Talking Data User Demographics ML Self Case Study

1. Business Problem

1.1. Problem Overview

Nothing is more comforting than being greeted by your favorite drink just as you walk through the door of the corner café. While a thoughtful barista knows you take a macchiato every Wednesday morning at 8:15, it's much more difficult in a digital space for your preferred brands to personalize your experience.

TalkingData, China's largest third-party mobile data platform, understands that everyday choices and behaviors paint a picture of who we are and what we value. Currently, TalkingData is seeking to leverage behavioral data from more than 70% of the 500 million mobile devices active daily in China to help its clients better understand and interact with their audiences.

In this competition, Kagglers are challenged to build a model predicting users' demographic characteristics based on their app usage, geolocation, and mobile device properties. Doing so will help millions of developers and brand advertisers around the world pursue data-driven marketing efforts which are relevant to their users and catered to their preferences.

Source:https://www.kaggle.com/c/talkingdata-mobile-user-demographics/overview (https://www.kaggle.com/c/talkingdata-mobile-user-demographics/overview)

1.2. Real-world/Business objectives and constraints

- No low-latency requirement.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data Overview

The Data is collected from TalkingData SDK integrated within mobile apps TalkingData serves under the service term between TalkingData and mobile app developers.

Source: https://www.kaggle.com/c/talkingdata-mobile-user-demographics/data (https://www.kaggle.com/c/talkingdata-mobile-user-demographics/data)

1. **gender_age_train.csv, gender_age_test.csv** - the training and test set group: this is the

target variable we are going to predict
2. **events.csv, app_events.csv** - when a user uses TalkingData SDK, the event gets logged in this data. Each event has an event id, location (lat/long), and the event corresponds to a list of apps in app_events. timestamp: when the user is using an app with TalkingData SDK
3. **app_labels.csv** - apps and their labels, the label_id's can be used to join with label_categories
4. **label_categories.csv** - apps' labels and their categories in text
5. **phone_brand_device_model.csv** - device ids, brand, and models phone_brand: note that the brands are in Chinese (translation courtesy of user fromandto)

- 三星 samsung
- 天语 Ktouch
- 海信 hisense
- 联想 lenovo
- 欧比 obi
- 爱派尔 ipair
- 努比亚 nubia
- 优米 youmi
- 朵唯 dowe
- 黑米 heymi
- 锤子 hammer
- 酷比魔方 koobee
- 美图 meitu
- 尼比鲁 nibilu
- 一加 oneplus
- 优购 yougo
- 诺基亚 nokia
- 糖葫芦 candy
- 中国移动 ccmc
- 语信 yuxin
- 基伍 kiwu
- 青橙 greeno
- 华硕 asus
- 夏新 panosonic
- 维图 weitu
- 艾优尼 aiyouni
- 摩托罗拉 moto
- 乡米 xiangmi
- 米奇 micky
- 大可乐 bigcola
- 沃普丰 wpf
- 神舟 hasse
- 摩乐 mole
- 飞秒 fs
- 米歌 mige
- 富可视 fks
- 德赛 desci
- 梦米 mengmi
- 乐视 lshi

- 小杨树 smallt
- 纽曼 newman
- 邦华 banghua
- E派 epai
- 易派 epai
- 普耐尔 pner
- 欧新 ouxin
- 西米 ximi
- 海尔 haier
- 波导 bodao
- 糯米 nuomi
- 唯米 weimi
- 酷珀 kupo
- 谷歌 google
- 昂达 ada
- 聆韵 lingyun

2.2 Mapping Real-World Problem to Machine-Learning Problem

2.2.1. Type of ML Problem

1. It is a Multi-Class Classification Problem.
2. The 12 classes to predict are:
    - F23-
    - F24-26
    - F27-28
    - F29-32
    - F33-42
    - F43+
    - M22-
    - M23-26
    - M27-28
    - M29-31
    - M32-38
    - M39+

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/talkingdata-mobile-user-demographics/overview/evaluation (https://www.kaggle.com/c/talkingdata-mobile-user-demographics/overview/evaluation)

1. Multi-class Log-loss
2. Confusion Matrix

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the 12 classes.

Constraints:

- Class probabilities are needed.
- No Latency constraints.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import os
import re
import time
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from collections import Counter
from scipy.sparse import hstack
from scipy.sparse import csr_matrix
import warnings
warnings.filterwarnings("ignore")
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV
from tqdm import tqdm
import joblib
from sklearn.externals import joblib as jobl
from joblib import dump
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, BatchNormalization,Input,PR
from keras.utils import np_utils
from keras.optimizers import Adam
from keras.models import Model
from keras.optimizers import Adagrad
import datetime
from keras.models import load_model
from IPython.display import Image
from keras.callbacks import EarlyStopping,TensorBoard
```

