitude"]<lon_max) &\
               (df_events["latitude"]>lat_min) &\
               (df_events["latitude"]<lat_max)

df_events_shanghai = df_events[idx_shanghai]

# Map of Shanghai region
plt.figure(3, figsize=(20,15))

m_shanghai = Basemap(projection='merc',
             llcrnrlat=lat_min,
             urcrnrlat=lat_max,
             llcrnrlon=lon_min,
             urcrnrlon=lon_max,
             lat_ts=35,
             resolution='c')

m_shanghai.fillcontinents(color='#500000',lake_color='#000000') # dark lan
d, black lakes
m_shanghai.drawmapboundary(fill_color='#000000')                # black bac
kground
m_shanghai.drawcountries(linewidth=0.1, color="w")              # white lin
e for country borders

# Plot the data
mxy = m_shanghai(df_events_shanghai["longitude"].tolist(), df_events_shangh
ai["latitude"].tolist())
m_shanghai.scatter(mxy[0], mxy[1], s=5, c="#12AABB", lw=0, alpha=0.1, zorde
r=5)

plt.title("Shanghai view of events")
plt.show()
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1767: MatplotlibDeprecationWarning: The get_axis_bgcolor functio
n was deprecated in version 2.0. Use get_facecolor instead.
  axisbgc = ax.get_axis_bgcolor()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1698: MatplotlibDeprecationWarning: The axesPatch function was d
eprecated in version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3222: MatplotlibDeprecationWarning: The ishold function was depr
ecated in version 2.0.
  b = ax.ishold()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3231: MatplotlibDeprecationWarning: axes.hold is deprecated.
    See the API Changes document (http://matplotlib.org/api/api_changes.htm
l)
    for more details.
  ax.hold(b)
```




We can see that population around big cities is very disperssed.

Now we'll show male and female app events

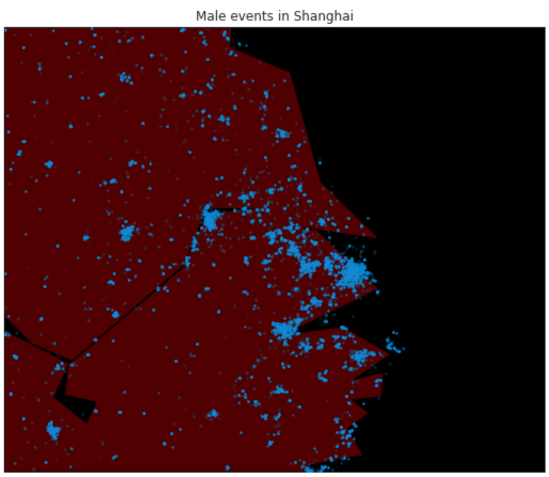
#### Male and female app events in region of Shanghai

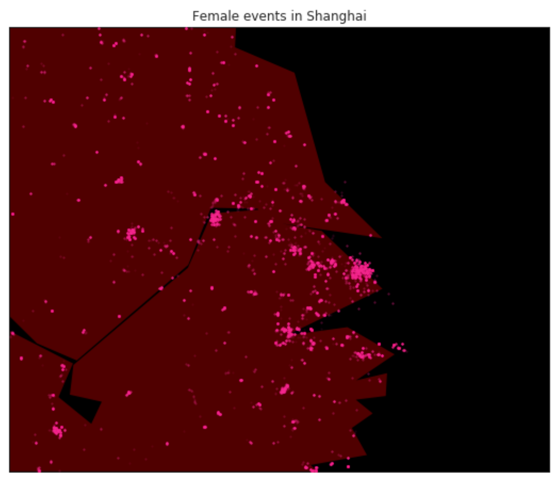
In [22]:
```
# Load the train data and join on the events
df_train = pd.read_csv("data/gender_age_train.csv", dtype={'device_id': np.
str})

df_plot = pd.merge(df_train, df_events_shanghai, on="device_id", how="inne
r")

df_m = df_plot[df_plot["gender"]=="M"]
df_f = df_plot[df_plot["gender"]=="F"]
```

Visualize male and female events in Shanghai

In [23]:
```
# Female and male plot

plt.figure(4, figsize=(20,10))

plt.subplot(121)
m_sh_m = Basemap(projection='merc',
             llcrnrlat=lat_min,
             urcrnrlat=lat_max,
             llcrnrlon=lon_min,
             urcrnrlon=lon_max,
             lat_ts=35,
             resolution='c')
m_sh_m.fillcontinents(color='#500000',lake_color='#000000') # dark grey lan
d, black lakes
m_sh_m.drawmapboundary(fill_color='#000000')                # black backgro
und
m_sh_m.drawcountries(linewidth=0.1, color="w")              # thin white li
ne for country borders
mxy = m_sh_m(df_m["longitude"].tolist(), df_m["latitude"].tolist())
m_sh_m.scatter(mxy[0], mxy[1], s=5, c="#1292db", lw=0, alpha=0.1, zorder=5)
plt.title("Male events in Shanghai")

plt.subplot(122)
m_sh_f = Basemap(projection='merc',
             llcrnrlat=lat_min,
             urcrnrlat=lat_max,
             llcrnrlon=lon_min,
             urcrnrlon=lon_max,
             lat_ts=35,
             resolution='c')
m_sh_f.fillcontinents(color='#500000',lake_color='#000000') # dark grey lan
d, black lakes
m_sh_f.drawmapboundary(fill_color='#000000')                # black backgro
und
m_sh_f.drawcountries(linewidth=0.1, color="w")              # thin white li
ne for country borders
mxy = m_sh_f(df_f["longitude"].tolist(), df_f["latitude"].tolist())
m_sh_f.scatter(mxy[0], mxy[1], s=5, c="#fd3096", lw=0, alpha=0.1, zorder=5)
plt.title("Female events in Shanghai")

plt.show()
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1767: MatplotlibDeprecationWarning: The get_axis_bgcolor functio
n was deprecated in version 2.0. Use get_facecolor instead.
  axisbgc = ax.get_axis_bgcolor()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:1698: MatplotlibDeprecationWarning: The axesPatch function was d
eprecated in version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3222: MatplotlibDeprecationWarning: The ishold function was depr
ecated in version 2.0.
  b = ax.ishold()
/home/ec2-user/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/_
_init__.py:3231: MatplotlibDeprecationWarning: axes.hold is deprecated.
    See the API Changes document (http://matplotlib.org/api/api_changes.htm
l)
    for more details.
  ax.hold(b)
```




For marketing analysis, this might be interesting for further exploration. Which city areas are showing more men activities and which are showing more female activities and in which times of day?

# Analysis

## Problem classification

Our task is to build a model predicting users’ demographic characteristics based on their app usage, geolocation, and mobile device properties. So we need to solv multiclass classification problem This is case where one label needs to be predicted based on several others.

## Logistic regression

Logistic regression alghoritham could be obvious choice for that. In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme.

In [24]:
```
from sklearn.preprocessing import LabelEncoder
from scipy.sparse import csr_matrix, hstack
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import StratifiedKFold
from sklearn.metrics import log_loss
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/sklearn/cross_validati
on.py:41: DeprecationWarning: This module was deprecated in version 0.18 in
favor of the model_selection module into which all the refactored classes a
nd functions are moved. Also note that the interface of the new CV iterator
s are different from that of this module. This module will be removed in 0.
20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

## Loading of data

In [25]:
```
# Read gender Train and Test data
datadir = 'data'
g_a_train = pd.read_csv(os.path.join(datadir,'gender_age_train.csv'),index_
col='device_id')
g_a_test = pd.read_csv(os.path.join(datadir,'gender_age_test.csv'),index_co
l = 'device_id')
phone = pd.read_csv(os.path.join(datadir,'phone_brand_device_model.csv'))

print(g_a_train.head())
print("---------")
print(g_a_test.head())
print("---------")
print(phone.head())
```

```
                     gender  age   group
device_id
-8076087639492063270  M      35   M32-38
-2897161552818060146  M      35   M32-38
-8260683887967679142  M      35   M32-38
-4938849341048082022  M      30   M29-31
 245133531816851882   M      30   M29-31
---------
Empty DataFrame
Columns: []
Index: [1002079943728939269, -1547860181818787117, 7374582448058474277, -62
20210354783429585, -5893464122623104785]
---------
             device_id phone_brand   device_model
0 -8890648629457979026  小米           红米
1  1277779817574759137  小米           MI 2
2  5137427614288105724  三星           Galaxy S4
3  3669464369358936369  SUGAR       时尚手机
4 -5019277647504317457  三星           Galaxy Note 2
```

In [26]:
```
# Remove duplicate device ids in the phones
phone = phone.drop_duplicates(subset=['device_id'], keep='first').set_index
('device_id')
events = pd.read_csv('data/events.csv',parse_dates=['timestamp'],index_col=
'event_id')
appevents = pd.read_csv('data/app_events.csv',usecols=['event_id','app_id',
'is_active'],dtype={'is_active':bool})
applabels = pd.read_csv('data/app_labels.csv')
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/numpy/lib/arraysetops.
py:463: FutureWarning: elementwise comparison failed; returning scalar inst
ead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```

In [27]:
```
print(phone.head())
print("---------")
print(events.head())
print("---------")
print(appevents.head())
print("---------")
print(applabels.head())
print("---------")
```

```
                     phone_brand   device_model
device_id
-8890648629457979026  小米           红米
 1277779817574759137  小米           MI 2
 5137427614288105724  三星           Galaxy S4
 3669464369358936369  SUGAR       时尚手机
-5019277647504317457  三星           Galaxy Note 2
---------
                    device_id           timestamp  longitude  latitude
event_id
1          29182687948017175   2016-05-01 00:55:25  121.38     31.24
2         -6401643145415154744 2016-05-01 00:54:12  103.65     30.97
3         -4833982096941402721 2016-05-01 00:08:05  106.60     29.70
4         -6815121365017318426 2016-05-01 00:06:40  104.27     23.28
5         -5373797595892518570 2016-05-01 00:07:18  115.88     28.66
---------
   event_id               app_id  is_active
0  2         5927333115845830913  True
1  2        -5720078949152207372  False
2  2        -1633887856876571208  False
3  2        -653184325010919369   True
4  2         8693964245073640147  True
---------
                app_id  label_id
0  7324884708820027918  251
1 -4494216993218550286  251
2  6058196446775239644  406
3  6058196446775239644  407
4  8694625920731541625  406
---------
```

## Main feature selection

Main features chosen are:

- phone brand
- device model
- installed apps
- app labels

We need to one-hot encode everything and put in sparse matrices which will help deal with a very large number of features. Regarding "Phone brand" feature; we'll make two columns that show which train or test set row a particular device_id belongs to.

In [28]:
```
g_a_train['trainr'] = np.arange(g_a_train.shape[0])
g_a_test['testr'] = np.arange(g_a_test.shape[0])
```

In [29]:
```
print(g_a_train.head())
print("---------")
print(g_a_test.head())
```

```
                     gender  age   group  trainr
device_id
-8076087639492063270  M      35   M32-38  0
-2897161552818060146  M      35   M32-38  1
-8260683887967679142  M      35   M32-38  2
-4938849341048082022  M      30   M29-31  3
 245133531816851882   M      30   M29-31  4
---------
                      testr
device_id
 1002079943728939269  0
-1547860181818787117  1
 7374582448058474277  2
-6220210354783429585  3
-5893464122623104785  4
```

Constructing sparse matrix of features in following wasy:

csr_matrix((data, (row_ind, col_ind)), [shape=(M, N)]) where `data`, `row_ind` and `col_ind` satisfy the relationship `a[row_ind[k], col_ind[k]] = data[k]`

This allows us to define what values to put into certain places in a sparse matrix. For phone brand data the data array will be all ones, row_ind will be the row number of a device and col_ind will be the number of brand.

### Brand features

In [30]:
```
brand_encoder = LabelEncoder().fit(phone.phone_brand)
phone['brand'] = brand_encoder.transform(phone['phone_brand'])
g_a_train['brand'] = phone['brand']
g_a_test['brand'] = phone['brand']
Xtr_brand = csr_matrix((np.ones(g_a_train.shape[0]),(g_a_train.trainr, g_a_
train.brand)))
Xte_brand = csr_matrix((np.ones(g_a_test.shape[0]),(g_a_test.testr,g_a_test
.brand)))
print('Brand features: train shape {}, test shape {}'.format(Xtr_brand.shap
e, Xte_brand.shape))
```

```
Brand features: train shape (74645, 131), test shape (112071, 131)
```

#### Device model

In [31]:
```
phone_model = phone.phone_brand.str.cat(phone.device_model)
model_encoder = LabelEncoder().fit(phone_model)
phone['model'] = model_encoder.transform(phone_model)
g_a_train['model'] = phone['model']
g_a_test['model'] = phone['model']
Xtr_model = csr_matrix((np.ones(g_a_train.shape[0]),
                        (g_a_train.trainr, g_a_train.model)))
Xte_model = csr_matrix((np.ones(g_a_test.shape[0]),
                        (g_a_test.testr, g_a_test.model)))
print('Device model features: train shape {}, test shape {}'.format(Xtr_mod
el.shape, Xte_model.shape))
```

```
Device model features: train shape (74645, 1667), test shape (112071, 1667)
```

#### Installed apps features

For each device we want to have list of installed applications. So we'll have as many feature columns as there are distinct apps.

Apps are linked to devices through events. So we'll do the following:

merge device_id column from events table to app_events group the resulting dataframe by device_id and app and aggregate merge in trainrow and testrow columns to know at which row to put each device in the features matrix

In [32]:
```
apps_encoder = LabelEncoder().fit(appevents.app_id)
appevents['app'] = apps_encoder.transform(appevents.app_id)
napps = len(apps_encoder.classes_)
deviceapps = (appevents.merge(events[['device_id']], how='left',left_on='ev
ent_id',right_index=True)
                       .groupby(['device_id','app'])['app'].agg(['size'])
                       .merge(g_a_train[['trainr']], how='left', left_index
=True, right_index=True)
                       .merge(g_a_test[['testr']], how='left', left_index=T
rue, right_index=True)
                       .reset_index())
```

In [33]:
```
deviceapps.head()
```

Out[33]:

| | device_id | app | size | trainr | testr |
|---|---|---|---|---|---|
| **0** | -9222956879900151005 | 548 | 18 | 21594.0 | NaN |
| **1** | -9222956879900151005 | 1096 | 18 | 21594.0 | NaN |
| **2** | -9222956879900151005 | 1248 | 26 | 21594.0 | NaN |
| **3** | -9222956879900151005 | 1545 | 12 | 21594.0 | NaN |
| **4** | -9222956879900151005 | 1664 | 18 | 21594.0 | NaN |

Next step is to build a feature matrix. Data will be all ones, row_ind comes from trainr or testr and col_ind is the label-encoded app_id.

In [34]:
```
dfm = deviceapps.dropna(subset=['trainr'])
Xtr_app = csr_matrix((np.ones(dfm.shape[0]), (dfm.trainr, dfm.app)),
                      shape=(g_a_train.shape[0],napps))
dfm = deviceapps.dropna(subset=['testr'])
Xte_app = csr_matrix((np.ones(dfm.shape[0]), (dfm.testr, dfm.app)),
                      shape=(g_a_test.shape[0],napps))
print('Apps data: train shape {}, test shape {}'.format(Xtr_app.shape, Xte_
app.shape))
```