```
Using TensorFlow backend.
```

3. Exploratory Data Analysis

3.1. Gender Age Data

In [2]:
```
gender_age_data_train = pd.read_csv('Datasets/gender_age_train.csv')
```

In [3]:
```
gender_age_data_train.shape
```

Out[3]: (74645, 4)

In [4]:
```
gender_age_data_train.head()
```

Out[4]:

	device_id	gender	age	group
0	-8076087639492063270	M	35	M32-38
1	-2897161552818060146	M	35	M32-38
2	-8260683887967679142	M	35	M32-38
3	-4938849341048082022	M	30	M29-31
4	245133531816851882	M	30	M29-31

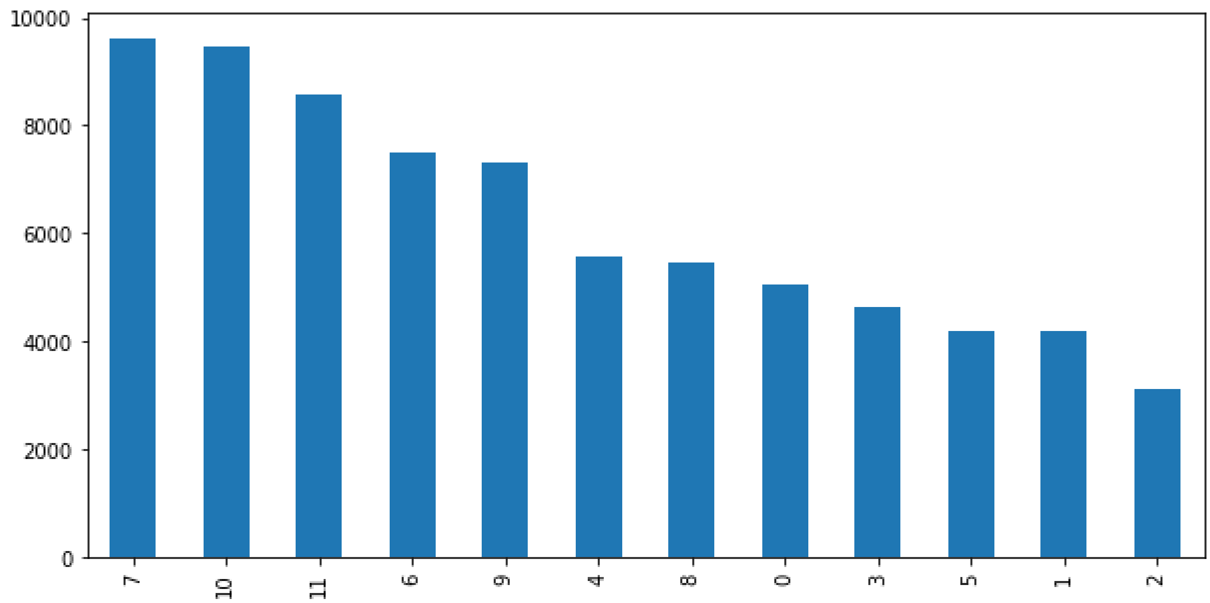
In [5]:
```
#Age group Count
gender_age_data_train['group'].value_counts()
```

Out[5]:
```
M23-26    9605
M32-38    9476
M39+      8581
M22-      7488
M29-31    7309
F33-42    5561
M27-28    5445
F23-      5050
F29-32    4628
F43+      4194
F24-26    4190
F27-28    3118
Name: group, dtype: int64
```

3.1.1. Bar Plot for Group Counts

In [178]:
```
plt.figure(figsize=(10,5))
gender_age_data_train['group'].value_counts().plot(kind='bar')
```

Out[178]: <matplotlib.axes._subplots.AxesSubplot at 0x23637cc2710>



Observation

Here we can conclude that the dataset is imbalanced between male and female. There are more number of males than females

In [7]:
```
#Encoding brands with numeric Label for confusion matrix
group_encoder = LabelEncoder().fit(gender_age_data_train['group'])
gender_age_data_train['group'] = group_encoder.transform(gender_age_data_train['g
```

In [8]:
```
gender_age_data_train.head()
```

Out[8]:

	device_id	gender	age	group
0	-8076087639492063270	M	35	10
1	-2897161552818060146	M	35	10
2	-8260683887967679142	M	35	10
3	-4938849341048082022	M	30	9
4	245133531816851882	M	30	9

3.1.2. Check For Gender Imbalance

In [9]:
```
gender_count = gender_age_data_train['gender'].value_counts()
```

In [10]:
```
gender_count
```

Out[10]:
```
M    47904
F    26741
Name: gender, dtype: int64
```

In [11]:
```
male_percent = gender_count[1]/gender_age_data_train.shape[0]*100
female_percent = gender_count[0]/gender_age_data_train.shape[0]*100
```

In [12]:
```
print("Percent of male",male_percent)
print("Percent of Female",female_percent)
```

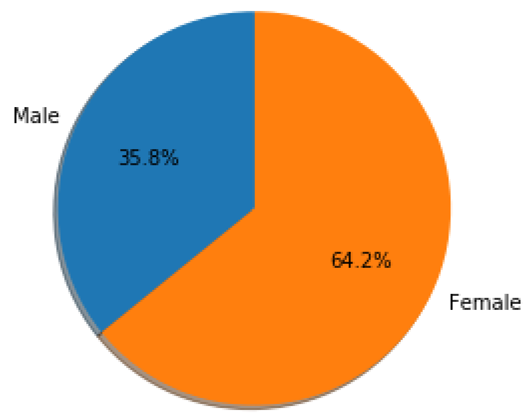
```
Percent of male 35.824234710965236
Percent of Female 64.17576528903477
```

Pie Plot

In [13]:
```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Male', 'Female'
sizes = [male_percent, female_percent]
explode = (0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



DistPlot

In [14]:
```
sns.FacetGrid(gender_age_data_train, hue = 'gender', height = 5)\
    .map(sns.distplot, 'age')\
    .add_legend()
plt.ylabel("Density of people")
plt.title("PDP of Age for Male and Female")
plt.show()
```

Observation

1. There is a overlap in the Number of Male and Female in the age group of 15 to 40
2. The Number of Female is slightly more than Male in the age group from 42 to 65

Percentile

In [21]:
```
male_data = gender_age_data_train[gender_age_data_train['gender']=='M']
female_data = gender_age_data_train[gender_age_data_train['gender']=='F']
```

In [22]:
```
male_data.shape
```

Out[22]: (47904, 4)

In [23]:
```
female_data.shape
```

Out[23]: (26741, 4)

In [24]:
```
female_data.head()
```

Out[24]:

	device_id	gender	age	group
5	-1297074871525174196	F	24	1
9	1596610250680140042	F	36	4
11	7477216237379271436	F	37	4
12	2478205222798310601	F	28	2
15	1508636020748379883	F	28	2

In [25]:
```
male_percentile = np.percentile(male_data['age'], [25,50,75,90])
female_percentile = np.percentile(female_data['age'],[25,50,75,90])
```

In [26]:
```
print(50*'*')
print("Male:")
print(50*'*')
print(male_data['age'].describe())
print("25th,50th,90th,100th percentile is",male_percentile)
print(50*'*')
print("Male:")
print(50*'*')
print(female_data['age'].describe())
print("25th,50th,90th,100th percentile is",female_percentile)
```

```
**************************************************
Male:
**************************************************
count    47904.000000
mean        31.052939
std          9.454653
min          1.000000
25%         25.000000
50%         29.000000
75%         35.000000
max         90.000000
Name: age, dtype: float64
25th,50th,90th,100th percentile is [25. 29. 35. 44.]
**************************************************
Male:
**************************************************
count    26741.000000
mean        32.050596
std         10.539967
min         10.000000
25%         25.000000
50%         29.000000
75%         37.000000
max         96.000000
Name: age, dtype: float64
25th,50th,90th,100th percentile is [25. 29. 37. 47.]
```

Observation:

1. The Number of Male is almost double the number of Female
2. The Mean, Median age is almost same for both Male and Female
3. 25 and 75 Percentile Age values for Both Male and Female are similar
4. In General Female users have more age than Male users in the Data

3.2. Phone Brand and Device Data

In [27]:
```
#Read the csv file
phone_brand_device_model_data = pd.read_csv('Datasets/phone_brand_device_model.c
```

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

Out[28]:

	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 [29]:
```
#Check for unique brands
len(phone_brand_device_model_data.phone_brand.unique())
```

Out[29]: 131

In [30]:
```
phone_brand_device_model_data.shape
```

Out[30]: (187245, 3)

3.2.1. Check and Remove Duplicates

In [31]:
```
#https://thispointer.com/pandas-find-duplicate-rows-in-a-dataframe-based-on-all-c
duplicate_devices = phone_brand_device_model_data[phone_brand_device_model_data.c
```

In [32]:
```
duplicate_devices.shape
```

Out[32]: (529, 3)

In [33]:
```
#Dropping duplicates
phone_brand_device_model_data = phone_brand_device_model_data.drop_duplicates('de
```

In [34]:
```
phone_brand_device_model_data.shape
```

Out[34]: (186716, 3)

In [35]:
```
unique_before = phone_brand_device_model_data.phone_brand.unique()
```

In [36]:
```
gender_age_data_test = pd.read_csv('Datasets/gender_age_test.csv')
```

In [37]:
```
gender_age_data_test.shape
```

Out[37]:
```
(112071, 1)
```

In [38]:
```
gender_age_data_test.head()
```

Out[38]:

	device_id
0	1002079943728939269
1	-1547860181818787117
2	7374582448058474277
3	-6220210354783429585
4	-5893464122623104785