```
Apps data: train shape (74645, 19237), test shape (112071, 19237)
```

### App labels features

We can create app labels merging app_labels with the deviceapps dataframe.

In [35]:
```
applabels = applabels.loc[applabels.app_id.isin(appevents.app_id.unique())]
applabels['app'] = apps_encoder.transform(applabels.app_id)
labelencoder = LabelEncoder().fit(applabels.label_id)
applabels['label'] = labelencoder.transform(applabels.label_id)
nlabels = len(labelencoder.classes_)
```

In [36]:
```
devicelabels = (deviceapps[['device_id','app']]
                .merge(applabels[['app','label']])
                .groupby(['device_id','label'])['app'].agg(['size'])
                .merge(g_a_train[['trainr']], how='left', left_index=True,
right_index=True)
                .merge(g_a_test[['testr']], how='left', left_index=True, ri
ght_index=True)
                .reset_index())
```

In [37]:
```
devicelabels.head()
```

Out[37]:

| | device_id | label | size | trainr | testr |
|---|---|---|---|---|---|
| **0** | -9222956879900151005 | 117 | 1 | 21594.0 | NaN |
| **1** | -9222956879900151005 | 120 | 1 | 21594.0 | NaN |
| **2** | -9222956879900151005 | 126 | 1 | 21594.0 | NaN |
| **3** | -9222956879900151005 | 138 | 2 | 21594.0 | NaN |
| **4** | -9222956879900151005 | 147 | 2 | 21594.0 | NaN |

In [38]:
```
dfm = devicelabels.dropna(subset=['trainr'])
Xtr_label = csr_matrix((np.ones(dfm.shape[0]), (dfm.trainr, dfm.label)),
                      shape=(g_a_train.shape[0],nlabels))
dfm = devicelabels.dropna(subset=['testr'])
Xte_label = csr_matrix((np.ones(dfm.shape[0]), (dfm.testr, dfm.label)),
                      shape=(g_a_test.shape[0],nlabels))
print('Labels data: train shape {}, test shape {}'.format(Xtr_label.shape,
Xte_label.shape))
```

```
Labels data: train shape (74645, 492), test shape (112071, 492)
```

Features concatenation

In [39]:
```
Xtrain = hstack((Xtr_brand, Xtr_model, Xtr_app, Xtr_label), format='csr')
Xtest =  hstack((Xte_brand, Xte_model, Xte_app, Xte_label), format='csr')
print('All features: train shape {}, test shape {}'.format(Xtrain.shape, Xt
est.shape))
```

```
All features: train shape (74645, 21527), test shape (112071, 21527)
```

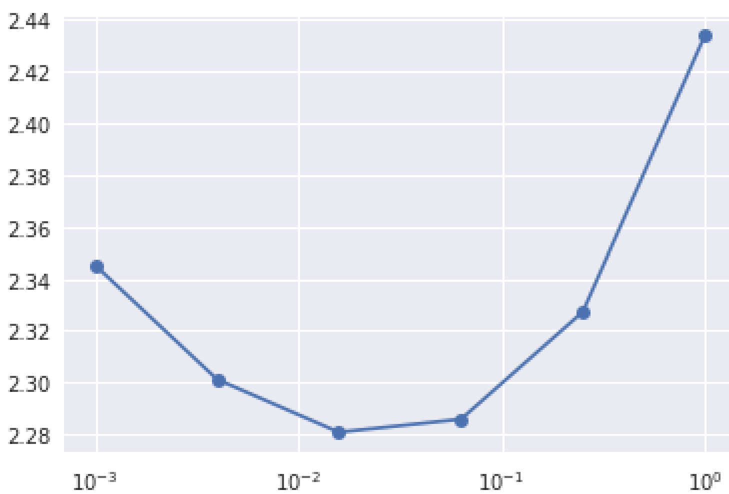
## Performing cross-validation

In [40]:
```
targ_encoder = LabelEncoder().fit(g_a_train.group)
y = targ_encoder.transform(g_a_train.group)
nclasses = len(targ_encoder.classes_)
```

In [42]:
```
# Defining loss- score function
def score(clf, random_state = 0):
    kf = StratifiedKFold(y, n_folds=5, shuffle=True, random_state=random_st
ate)
    pred = np.zeros((y.shape[0],nclasses))
    for itrain, itest in kf:
        Xtr, Xte = Xtrain[itrain, :], Xtrain[itest, :]
        ytr, yte = y[itrain], y[itest]
        clf.fit(Xtr, ytr)
        pred[itest,:] = clf.predict_proba(Xte)

        # Resize to one fold (for kernels)
        return log_loss(yte, pred[itest, :])
        print("{:.5f}".format(log_loss(yte, pred[itest,:])), end=' ')
    print('')
    return log_loss(y, pred)
```

We've tested values for regularization constant C. Since there is probably a lot of columns which are not so important (rare apps or models of brands) we are probably going to get better score with stronger regularization which means that C value will probably going to be below 1.

In [43]:
```
cvalue = np.logspace(-3,0,6)
res = []
for C in cvalue:
    res.append(score(LogisticRegression(C = C)))
plt.semilogx(cvalue, res,'-o');
```




So it looks like the best value for C could between 0.01 and 0.1.

In [44]:
```
score(LogisticRegression(C=0.01))
```

Out[44]: 2.2848755470140127

In [45]:
```
score(LogisticRegression(C=0.02))
```

Out[45]: 2.2797068236722908

In [46]:
```
score(LogisticRegression(C=0.03))
```

Out[46]: 2.2796060828323981

In [47]:
```
score(LogisticRegression(C=0.04))
```

Out[47]: 2.2809556715503021

In [48]:
```
score(LogisticRegression(C=0.05))
```

Out[48]: 2.2828903616369471

LogisticRegression classifier solves multiclass classification problem -in form of one versus rest fashion. But we can also fit a multinomial model that optimizes the multiclass logloss like in our case. We could improve results using this scenario since this is our exact setup.

In [49]:
```
score(LogisticRegression(C=0.02, multi_class='multinomial',solver='saga'))
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/s
ag.py:326: ConvergenceWarning: The max_iter was reached which means the coe
f_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

Out[49]: 2.2733463394827584

In [50]:
```
score(LogisticRegression(C=0.02, multi_class='multinomial',solver='lbfgs'))
```

Out[50]: 2.273326572493398

In [51]:
```
score(LogisticRegression(C=0.02, multi_class='multinomial',solver='newton-c
g'))
```

Out[51]: 2.2731559680466482

In [52]:
```
score(LogisticRegression(C=0.02, multi_class='multinomial',solver='sag'))
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/s
ag.py:326: ConvergenceWarning: The max_iter was reached which means the coe
f_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

Out[52]: 2.2731581798416354

# Benchmark comparison - Test dataset with XGBoost

In [3]:
```
import datetime
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
import xgboost as xgb
```

```
import random
import zipfile
import time
import shutil
from sklearn.metrics import log_loss

random.seed(2016)

def run_xgb(train, test, features, target, random_state=0):
    eta = 0.1
    max_depth = 3
    subsample = 0.7
    colsample_bytree = 0.7
    start_time = time.time()

    print('XGBoost params. ETA: {}, MAX_DEPTH: {}, SUBSAMPLE: {}, COLSAM
PLE_BY_TREE: {}'.format(eta, max_depth, subsample, colsample_bytree))
    params = {
        "objective": "multi:softprob",
        "num_class": 12,
        "booster" : "gbtree",
        "eval_metric": "mlogloss",
        "eta": eta,
        "max_depth": max_depth,
        "subsample": subsample,
        "colsample_bytree": colsample_bytree,
        "silent": 1,
        "seed": random_state,
    }
    num_boost_round = 500
    early_stopping_rounds = 50
    test_size = 0.3