Observation

529 duplicate rows are there in phone device and model data set

3.2.2. Phone Brand Naming Convention

In [39]:
```
chinese_english_brand_mapping = {
    "三星": "samsung",
    "天语": "Ktouch",
    "海信": "hisense",
    "联想": "lenovo",
    "欧比": "obi",
    "爱派尔": "ipair",
    "努比亚": "nubia",
    "优米": "youmi",
    "朵唯": "dowe",
    "黑米": "heymi",
    "锤子": "hammer",
    "酷比魔方": "koobee",
    "美图": "meitu",
    "尼比鲁": "nibilu",
    "一加": "oneplus",
    "优购": "yougo",
    "诺基亚": "nokia",
    "糖葫芦": "candy",
    "中国移动": "ccmc",
    "语信": "yuxin",
    "基伍": "kiwu",
    "青橙": "greeno",
    "华硕": "asus",
    "夏新": "panosonic",
    "维图": "weitu",
    "艾优尼": "aiyouni",
    "摩托罗拉": "moto",
    "乡米": "xiangmi",
    "米奇": "micky",
    "大可乐": "bigcola",
    "沃普丰": "wpf",
    "神舟": "hasse",
    "摩乐": "mole",
    "飞秒": "fs",
    "米歌": "mige",
    "富可视": "fks",
    "德赛": "desci",
    "梦米": "mengmi",
    "乐视": "lshi",
    "小杨树": "smallt",
    "纽曼": "newman",
    "邦华": "banghua",
    "E派": "epai",
    "易派": "epai",
    "普耐尔": "pner",
    "欧新": "ouxin",
    "西米": "ximi",
    "海尔": "haier",
    "波导": "bodao",
    "糯米": "nuomi",
    "唯米": "weimi",
    "酷珀": "kupo",
    "谷歌": "google",
    "昂达": "ada",
    "聆韵": "lingyun",
```

```
    "小米": "Xiaomi",
    "华为": "Huawei",
    "魅族": "Meizu",
    "中兴": "ZTE",
    "酷派": "Coolpad",
    "金立": "Gionee",
    "SUGAR": "SUGAR",
    "OPPO": "OPPO",
    "vivo": "vivo",
    "HTC": "HTC",
    "LG": "LG",
    "ZUK": "ZUK",
    "TCL": "TCL",
    "LOGO": "LOGO",
    "SUGAR": "SUGAR",
    "Lovme": "Lovme",
    "PPTV": "PPTV",
    "ZOYE": "ZOYE",
    "MIL": "MIL",
    "索尼" : "Sony",
    "欧博信" : "Opssom",
    "奇酷" : "Qiku",
    "酷比" : "CUBE",
    "康佳" : "Konka",
    "亿通" : "Yitong",
    "金星数码" : "JXD",
    "至尊宝" : "Monkey King",
    "百立丰" : "Hundred Li Feng",
    "贝尔丰" : "Bifer",
    "百加" : "Bacardi",
    "诺亚信" : "Noain",
    "广信" : "Kingsun",
    "世纪天元" : "Ctyon",
    "青葱" : "Cong",
    "果米" : "Taobao",
    "斐讯" : "Phicomm",
    "长虹" : "Changhong",
    "欧奇" : "Oukimobile",
    "先锋" : "XFPLAY",
    "台电" : "Teclast",
    "大Q" : "Daq",
    "蓝魔" : "Ramos",
    "奥克斯" : "AUX"
}
```

In [40]:
```
english_brands = map(lambda x: chinese_english_brand_mapping[x] if x in chinese_(
```

In [41]:
```
phone_brand_device_model_data['phone_brand_english'] = list(english_brands)
```

In [42]:
```
phone_brand_device_model_data.shape
```

Out[42]:
```
(186716, 4)
```

In [43]:
```
phone_brand_count = dict(phone_brand_device_model_data['phone_brand_english'].va
```

In [44]:
```
phone_brand_count
```

Out[44]:
```
{'Xiaomi': 43107,
 'samsung': 34191,
 'Huawei': 32465,
 'vivo': 14342,
 'OPPO': 14239,
 'Meizu': 11816,
 'Coolpad': 8382,
 'lenovo': 6752,
 'Gionee': 2763,
 'HTC': 2675,
 'ZTE': 2092,
 'lshi': 1916,
 'Sony': 1818,
 'nubia': 1142,
 'LG': 761,
 'ccmc': 668,
 'TCL': 583,
 'dowe': 544,
 'hammer': 534,
```

In [45]:
```
import itertools
top_brand_count = dict(itertools.islice(phone_brand_count.items(), 15))
```

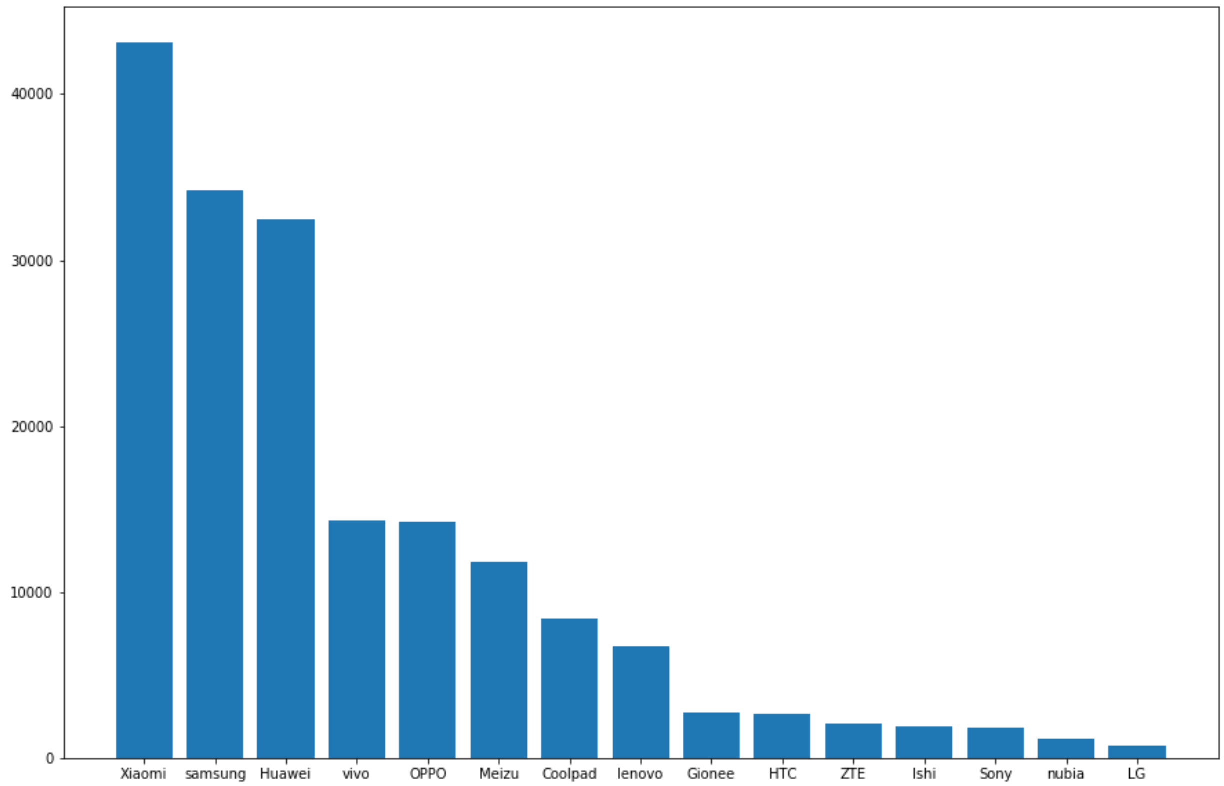
In [46]:
```
#Top 15 Brand Counts
top_brand_count
```

Out[46]:
```
{'Xiaomi': 43107,
 'samsung': 34191,
 'Huawei': 32465,
 'vivo': 14342,
 'OPPO': 14239,
 'Meizu': 11816,
 'Coolpad': 8382,
 'lenovo': 6752,
 'Gionee': 2763,
 'HTC': 2675,
 'ZTE': 2092,
 'lshi': 1916,
 'Sony': 1818,
 'nubia': 1142,
 'LG': 761}
```