    X_train, X_valid = train_test_split(train, test_size=test_size, rand
om_state=random_state)
    print('Length train:', len(X_train.index))
    print('Length valid:', len(X_valid.index))
    y_train = X_train[target]
    y_valid = X_valid[target]
    dtrain = xgb.DMatrix(X_train[features], y_train)
    dvalid = xgb.DMatrix(X_valid[features], y_valid)

    watchlist = [(dtrain, 'train'), (dvalid, 'eval')]
    gbm = xgb.train(params, dtrain, num_boost_round, evals=watchlist, ea
rly_stopping_rounds=early_stopping_rounds, verbose_eval=True)

    print("Validating...")
    check = gbm.predict(xgb.DMatrix(X_valid[features]), ntree_limit=gbm.
best_iteration)
    score = log_loss(y_valid.tolist(), check)

    print("Predict test set...")
    test_prediction = gbm.predict(xgb.DMatrix(test[features]), ntree_lim
it=gbm.best_iteration)

    print('Training time: {} minutes'.format(round((time.time() - start_
time)/60, 2)))
```

```
    return test_prediction.tolist(), score


def create_submission(score, test, prediction):
    # Make Submission
    now = datetime.datetime.now()
    sub_file = 'submission_' + str(score) + '_' + str(now.strftime("%Y-%
m-%d-%H-%M")) + '.csv'
    print('Writing submission: ', sub_file)
    f = open(sub_file, 'w')
    f.write('device_id,F23-,F24-26,F27-28,F29-32,F33-42,F43+,M22-,M23-2
6,M27-28,M29-31,M32-38,M39+\n')
    total = 0
    test_val = test['device_id'].values
    for i in range(len(test_val)):
        str1 = str(test_val[i])
        for j in range(12):
            str1 += ',' + str(prediction[i][j])
        str1 += '\n'
        total += 1
        f.write(str1)
    f.close()


def map_column(table, f):
    labels = sorted(table[f].unique())
    mappings = dict()
    for i in range(len(labels)):
        mappings[labels[i]] = i
    table = table.replace({f: mappings})
    return table


def read_train_test():
    # Events
    print('Read events data')
    events = pd.read_csv("data/events.csv", dtype={'device_id': np.str})
    events['counts'] = events.groupby(['device_id'])['event_id'].transfo
rm('count')
    events_small = events[['device_id', 'counts']].drop_duplicates('devi
ce_id', keep='first')

    # Phone brand
    print('Read brands data')
    pbd = pd.read_csv("data/phone_brand_device_model.csv", dtype={'devic
e_id': np.str})
    pbd.drop_duplicates('device_id', keep='first', inplace=True)
    pbd = map_column(pbd, 'phone_brand')
    pbd = map_column(pbd, 'device_model')

    # Train
    print('Read training data')
    train = pd.read_csv("data/gender_age_train.csv", dtype={'device_id':
 np.str})
    train = map_column(train, 'group')
    train = train.drop(['age'], axis=1)
    train = train.drop(['gender'], axis=1)
```

```
    train = pd.merge(train, pbd, how='left', on='device_id', left_index=
True)
    train = pd.merge(train, events_small, how='left', on='device_id', le
ft_index=True)
    train.fillna(-1, inplace=True)

    # Test
    print('Read test data')
    test = pd.read_csv("data/gender_age_test.csv", dtype={'device_id': n
p.str})
    test = pd.merge(test, pbd, how='left', on='device_id', left_index=Tr
ue)
    test = pd.merge(test, events_small, how='left', on='device_id', left
_index=True)
    test.fillna(-1, inplace=True)

    # Features
    features = list(test.columns.values)
    features.remove('device_id')

    return train, test, features


train, test, features = read_train_test()
print('Length of train: ', len(train))
print('Length of test: ', len(test))
print('Features [{}]: {}'.format(len(features), sorted(features)))
test_prediction, score = run_xgb(train, test, features, 'group')
print("LS: {}".format(round(score, 5)))
create_submission(score, test, test_prediction)
```

```
Read events data
Read brands data
Read training data
Read test data
Length of train:  74645
Length of test:  112071
Features [3]: ['counts', 'device_model', 'phone_brand']
XGBoost params. ETA: 0.1, MAX_DEPTH: 3, SUBSAMPLE: 0.7, COLSAMPLE_BY_TRE
E: 0.7
Length train: 52251
Length valid: 22394
[0]     train-mlogloss:2.47571  eval-mlogloss:2.47616
Multiple eval metrics have been passed: 'eval-mlogloss' will be used for
early stopping.

Will train until eval-mlogloss hasn't improved in 50 rounds.
[1]     train-mlogloss:2.46755  eval-mlogloss:2.46858
[2]     train-mlogloss:2.46021  eval-mlogloss:2.46177
[3]     train-mlogloss:2.45349  eval-mlogloss:2.45551
[4]     train-mlogloss:2.4475   eval-mlogloss:2.44996
[5]     train-mlogloss:2.4422   eval-mlogloss:2.44512
[6]     train-mlogloss:2.43738  eval-mlogloss:2.44074
[7]     train-mlogloss:2.43297  eval-mlogloss:2.43675
[8]     train-mlogloss:2.42895  eval-mlogloss:2.43312
[9]     train-mlogloss:2.42525  eval-mlogloss:2.42983
[10]    train-mlogloss:2.42203  eval-mlogloss:2.42711
[11]    train-mlogloss:2.41906  eval-mlogloss:2.4245
[12]    train-mlogloss:2.41634  eval-mlogloss:2.42212
[13]    train-mlogloss:2.41398  eval-mlogloss:2.42011
[14]    train-mlogloss:2.41178  eval-mlogloss:2.41839
[15]    train-mlogloss:2.40971  eval-mlogloss:2.41665
[16]    train-mlogloss:2.40777  eval-mlogloss:2.41508
[17]    train-mlogloss:2.40596  eval-mlogloss:2.41359
[18]    train-mlogloss:2.40433  eval-mlogloss:2.41223
[19]    train-mlogloss:2.40279  eval-mlogloss:2.41103
[20]    train-mlogloss:2.40146  eval-mlogloss:2.41
[21]    train-mlogloss:2.40015  eval-mlogloss:2.40903
[22]    train-mlogloss:2.3989   eval-mlogloss:2.4081
[23]    train-mlogloss:2.39779  eval-mlogloss:2.40738
[24]    train-mlogloss:2.39677  eval-mlogloss:2.40656
[25]    train-mlogloss:2.39576  eval-mlogloss:2.40586
[26]    train-mlogloss:2.39496  eval-mlogloss:2.40534
[27]    train-mlogloss:2.39421  eval-mlogloss:2.40485
[28]    train-mlogloss:2.39336  eval-mlogloss:2.40437
[29]    train-mlogloss:2.39266  eval-mlogloss:2.40399
[30]    train-mlogloss:2.39194  eval-mlogloss:2.40356
[31]    train-mlogloss:2.39125  eval-mlogloss:2.40323
[32]    train-mlogloss:2.39069  eval-mlogloss:2.40295
[33]    train-mlogloss:2.39008  eval-mlogloss:2.40264
[34]    train-mlogloss:2.38951  eval-mlogloss:2.40238
[35]    train-mlogloss:2.38893  eval-mlogloss:2.40205
[36]    train-mlogloss:2.38842  eval-mlogloss:2.40175
[37]    train-mlogloss:2.38799  eval-mlogloss:2.40155
[38]    train-mlogloss:2.38752  eval-mlogloss:2.4013
[39]    train-mlogloss:2.38705  eval-mlogloss:2.40119
[40]    train-mlogloss:2.38662  eval-mlogloss:2.40101
[41]    train-mlogloss:2.38616  eval-mlogloss:2.4008
```