3.2.3. Bar plot for Brand Counts

In [47]:
```
plt.figure(figsize=(15,10))
plt.bar(top_brand_count.keys(),top_brand_count.values())
```

Out[47]: <BarContainer object of 15 artists>



Observation

Xiaomi and Samsung hold the highest percentage of devices used by the users

In [48]:
```
#Encoding brands with numeric Label
brand_encoder = LabelEncoder().fit(phone_brand_device_model_data['phone_brand'])
phone_brand_device_model_data['brand'] = brand_encoder.transform(phone_brand_dev
nbrands=len(brand_encoder.classes_)#Will be used for one hot encoding
```

In [49]:
```
#Concatinating Phone Brand and Model and encoding with numeric label
concat_model = phone_brand_device_model_data['phone_brand'].str.cat(phone_brand_
model_encoder=LabelEncoder().fit(concat_model)
phone_brand_device_model_data['model_brand']=model_encoder.transform(concat_mode
nmodels=len(model_encoder.classes_)#Will be used for one hot encoding
```

In [50]:
```
#Encoding models with numeric Label
model_encode=LabelEncoder().fit(phone_brand_device_model_data['device_model'])
phone_brand_device_model_data['model']=model_encode.transform(phone_brand_device_
num_models=len(model_encoder.classes_)#Will be used for one hot encoding
```

In [51]:
```
phone_brand_device_model_data.head()
```

Out[51]:

	device_id	phone_brand	device_model	phone_brand_english	brand	model_brand
0	-8890648629457979026	小米	红米	Xiaomi	51	858
1	1277779817574759137	小米	MI 2	Xiaomi	51	843
2	5137427614288105724	三星	Galaxy S4	samsung	15	371
3	3669464369358936369	SUGAR	时尚手机	SUGAR	9	166
4	-5019277647504317457	三星	Galaxy Note 2	samsung	15	347

In [52]:
```
#Making device_id as index in order to have an easier interpretation for mapping
phone_brand_device_model_data = phone_brand_device_model_data.set_index('device_
phone_brand_device_model_data.head()
```

Out[52]:

	phone_brand	device_model	phone_brand_english	brand	model_brand	mo
device_id						
-8890648629457979026	小米	红米	Xiaomi	51	858	1!
1277779817574759137	小米	MI 2	Xiaomi	51	843	7
5137427614288105724	三星	Galaxy S4	samsung	15	371	!
3669464369358936369	SUGAR	时尚手机	SUGAR	9	166	1!
-5019277647504317457	三星	Galaxy Note 2	samsung	15	347	!

3.3. Events

In [53]:
```
#Read the csv file
events_data = pd.read_csv('Datasets/events.csv')
```

In [54]:
```
events_data.shape
```

Out[54]: (3252950, 5)

In [55]:
```
events_data.head()
```

Out[55]:

	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

In [56]:
```
#Check for unique devices
unique_device_ids = np.unique(events_data['device_id'])
print(len(unique_device_ids))
```

```
60865
```

3.3.1. Devices Having and not having Events- Train Data

In [57]:
```
event_flag = gender_age_data_train['device_id'].apply(lambda x: 1 if x in unique_
```

In [58]:
```
event_flag.value_counts()
```

Out[58]:
```
0    51336
1    23309
Name: device_id, dtype: int64
```

Percentage of Devices not having events

In [59]:
```
no_events = np.round(event_flag.value_counts()[0]/len(gender_age_data_train['dev
print(no_events)
```

```
68.77
```

Percentage of Devices having events Train

In [60]:
```
having_events = np.round(event_flag.value_counts()[1]/len(gender_age_data_train[
print(having_events)
```

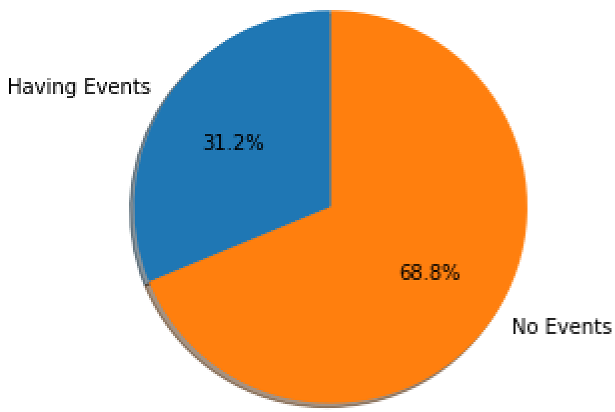
```
31.23
```

Pie Plot for Percentages

In [61]:
```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Having Events', 'No Events'
sizes = [having_events, no_events]
explode = (0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



In [62]:
```
gender_age_data_train['event_flag'] = event_flag
```

In [63]:
```
gender_age_data_train.head()
```

Out[63]:

	device_id	gender	age	group	event_flag
0	-8076087639492063270	M	35	10	0
1	-2897161552818060146	M	35	10	0
2	-8260683887967679142	M	35	10	1
3	-4938849341048082022	M	30	9	0
4	245133531816851882	M	30	9	0

In [64]:
```
having_events_data_train = gender_age_data_train[gender_age_data_train['event_fl
no_events_data_train = gender_age_data_train[gender_age_data_train['event_flag']=
```

In [65]:
```
having_events_data_train = having_events_data_train.drop(['event_flag'],axis=1)
no_events_data_train = no_events_data_train.drop(['event_flag'],axis=1)
```

In [66]:
```
having_events_data_train.shape
```
Out[66]:
```
(23309, 4)
```

In [67]:
```
no_events_data_train.shape
```
Out[67]:
```
(51336, 4)
```

In [68]:
```
#Creating two separate datasets for devices having and not having events
having_events_data_train.to_csv('Datasets/NewData/having_events_data_train.csv')
no_events_data_train.to_csv('Datasets/NewData/no_events_data_train.csv')
```

Conclusion

For train data 23309 devices have events and 51336 devices do not have events

3.3.2. Devices Having and not having Events- Train Data

In [69]:
```
event_flag = gender_age_data_test['device_id'].apply(lambda x: 1 if x in unique_
```

Percentage of Devices having events Test

In [70]:
```
having_events = np.round(event_flag.value_counts()[1]/len(gender_age_data_test['
print(having_events)
```
```
31.4
```

Percentage of Devices having events Test

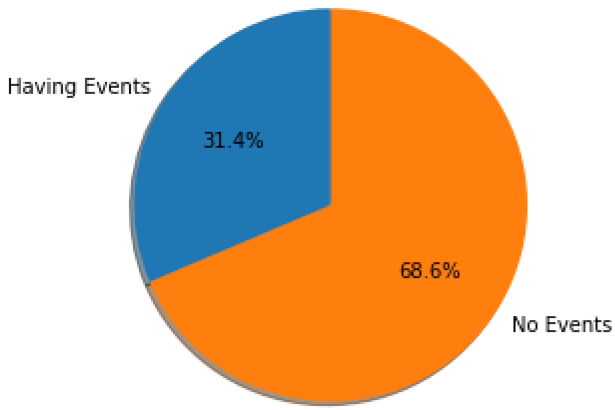
In [71]:
```
no_events = np.round(event_flag.value_counts()[0]/len(gender_age_data_test['devi
print(no_events)
```
```
68.6
```

Pie Plot for Percentages

In [72]:
```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Having Events', 'No Events'
sizes = [having_events, no_events]
explode = (0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



3.3.3. Checking for Latitude and Logitude details

In [73]:
```
events_data['latitude'].describe()
```

Out[73]:
```
count    3.252950e+06
mean     2.162949e+01
std      1.569697e+01
min     -3.843000e+01
25%      0.000000e+00
50%      2.802000e+01
75%      3.407000e+01
max      5.994000e+01
Name: latitude, dtype: float64
```

In [74]:
```
events_data['longitude'].describe()
```

Out[74]:
```
count    3.252950e+06
mean     7.796192e+01
std      5.405801e+01
min     -1.800000e+02
25%      0.000000e+00
50%      1.129500e+02
75%      1.172100e+02
max      1.747600e+02
Name: longitude, dtype: float64
```

Conclusion

68 percent of devices donot have events while 31 percent of devices have.