```
[42]	train-mlogloss:2.38566	eval-mlogloss:2.40061
[43]	train-mlogloss:2.3853	eval-mlogloss:2.4005
[44]	train-mlogloss:2.38492	eval-mlogloss:2.40039
[45]	train-mlogloss:2.38458	eval-mlogloss:2.40023
[46]	train-mlogloss:[illegible]	eval-mlogloss:2.39999
[47]	train-mlogloss:2.38384	eval-mlogloss:2.3999
[48]	train-mlogloss:2.38345	eval-mlogloss:2.39978
[49]	train-mlogloss:2.38313	eval-mlogloss:2.3997
[50]	train-mlogloss:2.38285	eval-mlogloss:2.39964
[51]	train-mlogloss:2.38251	eval-mlogloss:2.39948
[52]	train-mlogloss:2.38223	eval-mlogloss:2.39946
[53]	train-mlogloss:2.38196	eval-mlogloss:2.39935
[54]	train-mlogloss:2.38165	eval-mlogloss:2.39927
[55]	train-mlogloss:2.38142	eval-mlogloss:2.39926
[56]	train-mlogloss:2.38119	eval-mlogloss:2.39922
[57]	train-mlogloss:2.38091	eval-mlogloss:2.39914
[58]	train-mlogloss:2.38063	eval-mlogloss:2.39903
[59]	train-mlogloss:2.3804	eval-mlogloss:2.39901
[60]	train-mlogloss:2.3801	eval-mlogloss:2.39894
[61]	train-mlogloss:2.37987	eval-mlogloss:2.39894
[62]	train-mlogloss:[illegible]	eval-mlogloss:[illegible]
[63]	train-mlogloss:2.37938	eval-mlogloss:2.3989
[64]	train-mlogloss:2.37915	eval-mlogloss:2.39888
[65]	train-mlogloss:2.37884	eval-mlogloss:2.39883
[66]	train-mlogloss:2.37856	eval-mlogloss:2.39872
[67]	train-mlogloss:[illegible]	eval-mlogloss:[illegible]
[68]	train-mlogloss:2.37805	eval-mlogloss:2.39873
[69]	[illegible]
[70]	train-mlogloss:2.37756	eval-mlogloss:2.39863
[71]	train-mlogloss:2.37727	eval-mlogloss:2.39859
[72]	train-mlogloss:2.37709	eval-mlogloss:2.39859
[73]	train-mlogloss:2.37683	eval-mlogloss:2.3985
[74]	train-mlogloss:2.37658	eval-mlogloss:2.39848
[75]	train-mlogloss:2.37637	eval-mlogloss:2.3985
[76]	train-mlogloss:2.37612	eval-mlogloss:2.39845
[77]	train-mlogloss:2.37586	eval-mlogloss:2.39835
[78]	train-mlogloss:2.37559	eval-mlogloss:2.39828
[79]	train-mlogloss:2.37541	eval-mlogloss:2.39824
[80]	train-mlogloss:2.37517	eval-mlogloss:2.3982
[81]	train-mlogloss:2.37495	eval-mlogloss:2.3982
[82]	train-mlogloss:2.37476	eval-mlogloss:2.39816
[83]	train-mlogloss:2.37454	eval-mlogloss:2.39813
[84]	train-mlogloss:2.37434	eval-mlogloss:2.39811
[85]	train-mlogloss:2.37404	eval-mlogloss:2.39805
[86]	train-mlogloss:2.37378	eval-mlogloss:2.39802
[87]	train-mlogloss:2.37358	eval-mlogloss:2.39799
[88]	train-mlogloss:2.37338	eval-mlogloss:2.39797
[89]	train-mlogloss:2.37319	eval-mlogloss:2.39797
[90]	train-mlogloss:2.37302	eval-mlogloss:2.39799
[91]	train-mlogloss:2.37283	eval-mlogloss:2.39794
[92]	train-mlogloss:2.37264	eval-mlogloss:2.3979
[93]	train-mlogloss:2.37243	eval-mlogloss:2.3978
[94]	train-mlogloss:2.37219	eval-mlogloss:2.39781
[95]	train-mlogloss:2.37197	eval-mlogloss:2.39775
[96]	train-mlogloss:2.37179	eval-mlogloss:2.3977
[97]	train-mlogloss:2.37162	eval-mlogloss:2.39776
[98]	train-mlogloss:2.37146	eval-mlogloss:2.39773
```

## Test data predictions : LogisticRegression

In [52]:
```
clf = LogisticRegression(C=0.02, multi_class='multinomial',solver='lbfgs'
clf.fit(Xtrain, y)
pred = pd.DataFrame(clf.predict_proba(Xtest), index = g_a_test.index, col
ns=targ_encoder.classes_)
```

In [53]:
```
pred.head()
```

Out[53]:

| | F23- | F24-26 | F27-28 | F29-32 | F33-42 | F43+ | M22 |
|---|---|---|---|---|---|---|---|
| **device_id** | | | | | | | |
| **1002079943728939269** | 0.001424 | 0.005998 | 0.013605 | 0.013286 | 0.025313 | 0.046103 | 0.01 |
| **-1547860181818787117** | 0.007414 | 0.013299 | 0.031228 | 0.058677 | 0.072686 | 0.151391 | 0.00 |
| **7374582448058474277** | 0.023158 | 0.036713 | 0.036233 | 0.158343 | 0.162774 | 0.079852 | 0.01 |
| **-6220210354783429585** | 0.003474 | 0.030860 | 0.008801 | 0.012351 | 0.050697 | 0.172943 | 0.04 |
| **-5893464122623104785** | 0.046982 | 0.065640 | 0.042578 | 0.062522 | 0.056329 | 0.043467 | 0.09 |

Storing best predictions in CSV file

In [54]:
```
pred.to_csv('predictions.csv',index=True)
```

# Free form data visualizations