In [75]:
```
gender_age_data_test['event_flag'] = event_flag
```

In [76]:
```
having_events_data_test = gender_age_data_test[gender_age_data_test['event_flag'
no_events_data_test = gender_age_data_test[gender_age_data_test['event_flag']==0
```

In [77]:
```
having_events_data_test.shape
```

Out[77]: (35194, 2)

In [78]:
```
no_events_data_test.shape
```

Out[78]: (76877, 2)

In [79]:
```
gender_age_data_test.shape
```

Out[79]: (112071, 2)

In [80]:
```
#Creating new train datasets with having and not having events
having_events_data_test = having_events_data_test.drop(['event_flag'],axis=1)
no_events_data_test = no_events_data_test.drop(['event_flag'],axis=1)
```

In [81]:
```
#Creating new test datasets with having and not having events
having_events_data_test.to_csv('Datasets/NewData/having_events_data_test.csv')
no_events_data_test.to_csv('Datasets/NewData/no_events_data_test.csv')
```

3.3.4. Process and merge Latitute and Longitude

In [82]:
```
#Computing median of lattitude and longitude for all devices
lat_medians = events_data.groupby("device_id")["latitude"].apply(lambda x: np.me
long_medians = events_data.groupby("device_id")["longitude"].apply(lambda x: np.
```

In [83]:
```
#Merging the mean latitude and logitude with the respective devices
having_events_data_train = pd.merge(left=having_events_data_train, right=lat_medi
having_events_data_train = pd.merge(left=having_events_data_train, right=long_me
having_events_data_test = pd.merge(left=having_events_data_test, right=lat_median
having_events_data_test = pd.merge(left=having_events_data_test, right=long_medi
```

In [84]:
```
#Computing number of evnts a device has
number_of_events = dict(events_data["device_id"].value_counts())
having_events_data_train['NumberOfEvents'] = having_events_data_train['device_id
having_events_data_test['NumberOfEvents'] = having_events_data_test['device_id']
```

In [85]:
```
having_events_data_train.head()
```

Out[85]:

	device_id	gender	age	group	latitude	longitude	NumberOfEvents
0	-8260683887967679142	M	35	10	0.00	0.00	1
1	7477216237379271436	F	37	4	31.75	119.57	7
2	6352067998666467520	M	32	10	0.00	0.00	11
3	1508636020748379883	F	28	2	31.90	120.26	35
4	-6876541075223249434	M	75	11	39.14	117.20	28

In [86]:
```
events_data[events_data['device_id']==-8260683887967679142]
```

Out[86]:

	event_id	device_id	timestamp	longitude	latitude
2479655	2479656	-8260683887967679142	2016-05-01 14:23:37	0.0	0.0

3.3.5.Merging Phone Brand and model to train and test data

In [87]:
```
# Setting device_id as the index of the below tables in order to map with the nex
having_events_data_train = having_events_data_train.set_index('device_id')
having_events_data_test = having_events_data_test.set_index('device_id')
no_events_data_train = no_events_data_train.set_index('device_id')
no_events_data_test = no_events_data_test.set_index('device_id')
gender_age_data_train = gender_age_data_train.set_index('device_id')
gender_age_data_test = gender_age_data_test.set_index('device_id')
```

In [364]:
```
# Phone brnd and model are mapped to the event and no event dataset by index(dev
having_events_data_train['phone_brand'] = having_events_data_train.index.map(phon
having_events_data_train['device_model'] = having_events_data_train.index.map(ph
having_events_data_test['phone_brand'] = having_events_data_test.index.map(phone_
having_events_data_test['device_model'] = having_events_data_test.index.map(phon
no_events_data_train['phone_brand'] = no_events_data_train.index.map(phone_brand_
no_events_data_train['device_model'] = no_events_data_train.index.map(phone_bran
no_events_data_test['phone_brand'] = no_events_data_test.index.map(phone_brand_d
no_events_data_test['device_model'] = no_events_data_test.index.map(phone_brand_
gender_age_data_train['phone_brand'] = gender_age_data_train.index.map(phone_bra
gender_age_data_train['device_model'] = gender_age_data_train.index.map(phone_br
gender_age_data_test['phone_brand'] = gender_age_data_test.index.map(phone_brand_
gender_age_data_test['device_model'] = gender_age_data_test.index.map(phone_bran
```

3.3.6. Computing Time of the events performed

In [368]:
```
# Excat time of the event is stored by converting into 24 hour format
events_data['hours'] = events_data['timestamp'].map(lambda x:pd.to_datetime(x).h
hour_events = events_data.groupby("device_id")["hours"].apply(lambda x: " ".join
hour_events.head()
```

Out[368]:
```
device_id
-9222956879900151005    011 012 015 012 015 021 015 015 021 07 012 015...
-9222661944218806987                       021 019 022 018 018 018 00 018
-9222399302879214035              011 013 023 021 013 023 010 013 023 013
-9221825537663503111    07 07 07 08 013 07 06 07 07 08 013 08 013 010 ...
-9221767098072603291                         05 015 014 012 018 05 013 07
Name: hours, dtype: object
```

3.3.7 Computing Interval of Events

In [369]:
```
# Event time is divided into 4 different intervals
# 1. 04 to 10 - Morning
# 2. 10 to 16 - AfterNoon
# 3. 16 to 20 - Evening
# 4. 20 to 24 - Night
events_data['intervals'] = ["Morning" if ((x>=4)&(x<=10)) else "AfterNoon" if ((
intervals_events = events_data.groupby("device_id")["intervals"].apply(lambda x:
intervals_events.head()
```

Out[369]:
```
device_id
-9222956879900151005    AfterNoon AfterNoon AfterNoon AfterNoon AfterN...
-9222661944218806987    Night Evening Night Evening Evening Evening Ni...
-9222399302879214035    AfterNoon AfterNoon Night Night AfterNoon Nigh...
-9221825537663503111    Morning Morning Morning Morning AfterNoon Morn...
-9221767098072603291    Morning AfterNoon AfterNoon AfterNoon Evening ...
Name: intervals, dtype: object
```

3.3.8. Computing Day of the Event

In [371]:
```
# Day of the week on which event is carried out
events_data['day'] = events_data['timestamp'].map(lambda x:pd.to_datetime(x).day_
day_events = events_data.groupby("device_id")["day"].apply(lambda x: " ".join(s 
day_events.head()
```

Out[371]:
```
device_id
-9222956879900151005    Saturday Saturday Saturday Saturday Friday Fri...
-9222661944218806987    Wednesday Thursday Monday Sunday Saturday Frid...
-9222399302879214035    Wednesday Monday Tuesday Friday Wednesday Wedn...
-9221825537663503111    Saturday Saturday Friday Sunday Thursday Frida...
-9221767098072603291    Wednesday Sunday Sunday Wednesday Thursday Tue...
Name: day, dtype: object
```