```
[99]    train-mlogloss:2.37127  eval-mlogloss:2.39769
[100]   train-mlogloss:2.37106  eval-mlogloss:2.39767
[101]   train-mlogloss:2.37085  eval-mlogloss:2.39762
[102]   train-mlogloss:2.37061  eval-mlogloss:2.39756
[103]   train-mlogloss:2.37044  eval-mlogloss:2.39752
[104]   train-mlogloss:2.37023  eval-mlogloss:2.39745
[105]   train-mlogloss:2.37003  eval-mlogloss:2.39748
[106]   train-mlogloss:2.36982  eval-mlogloss:2.39747
[107]   train-mlogloss:2.36964  eval-mlogloss:2.39754
[108]   train-mlogloss:2.36945  eval-mlogloss:2.39757
[109]   train-mlogloss:2.36928  eval-mlogloss:2.3976
[110]   train-mlogloss:2.36908  eval-mlogloss:2.39755
[111]   train-mlogloss:2.36892  eval-mlogloss:2.3976
[112]   train-mlogloss:2.3687   eval-mlogloss:2.3976
[113]   train-mlogloss:2.36848  eval-mlogloss:2.39758
[114]   train-mlogloss:2.36834  eval-mlogloss:2.39757
[115]   train-mlogloss:2.36818  eval-mlogloss:2.39755
[116]   train-mlogloss:2.36802  eval-mlogloss:2.39754
[117]   train-mlogloss:2.36782  eval-mlogloss:2.3975
[118]   train-mlogloss:2.36767  eval-mlogloss:2.39746
[119]   train-mlogloss:2.36749  eval-mlogloss:2.39745
[120]   train-mlogloss:2.36731  eval-mlogloss:2.39745
[121]   train-mlogloss:2.36714  eval-mlogloss:2.39746
[122]   train-mlogloss:2.36695  eval-mlogloss:2.39748
[123]   train-mlogloss:2.36675  eval-mlogloss:2.39747
[124]   train-mlogloss:2.36655  eval-mlogloss:2.39744
[125]   train-mlogloss:2.36637  eval-mlogloss:2.39743
[126]   train-mlogloss:2.36616  eval-mlogloss:2.39735
[127]   train-mlogloss:2.36599  eval-mlogloss:2.39735
[128]   train-mlogloss:2.3658   eval-mlogloss:2.39723
[129]   train-mlogloss:2.36561  eval-mlogloss:2.39722
[130]   train-mlogloss:2.36546  eval-mlogloss:2.39725
[131]   train-mlogloss:2.3653   eval-mlogloss:2.39726
[132]   train-mlogloss:2.36514  eval-mlogloss:2.39721
[133]   train-mlogloss:2.36495  eval-mlogloss:2.39721
[134]   train-mlogloss:2.36482  eval-mlogloss:2.3972
[135]   train-mlogloss:2.36462  eval-mlogloss:2.3972
[136]   train-mlogloss:2.36445  eval-mlogloss:2.39715
[137]   train-mlogloss:2.36428  eval-mlogloss:2.39711
[138]   train-mlogloss:2.36414  eval-mlogloss:2.3971
[139]   train-mlogloss:2.36397  eval-mlogloss:2.39709
[140]   train-mlogloss:2.36381  eval-mlogloss:2.39709
[141]   train-mlogloss:2.36361  eval-mlogloss:2.39706
[142]   train-mlogloss:2.36345  eval-mlogloss:2.39703
[143]   train-mlogloss:2.3633   eval-mlogloss:2.39701
[144]   train-mlogloss:2.36314  eval-mlogloss:2.39705
[145]   train-mlogloss:2.36295  eval-mlogloss:2.39705
[146]   train-mlogloss:2.36276  eval-mlogloss:2.39704
[147]   train-mlogloss:2.36259  eval-mlogloss:2.39703
[148]   train-mlogloss:2.36245  eval-mlogloss:2.39702
[149]   train-mlogloss:2.36227  eval-mlogloss:2.39705
[150]   train-mlogloss:2.3621   eval-mlogloss:2.39707
[151]   train-mlogloss:2.36197  eval-mlogloss:2.39708
[152]   train-mlogloss:2.36181  eval-mlogloss:2.39709
[153]   train-mlogloss:2.36163  eval-mlogloss:2.39701
[154]   train-mlogloss:2.36144  eval-mlogloss:2.39696
[155]   train-mlogloss:2.36125  eval-mlogloss:2.39694
```

In [4]:

```
__author__ = 'ZFTurbo: https://kaggle.com/zfturbo'
__author__ = 'tilii: https://kaggle.com/tilii7'

# ZFTurbo defined first 3 features
# tilii added two new features and t-SNE clustering & visualization
# used some ideas from https://www.kaggle.com/cast42/santander-customer-satisfaction/t-sne-manifold-visualisation/code

import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn import manifold
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.preprocessing import normalize
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

```
[156]	train-mlogloss:2.36109	eval-mlogloss:2.39695
[157]	train-mlogloss:2.36094	eval-mlogloss:2.39694
[158]	train-mlogloss:2.36082	eval-mlogloss:2.39694
[159]	train-mlogloss:2.36066	eval-mlogloss:2.39695
[160]	train-mlogloss:2.36048	eval-mlogloss:2.39693
[161]	train-mlogloss:2.36034	eval-mlogloss:2.39697
[162]	train-mlogloss:2.36019	eval-mlogloss:2.397
[163]	train-mlogloss:2.36004	eval-mlogloss:2.39697
[164]	train-mlogloss:2.35991	eval-mlogloss:2.39692
[165]	train-mlogloss:2.35969	eval-mlogloss:2.39686
[166]	train-mlogloss:2.35956	eval-mlogloss:2.39684
[167]	train-mlogloss:2.35937	eval-mlogloss:2.39677
[168]	train-mlogloss:2.35923	eval-mlogloss:2.39677
[169]	train-mlogloss:2.3591	eval-mlogloss:2.39674
[170]	train-mlogloss:2.35895	eval-mlogloss:2.39674
[171]	train-mlogloss:2.3588	eval-mlogloss:2.39678
[172]	train-mlogloss:2.35865	eval-mlogloss:2.39673
[173]	train-mlogloss:2.35851	eval-mlogloss:2.39672
[174]	train-mlogloss:2.35834	eval-mlogloss:2.3967
[175]	train-mlogloss:2.35814	eval-mlogloss:2.39672
[176]	train-mlogloss:2.358	eval-mlogloss:2.39669
[177]	train-mlogloss:2.35784	eval-mlogloss:2.39669
[178]	train-mlogloss:2.35769	eval-mlogloss:2.39669
[179]	train-mlogloss:2.35752	eval-mlogloss:2.39667
[180]	train-mlogloss:2.35738	eval-mlogloss:2.39665
[181]	train-mlogloss:2.35726	eval-mlogloss:2.39668
[182]	train-mlogloss:2.35709	eval-mlogloss:2.39667
[183]	train-mlogloss:2.35693	eval-mlogloss:2.39665
[184]	train-mlogloss:2.35681	eval-mlogloss:2.39667
[185]	train-mlogloss:2.35667	eval-mlogloss:2.39667
[186]	train-mlogloss:2.35651	eval-mlogloss:2.3967
[187]	train-mlogloss:2.35635	eval-mlogloss:2.39672
[188]	train-mlogloss:2.35623	eval-mlogloss:2.39672
[189]	train-mlogloss:2.35603	eval-mlogloss:2.3967
[190]	train-mlogloss:2.35588	eval-mlogloss:2.39668
[191]	train-mlogloss:2.35574	eval-mlogloss:2.39667
[192]	train-mlogloss:2.35562	eval-mlogloss:2.39665
[193]	train-mlogloss:2.35547	eval-mlogloss:2.39668
[194]	train-mlogloss:2.35536	eval-mlogloss:2.39669
[195]	train-mlogloss:2.35523	eval-mlogloss:2.39669
[196]	train-mlogloss:2.3551	eval-mlogloss:2.39671
[197]	train-mlogloss:2.35497	eval-mlogloss:2.39676
[198]	train-mlogloss:2.35484	eval-mlogloss:2.39675
[199]	train-mlogloss:2.35468	eval-mlogloss:2.39677
[200]	train-mlogloss:2.35456	eval-mlogloss:2.39679
[201]	train-mlogloss:2.35444	eval-mlogloss:2.39681
[202]	train-mlogloss:2.35434	eval-mlogloss:2.39683
[203]	train-mlogloss:2.35421	eval-mlogloss:2.39686
[204]	train-mlogloss:2.35405	eval-mlogloss:2.39683
[205]	train-mlogloss:2.35391	eval-mlogloss:2.39683
[206]	train-mlogloss:2.35373	eval-mlogloss:2.39683
[207]	train-mlogloss:2.3536	eval-mlogloss:2.39679
[208]	train-mlogloss:2.35346	eval-mlogloss:2.39681
[209]	train-mlogloss:2.35334	eval-mlogloss:2.39682
[210]	train-mlogloss:2.35321	eval-mlogloss:2.39685
[211]	train-mlogloss:2.35307	eval-mlogloss:2.39688
[212]	train-mlogloss:2.35293	eval-mlogloss:2.39695
```