3.3.9. Merging All the Computed features into train and test data

In [372]:
```
# Mapping hours, intervals and days are mapped to events table by index
having_events_data_train['hours'] = having_events_data_train.index.map(hour_even
having_events_data_test['hours'] = having_events_data_test.index.map(hour_events
having_events_data_train['intervals'] = having_events_data_train.index.map(inter
having_events_data_test['intervals'] = having_events_data_test.index.map(interva
having_events_data_train['days'] = having_events_data_train.index.map(day_events
having_events_data_test['days'] = having_events_data_test.index.map(day_events)
```

3.4. App Events

In [375]:
```
#Read the csv file
app_events_data = pd.read_csv('Datasets/app_events.csv')
```

In [376]:
```
app_events_data.shape
```

Out[376]:
```
(32473067, 4)
```

In [377]:
```
app_events_data.head()
```

Out[377]:

	event_id	app_id	is_installed	is_active
0	2	5927333115845830913	1	1
1	2	-5720078949152207372	1	0
2	2	-1633887856876571208	1	0
3	2	-653184325010919369	1	1
4	2	8693964245073640147	1	1

3.4.1 Computing if App is active and installed

In [378]:
```
# Checking if app is active
app_events_data['is_installed'].value_counts()
```

Out[378]:
```
1    32473067
Name: is_installed, dtype: int64
```

In [379]:
```
# Checking if app is installed
app_events_data['is_active'].value_counts()
```

Out[379]:
```
0    19740071
1    12732996
Name: is_active, dtype: int64
```

In [380]:
```
app_events_data['is_active'] = app_events_data['is_active'].apply(lambda x: True
```

In [381]:
```
app_events_data.head()
```

Out[381]:

	event_id	app_id	is_installed	is_active
0	2	5927333115845830913	1	True
1	2	-5720078949152207372	1	False
2	2	-1633887856876571208	1	False
3	2	-653184325010919369	1	True
4	2	8693964245073640147	1	True

3.4.2. Creating Ttrainrow and testrow

In [382]:
```
# trainrow and testrow are created in order to keep a record there in the respect
# train and test row used by some features for mapping
having_events_data_train['trainrow'] = np.arange(having_events_data_train.shape[
having_events_data_test['testrow'] = np.arange(having_events_data_test.shape[0])
no_events_data_train['trainrow'] = np.arange(no_events_data_train.shape[0])
no_events_data_test['testrow'] = np.arange(no_events_data_test.shape[0])
gender_age_data_train['trainrow'] = np.arange(gender_age_data_train.shape[0])
gender_age_data_test['testrow'] = np.arange(gender_age_data_test.shape[0])
```

In [383]:
```
events_data = events_data.set_index('event_id')
```

In [384]:
```
events_data[events_data['device_id']==29182687948017175].head()
```

Out[384]:

event_id	device_id	timestamp	longitude	latitude	hours	intervals	day
1	29182687948017175	2016-05-01 00:55:25	121.38	31.24	0	Night	Sunday
7104	29182687948017175	2016-05-02 09:37:02	121.38	31.24	9	Morning	Monday
29661	29182687948017175	2016-05-04 00:56:04	121.39	31.23	0	Night	Wednesday
33133	29182687948017175	2016-05-06 05:01:15	121.38	31.24	5	Morning	Friday
38980	29182687948017175	2016-05-06 09:55:04	121.16	31.00	9	Morning	Friday

3.4.3. Computing Bag of Apps

For Each device we want to know which all apps were installed in the device. So we will first encode All the App Ids as integers from 0 to number of unique apps - 1. To get the Apps which are installed in a device denoted by device_id, we merge device_id column from events table to app_events group the resulting dataframe by device_id and app and aggregate. we then Merge in the trainrow, testrow columns to know at which row to put each device in the features matrix

In [386]:
```
#https://www.kaggle.com/dvasyukova/a-linear-model-on-apps-and-labels

# Number of app_id a particular device has
# For a device first the event is computed and for the event respective app ids c
app_encoder = LabelEncoder().fit(app_events_data['app_id'])
app_events_data['app'] = app_encoder.transform(app_events_data['app_id'])
napps = len(app_encoder.classes_)# number of unique apps it will be used in creat
deviceapps = (app_events_data.merge(events_data[['device_id']], how='left',left_c
                       .groupby(['device_id','app'])['app'].agg(['count'])
                       .merge(having_events_data_train[['trainrow']], how='left'
                       .merge(having_events_data_test[['testrow']], how='left', l
                       .reset_index())
deviceapps.head()
```

Out[386]:

	device_id	app	count	trainrow	testrow
0	-9222956879900151005	548	18	5145.0	NaN
1	-9222956879900151005	1096	18	5145.0	NaN
2	-9222956879900151005	1248	26	5145.0	NaN
3	-9222956879900151005	1545	12	5145.0	NaN
4	-9222956879900151005	1664	18	5145.0	NaN

3.4,4 Computing and mergeing app active status to train and test data

In [387]:
```
appisactive = app_events_data.groupby("event_id")["is_active"].apply(lambda x: "
appisactive.head()
```

Out[387]:
```
event_id
2     True False False True True True False False Fa...
6     True True True True True False True False True...
7     False True False False True True False False F...
9     False False False False False False False True...
16    False False False False False False False True...
Name: is_active, dtype: object
```

In [388]:
```
#Mapping apps is_active to device_id
events_data["apps_active"] = events_data.index.map(appisactive)
events_apps_active_map = events_data.groupby("device_id")["apps_active"].apply(la
events_apps_active_map.head()
```

Out[388]:
```
device_id
-9222956879900151005    False False False False False False False True...
-9222661944218806987    True False True True True True True False Fals...
-9222399302879214035    False False False False False False False Fals...
-9221825537663503111    False False True False False True True False F...
-9221767098072603291    True False False False False True False True F...
Name: apps_active, dtype: object
```

In [389]:
```
having_events_data_train['app_is_active'] = having_events_data_train.index.map(ev
having_events_data_test['app_is_active'] = having_events_data_test.index.map(even
```

3.5. App Labels

In [392]:
```
#Read the CSV file
app_labels_data = pd.read_csv("Datasets/app_labels.csv")
```

In [393]:
```
app_labels_data.head()
```

Out[393]:

	app_id	label_id
0	7324884708820027918	251
1	-4494216993218550286	251
2	6058196446775239644	406
3	6058196446775239644	407
4	8694625920731541625	406

In [394]:
```
app_labels_data[app_labels_data['app_id']==7324884708820027918]
```

Out[394]:

	app_id	label_id
0	7324884708820027918	251
403	7324884708820027918	691
135925	7324884708820027918	751
142457	7324884708820027918	786
145794	7324884708820027918	775
149151	7324884708820027918	781
152495	7324884708820027918	405
155783	7324884708820027918	730
438422	7324884708820027918	1015

Computing Bag of Labels

App Labels are also created in a similar approach by merging with deviceapps dataframe and grouping by labels and then merging it with trainrow, testrow to know at which row to put each device in the feature matrix

In [395]:
```
#https://www.kaggle.com/dvasyukova/a-linear-model-on-apps-and-labels

# Unique app_id are computed and are encode with numeric label
app_labels_data = app_labels_data.loc[app_labels_data['app_id'].isin(app_events_
app_labels_data['app'] = app_encoder.transform(app_labels_data['app_id'])
labelencoder = LabelEncoder().fit(app_labels_data['label_id'])
app_labels_data['label'] = labelencoder.transform(app_labels_data['label_id'])
nlabels = len(labelencoder.classes_)
```

In [396]:
```
# Number of app labels a particular device has
# For a device first the event is computed and for the event respective app ids 
#number of lables are computed
devicelabels = (deviceapps[['device_id','app']]
                .merge(app_labels_data[['app','label']])
                .groupby(['device_id','label'])['app'].agg(['size'])
                .merge(having_events_data_train[['trainrow']], how='left', left_
                .merge(having_events_data_test[['testrow']], how='left', left_in
                .reset_index())
devicelabels.head()
```