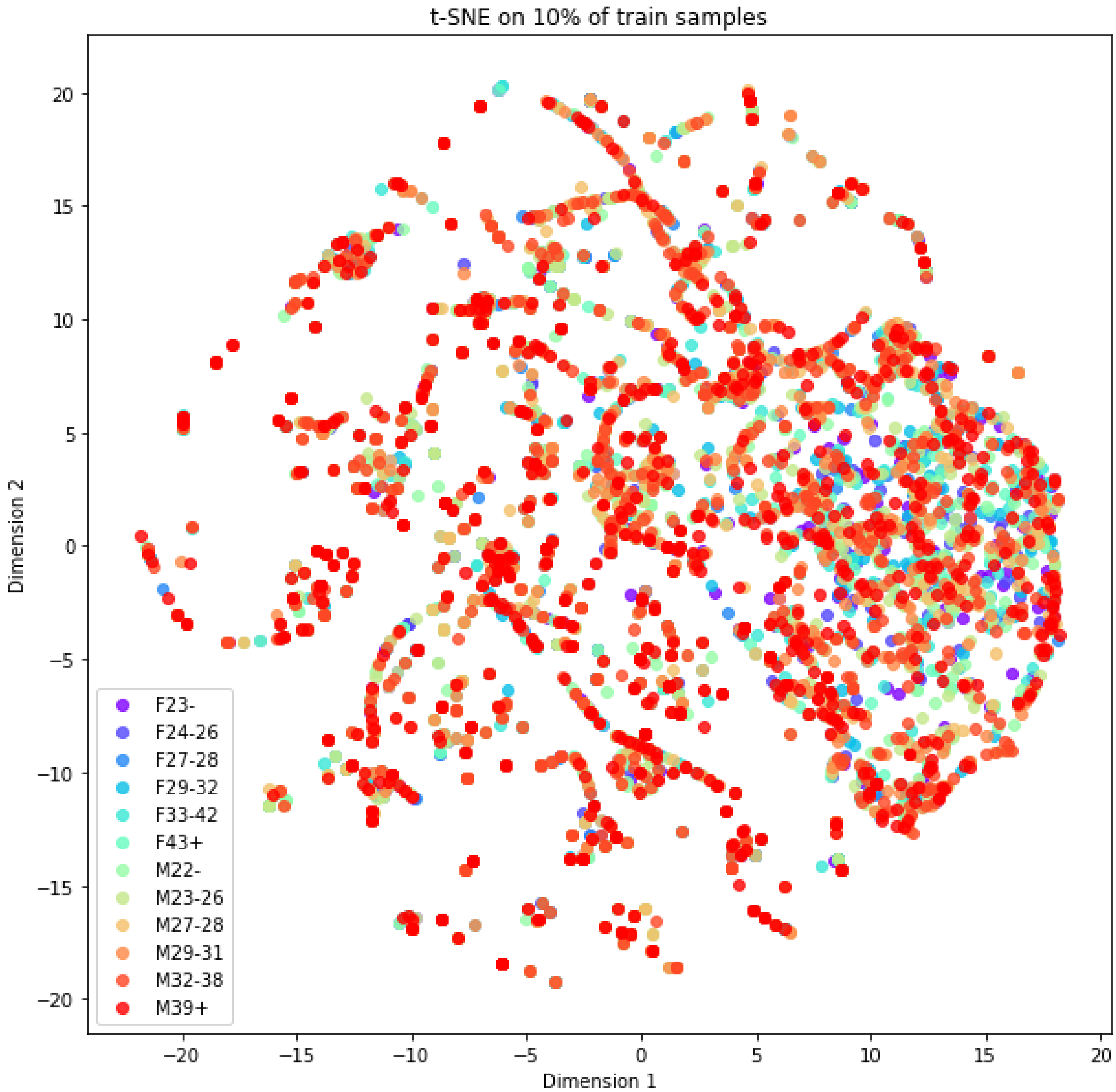
```
def run_tsne(train, target):
    sss = StratifiedShuffleSplit(target, test_size=0.1)
    for train_index, test_index in sss:
        break

    X_train, X_valid = train[train_index], train[test_index]
    Y_train, Y_valid = target[train_index], target[test_index]

    train_norm = normalize(X_valid, axis=0)
    tsne = manifold.TSNE(n_components=3,
                         init='pca',
                         random_state=101,
                         method='barnes_hut',
                         n_iter=500,
                         verbose=2)
    train_tsne = tsne.fit_transform(train_norm)
    return (train_tsne, Y_valid)

def tsne_vis(tsne_data, tsne_groups):
    colors = cm.rainbow(np.linspace(0, 1, 12))
    labels = ['F23-', 'F24-26', 'F27-28', 'F29-32', 'F33-42', 'F43+', 'M
22-',
              'M23-26', 'M27-28', 'M29-31', 'M32-38', 'M39+']

    plt.figure(figsize=(10, 10))
    for l, c, co, in zip(labels, colors, range(12)):
        plt.scatter(tsne_data[np.where(tsne_groups == co), 0],
                    tsne_data[np.where(tsne_groups == co), 1],
                    marker='o',
                    color=c,
                    linewidth='1',
                    alpha=0.8,
                    label=l)
    plt.xlabel("Dimension 1")
    plt.ylabel("Dimension 2")
    plt.title("t-SNE on 10% of train samples")
    plt.legend(loc='best')
    plt.savefig('rainbow-01.png')
    plt.show(block=False)

    plt.figure(figsize=(10, 10))
    for l, c, co, in zip(labels, colors, range(12)):
        plt.scatter(tsne_data[np.where(tsne_groups == co), 0],
                    tsne_data[np.where(tsne_groups == co), 2],
                    marker='o',
                    color=c,
                    linewidth='1',
                    alpha=0.8,
                    label=l)
    plt.xlabel("Dimension 1")
    plt.ylabel("Dimension 3")
    plt.title("t-SNE on 10% of train samples")
    plt.legend(loc='best')
    plt.savefig('rainbow-02.png')
    plt.show(block=False)
```

```
[213]   train-mlogloss:2.35278  eval-mlogloss:2.39693
[214]   train-mlogloss:2.35267  eval-mlogloss:2.39699
[215]   train-mlogloss:2.35254  eval-mlogloss:2.39697
[216]   train-mlogloss:2.35238  eval-mlogloss:2.39695
[217]   train-mlogloss:2.35225  eval-mlogloss:2.39696
[218]   train-mlogloss:2.35213  eval-mlogloss:2.39696
[219]   train-mlogloss:2.352[illegible]  eval-mlogloss:2.39695
[220]   train-mlogloss:2.35188  eval-mlogloss:2.39692
[221]   train-mlogloss:2.35174  eval-mlogloss:2.3969
[222]   train-mlogloss:2.3516   eval-mlogloss:2.39691
[223]   train-mlogloss:2.35149  eval-mlogloss:2.3969
[224]   train-mlogloss:2.35135  eval-mlogloss:2.39688
[225]   train-mlogloss:2.35124  eval-mlogloss:2.39692
[226]   train-mlogloss:2.35109  eval-mlogloss:2.39691
[227]   train-mlogloss:2.35096  eval-mlogloss:2.39689
[228]   train-mlogloss:2.35086  eval-mlogloss:2.3969
[229]   train-mlogloss:2.35074  eval-mlogloss:2.39691
[230]   train-mlogloss:2.3506   eval-mlogloss:2.39689
[231]   train-mlogloss:2.35045  eval-mlogloss:2.39689
[232]   train-mlogloss:2.35034  eval-mlogloss:2.39696
[233]   train-mlogloss:2.35022  eval-mlogloss:2.39699
[234]   train-mlogloss:2.35009  eval-mlogloss:2.39699
[235]   train-mlogloss:2.34997  eval-mlogloss:2.39703
[236]   train-mlogloss:2.34983  eval-mlogloss:2.39706
[237]   train-mlogloss:2.34971  eval-mlogloss:2.39705
[238]   train-mlogloss:2.34959  eval-mlogloss:2.39707
[239]   train-mlogloss:2.34945  eval-mlogloss:2.39705
[240]   train-mlogloss:2.34931  eval-mlogloss:2.39705
[241]   train-mlogloss:2.34916  eval-mlogloss:2.39703
[242]   train-mlogloss:2.34907  eval-mlogloss:2.39702
Stopping. Best iteration:
[192]   train-mlogloss:2.35562  eval-mlogloss:2.39665

Validating...
Predict test set...
Training time: 0.45 minutes
LS: 2.39667
Writing submission:  submission_2.39667065647_2017-12-07-17-58.csv
```

```python
plt.figure(figsize=(12,10))
for l, c, co, in zip(labels, colors, range(12)):
    plt.scatter(tsne_data[np.where(tsne_groups == co), 1],
                tsne_data[np.where(tsne_groups == co), 2],
                marker='o',
                color=c,
                linewidth='1',
                alpha=0.8,
                label=l)
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('t-SNE on 20% of train samples')
plt.legend(loc='best')
plt.savefig('clusters.png')
plt.show(block=False)

def map_column(table, f):
    labels = sorted(table[f].unique())
    mappings = dict()
    for i in range(len(labels)):
        mappings[labels[i]] = i
    table = table.replace({f: mappings})
    return table

def read_train_test():
    # App events
    print('\nReading app events...')
    ape = pd.read_csv('data/app_events.csv')
    ape['installed'] = ape.groupby(
        ['event_id'])['is_installed'].transform('sum')
    ape['active'] = ape.groupby(
        ['event_id'])['is_active'].transform('sum')
    ape.drop(['is_installed', 'is_active'], axis=1, inplace=True)
    ape.drop_duplicates('event_id', keep='first', inplace=True)
    ape.drop(['app_id'], axis=1)

    # Events
    print('Reading events...')