Out[396]:

	device_id	label	size	trainrow	testrow
0	-9222956879900151005	117	1	5145.0	NaN
1	-9222956879900151005	120	1	5145.0	NaN
2	-9222956879900151005	126	1	5145.0	NaN
3	-9222956879900151005	138	2	5145.0	NaN
4	-9222956879900151005	147	2	5145.0	NaN

In [397]:
```
devicelabels[devicelabels['label']==1]
```

Out[397]:

	device_id	label	size	trainrow	testrow
3295403	5083019926611946481	1	1	NaN	30043.0
3325396	5197530142143050644	1	1	13761.0	NaN

3.6. Label Categories

From this data set we only get the name of the labels

In [501]:
```
#Read the CSV file
label_categories_data = pd.read_csv("Datasets/label_categories.csv")
```

In [399]:
```
label_categories_data.head()
```

Out[399]:

	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

In [400]:
```
label_categories_data.shape
```

Out[400]:
```
(930, 2)
```

4. Featurization

Save and Load Sparse matrix

In [89]:
```
#https://stackoverflow.com/questions/8955448/save-load-scipy-sparse-csr-matrix-ir
#Saves a file with .npz extension
def save_sparse_matrix(filename, xmtr):
    np.savez(filename,data = xmtr.data ,indices= xmtr.indices,
             indptr =xmtr.indptr, shape=xmtr.shape )

#Loads a sparse matrix
def load_sparse_matrix(filename):
    tmp = np.load(filename)
    return csr_matrix((tmp['data'], tmp['indices'], tmp['indptr']), shape= tmp['s
```

4.1. All Data

In [90]:
```
#Loading the one hot encoding matrices for All Devices
x_ohe_train = load_sparse_matrix('sparse/x_ohe_train.npz')
x_ohe_test = load_sparse_matrix('sparse/x_ohe_test.npz')
```

4.1.1. One Hot Encoding - Phone brand

In [125]:
```
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.htm
X_device_brand_ohe_train = csr_matrix((np.ones(gender_age_data_train.shape[0]), #
                          (gender_age_data_train.trainrow,
                           gender_age_data_train.phone_brand)))
X_device_brand_ohe_test = csr_matrix((np.ones(gender_age_data_test.shape[0]), # N
                          (gender_age_data_test.testrow, gender_age_data_test.phone_
print("Train Brand One-hot Shape: ",X_device_brand_ohe_train.shape)
print("Test Brand One-hot Shape: ",X_device_brand_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (74645, 131)
Test Brand One-hot Shape:  (112071, 131)
```

4.1.2. One Hot Encoding - Device Model - Regular

In [126]:
```
X_device_model_ohe_train = csr_matrix((np.ones(gender_age_data_train.shape[0]), #
                          (gender_age_data_train.trainrow,
                           gender_age_data_train.device_model)))
X_device_model_ohe_test = csr_matrix((np.ones(gender_age_data_test.shape[0]), # N
                          (gender_age_data_test.testrow, gender_age_data_test.devic
print("Train Brand One-hot Shape: ",X_device_model_ohe_train.shape)
print("Test Brand One-hot Shape: ",X_device_model_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (74645, 1599)
Test Brand One-hot Shape:  (112071, 1599)
```

In [144]:
```
#Merge all the features for ALL Devices
x_ohe_train = hstack((X_device_brand_ohe_train,X_device_model_ohe_train))
x_ohe_test = hstack((X_device_brand_ohe_test,X_device_model_ohe_test))
```

In [145]:
```
#Print the shapes
print("Train Brand One-hot Shape: ",x_ohe_train.shape)
print("Test Brand One-hot Shape: ",x_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (74645, 1730)
Test Brand One-hot Shape:  (112071, 1730)
```

In [6]:
```
# #Saving One-hot encoded Matrices
# save_sparse('sparse/x_ohe_train.npz',x_ohe_train.tocsr())
# save_sparse('sparse/x_ohe_test.npz',x_ohe_test.tocsr())
```

4.2. No Events

In [92]:
```
#Loading the one hot encoding matrices for Devices with no events
X_no_events_ohe_train = load_sparse_matrix('sparse/X_no_events_ohe_train.npz')
X_no_events_ohe_test = load_sparse_matrix('sparse/X_no_events_ohe_test.npz')
```

4.2.1. One Hot Encoding Device Brand

In [147]:
```
X_no_events_device_brand_ohe_train = csr_matrix((np.ones(no_events_data_train.sh
                       (no_events_data_train.trainrow, no_events_data_train.phon
X_no_events_device_brand_ohe_test = csr_matrix((np.ones(no_events_data_test.shap
                       (no_events_data_test.testrow, no_events_data_test.phone_b
print("Train Brand One-hot Shape: ",X_no_events_device_brand_ohe_train.shape)
print("Test Brand One-hot Shape: ",X_no_events_device_brand_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (51336, 131)
Test Brand One-hot Shape:  (76877, 131)
```

4.2.2. One Hot Encoding - Device Model - Regular

In [149]:
```
X_no_events_device_model_ohe_train = csr_matrix((np.ones(no_events_data_train.sh
                       (no_events_data_train.trainrow, no_events_data_train.devi
X_no_events_device_model_ohe_test = csr_matrix((np.ones(no_events_data_test.shap
                       (no_events_data_test.testrow, no_events_data_test.device_
print("Train Brand One-hot Shape: ",X_no_events_device_model_ohe_train.shape)
print("Test Brand One-hot Shape: ",X_no_events_device_model_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (51336, 1599)
Test Brand One-hot Shape:  (76877, 1599)
```

In [150]:
```
#Merging all the features for Device with no events
X_no_events_ohe_train = hstack((X_no_events_device_brand_ohe_train,X_no_events_d
X_no_events_ohe_test = hstack((X_no_events_device_brand_ohe_test,X_no_events_dev
```

In [151]:
```
print("Train Brand One-hot Shape: ",X_no_events_ohe_train.shape)
print("Test Brand One-hot Shape: ",X_no_events_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (51336, 1730)
Test Brand One-hot Shape:  (76877, 1730)
```

In [7]:
```
#Saving One-hot encoded Matrices for devices with no events
save_sparse('sparse/X_no_events_ohe_train.npz',X_no_events_ohe_train.tocsr())
save_sparse('sparse/X_no_events_ohe_test.npz',X_no_events_ohe_test.tocsr())
```

4.3. Having Events

In [94]:
```
#Loading the one hot encoding matrices for Devices with events
X_having_events_train = load_sparse_matrix('sparse/X_having_events_train.npz')
X_having_events_test = load_sparse_matrix('sparse/X_having_events_test.npz')
```

4.3.1. One Hot Encoding - Device Brand

In [152]:
```
X_having_events_device_brand_ohe_train = csr_matrix((np.ones(having_events_data_
                       (having_events_data_train.trainrow, having_events_data_tra
                        shape=(having_events_data_train.shape[0],nbrands))
X_having_events_device_brand_ohe_test = csr_matrix((np.ones(having_events_data_te
                       (having_events_data_test.testrow, having_events_data_test
                        shape=(having_events_data_test.shape[0],nbrands))
print("Train Brand One-hot Shape: ",X_having_events_device_brand_ohe_train.shape
print("Test Brand One-hot Shape: ",X_having_events_device_brand_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (23309, 131)
Test Brand One-hot Shape:  (35194, 131)
```