    events = pd.read_csv('data/events.csv', dtype={'device_id': np.str})
    events['counts'] = events.groupby(
        ['device_id'])['event_id'].transform('count')

    print('Making events features...')
    # The idea here is to count the number of installed apps using the d
ata
    # from app_events.csv above. Also to count the number of active app
s.
    events = pd.merge(events, ape, how='left', on='event_id', left_index
=True)

    # Below is the original events_small table
    # events_small = events[['device_id', 'counts']].drop_duplicates('de
vice_id', keep='first')
    # And this is the new events_small table with two extra features
    events_small = events[['device_id', 'counts', 'installed',
```

```
                              'active']].drop_duplicates('device_id',
                                                          keep='first')

    # Phone brand
    print('Reading phone brands...')
    pbd = pd.read_csv('data/phone_brand_device_model.csv',
                      dtype={'device_id': np.str})
    pbd.drop_duplicates('device_id', keep='first', inplace=True)
    pbd = map_column(pbd, 'phone_brand')
    pbd = map_column(pbd, 'device_model')

    # Train
    print('Reading train data...')
    train = pd.read_csv('data/gender_age_train.csv',
                        dtype={'device_id': np.str})
    train = map_column(train, 'group')
    train = train.drop(['age'], axis=1)
    train = train.drop(['gender'], axis=1)
    print('Merging features with train data...')
    train = pd.merge(train, pbd, how='left', on='device_id', left_index=
True)
    train = pd.merge(train,
                     events_small,
                     how='left',
                     on='device_id',
                     left_index=True)
    train.fillna(-1, inplace=True)

    # Test
    print('Reading test data...')
    test = pd.read_csv('data/gender_age_test.csv',
                       dtype={'device_id': np.str})
    print('Merging features with test data...\n')
    test = pd.merge(test, pbd, how='left', on='device_id', left_index=Tr
ue)
    test = pd.merge(test,
                    events_small,
                    how='left',
                    on='device_id',
                    left_index=True)
    test.fillna(-1, inplace=True)

    # Features
    features = list(test.columns.values)
    features.remove('device_id')
    return train, test, features


train, test, features = read_train_test()
print('Length of train: ', len(train))
print('Length of test: ', len(test))
print('Features [{}]: {}\n'.format(len(features), sorted(features)))
train_df = pd.DataFrame(data=train)
X = train_df.drop(['group', 'device_id'], axis=1).values
Y = train_df['group'].values
tsne_data, tsne_groups = run_tsne(X, Y)
tsne_vis(tsne_data, tsne_groups)
```

```
Reading app events...
Reading events...
Making events features...
Reading phone brands...
Reading train data...
Merging features with train data...
Reading test data...
Merging features with test data...

Length of train:  74645
Length of test:  112071
Features [5]: ['active', 'counts', 'device_model', 'installed', 'phone_bran
d']

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 7465 samples in 0.024s...
[t-SNE] Computed neighbors for 7465 samples in 0.180s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7465
[t-SNE] Computed conditional probabilities for sample 2000 / 7465
[t-SNE] Computed conditional probabilities for sample 3000 / 7465
[t-SNE] Computed conditional probabilities for sample 4000 / 7465
[t-SNE] Computed conditional probabilities for sample 5000 / 7465
[t-SNE] Computed conditional probabilities for sample 6000 / 7465
[t-SNE] Computed conditional probabilities for sample 7000 / 7465
[t-SNE] Computed conditional probabilities for sample 7465 / 7465
[t-SNE] Mean sigma: 0.000000
[t-SNE] Computed conditional probabilities in 0.793s
[t-SNE] Iteration 50: error = 67.1231308, gradient norm = 0.0067935 (50 ite
rations in 17.044s)
[t-SNE] Iteration 100: error = 59.3737221, gradient norm = 0.0023253 (50 it
erations in 13.042s)
[t-SNE] Iteration 150: error = 56.5824242, gradient norm = 0.0015247 (50 it
erations in 12.257s)
[t-SNE] Iteration 200: error = 55.1853828, gradient norm = 0.0010795 (50 it
erations in 12.140s)
[t-SNE] Iteration 250: error = 54.3001022, gradient norm = 0.0007644 (50 it
erations in 12.089s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 54.3001
02
[t-SNE] Iteration 300: error = 1.1290563, gradient norm = 0.0007107 (50 ite
rations in 13.919s)
[t-SNE] Iteration 350: error = 0.6922646, gradient norm = 0.0003353 (50 ite
rations in 14.901s)
[t-SNE] Iteration 400: error = 0.4979769, gradient norm = 0.0001582 (50 ite
rations in 15.257s)
[t-SNE] Iteration 450: error = 0.3815096, gradient norm = 0.0000979 (50 ite
rations in 15.627s)
[t-SNE] Iteration 500: error = 0.3082786, gradient norm = 0.0000712 (50 ite
rations in 15.849s)
[t-SNE] Error after 500 iterations: 0.308279
```


t-SNE on 10% of train samples
Dimension 2
Dimension 1
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20
F23-
F24-26
F27-28
F29-32
F33-42
F43+
M22-
M23-26
M27-28
M29-31
M32-38
M39+