4.3.2. One Hot Encoding- Device Model - Concatenation of Brand and Model

In [153]:
```
X_having_events_device_model_ohe_train = csr_matrix((np.ones(having_events_data_
                       (having_events_data_train.trainrow, having_events_data_tra
                        shape=(having_events_data_train.shape[0],nmodels))
X_having_events_device_model_ohe_test = csr_matrix((np.ones(having_events_data_te
                       (having_events_data_test.testrow, having_events_data_test
                        shape=(having_events_data_test.shape[0],nmodels))
print("Train Brand One-hot Shape: ",X_having_events_device_model_ohe_train.shape
print("Test Brand One-hot Shape: ",X_having_events_device_model_ohe_test.shape)
```

```
Train Brand One-hot Shape:  (23309, 1667)
Test Brand One-hot Shape:  (35194, 1667)
```

4.3.3. Event App - Bag of Apps

In [154]:
```
#Since the Deviceapps has both train and test columns merged to create Train Apps
#Once we remove Nan in Train Rows we will get the Apps in Train Data and we creat
d = deviceapps.dropna(subset=['trainrow'])
X_events_app_train = csr_matrix((np.ones(d.shape[0]), (d.trainrow, d.app)),
                      shape=(having_events_data_train.shape[0],napps))
#Since the Deviceapps has both train and test columns merged to create Test Apps
#Once we remove Nan in Test Rows we will get the Apps in Test Data and we create
d = deviceapps.dropna(subset=['testrow'])
X_events_app_test = csr_matrix((np.ones(d.shape[0]), (d.testrow, d.app)),
                      shape=(having_events_data_test.shape[0],napps))
print("Train Event Apps One-hot Shape: ",X_events_app_train.shape)
print("Test Event Apps One-hot Shape: ",X_events_app_test.shape)
```

```
Train Event Apps One-hot Shape:  (23309, 19237)
Test Event Apps One-hot Shape:  (35194, 19237)
```

4.3.4. Event Labels - Bag of Labels

In [155]:
```
#Since the DeviceLabels has both train and test columns merged to create Train La
#Once we remove Nan in Train Rows we will get the Labels in Train Data and we cre
d = devicelabels.dropna(subset=['trainrow'])
X_events_label_train = csr_matrix((np.ones(d.shape[0]), (d.trainrow, d.label)),
                      shape=(having_events_data_train.shape[0],nlabels))
#Since the DeviceLabels has both train and test columns merged to create Test Lal
#Once we remove Nan in Test Rows we will get the Labels in Test Data and we creat
d = devicelabels.dropna(subset=['testrow'])
X_events_label_test = csr_matrix((np.ones(d.shape[0]), (d.testrow, d.label)),
                      shape=(having_events_data_test.shape[0],nlabels))
print("Train Event Labels One-hot Shape: ",X_events_label_train.shape)
print("Test Event Labels One-hot Shape: ",X_events_label_test.shape)
```

```
Train Event Labels One-hot Shape:  (23309, 492)
Test Event Labels One-hot Shape:  (35194, 492)
```

4.3.5. TFIDF - Event Hours

In [156]:
```
vectorizer=TfidfVectorizer()
vectorizer.fit(having_events_data_train['hours'].values)

X_event_hours_tfidf_train = vectorizer.transform(having_events_data_train['hours
X_event_hours_tfidf_test = vectorizer.transform(having_events_data_test['hours']
print("After vectorizations")
print("Train Event Hours One-hot Shape: ",X_event_hours_tfidf_train.shape)
print("Test Event Hours One-hot Shape: ",X_event_hours_tfidf_test.shape)
```

```
After vectorizations
Train Event Hours One-hot Shape:  (23309, 24)
Test Event Hours One-hot Shape:  (35194, 24)
```

4.3.6. TFIDF - Intervals

In [157]:
```
vectorizer=TfidfVectorizer()
vectorizer.fit(having_events_data_train['intervals'].values)

X_event_intervals_tfidf_train = vectorizer.transform(having_events_data_train['i
X_event_intervals_tfidf_test = vectorizer.transform(having_events_data_test['int
print("After vectorizations")
print("Train Event Hours One-hot Shape: ",X_event_intervals_tfidf_train.shape)
print("Test Event Hours One-hot Shape: ",X_event_intervals_tfidf_test.shape)
```

```
After vectorizations
Train Event Hours One-hot Shape:  (23309, 4)
Test Event Hours One-hot Shape:  (35194, 4)
```

4.3.7. TFIDF - Days

In [158]:
```
vectorizer=TfidfVectorizer()
vectorizer.fit(having_events_data_train['days'].values)

X_event_days_tfidf_train = vectorizer.transform(having_events_data_train['days']
X_event_days_tfidf_test = vectorizer.transform(having_events_data_test['days'].va
print("After vectorizations")
print("Train Event Day One-hot Shape: ",X_event_days_tfidf_train.shape)
print("Test Event Day One-hot Shape: ",X_event_days_tfidf_test.shape)
```

```
After vectorizations
Train Event Day One-hot Shape:  (23309, 7)
Test Event Day One-hot Shape:  (35194, 7)
```

4.3.8. TFIDF - App is Active

In [159]:
```
vectorizer=TfidfVectorizer()
vectorizer.fit(having_events_data_train['app_is_active'].values)

X_app_is_active_tfidf_train = vectorizer.transform(having_events_data_train['app_
X_app_is_active_tfidf_test = vectorizer.transform(having_events_data_test['app_i
print("After vectorizations")
print("Train Apps Active One-hot Shape: ",X_app_is_active_tfidf_train.shape)
print("Test Apps Active One-hot Shape: ",X_app_is_active_tfidf_test.shape)
```

```
After vectorizations
Train Apps Active One-hot Shape:  (23309, 2)
Test Apps Active One-hot Shape:  (35194, 2)
```

4.3.9. Standardiztion - Latitude

In [160]:
```
scaler=StandardScaler()
scaler.fit(having_events_data_train['latitude'].values.reshape(-1,1))

X_lat_standard_train = scaler.transform(having_events_data_train['latitude'].valu
X_lat_standard_test = scaler.transform(having_events_data_test['latitude'].values
print("After Standardizing")
print("Train Event Latitude Standardized Shape: ",X_lat_standard_train.shape)
print("Test Event Latitude Standardized  Shape: ",X_lat_standard_test.shape)
```

```
After Standardizing
Train Event Latitude Standardized Shape:  (23309, 1)
Test Event Latitude Standardized  Shape:  (35194, 1)
```

4.3.10. Standardization - Longitude

In [161]:
```
scaler=StandardScaler()
scaler.fit(having_events_data_train['longitude'].values.reshape(-1,1))

X_long_standard_train = scaler.transform(having_events_data_train['longitude'].va
X_long_standard_test = scaler.transform(having_events_data_test['longitude'].valu
print("After Standardizing")
print("Train Event Latitude Standardized Shape: ",X_long_standard_train.shape)
print("Test Event Latitude Standardized  Shape: ",X_long_standard_test.shape)
```

```
After Standardizing
Train Event Latitude Standardized Shape:  (23309, 1)
Test Event Latitude Standardized  Shape:  (35194, 1)
```

4.3.11. Standardization - Number of Events

In [502]:
```
# scaler=StandardScaler()
# scaler.fit(having_events_data_train['NumberOfEvents'].values.reshape(-1,1))

# X_events_num_standard_train = scaler.transform(having_events_data_train['Number
# X_events_num_standard_test = scaler.transform(having_events_data_test['NumberO
# print("After Standardizing")
# print("Train Event Latitude Standardized Shape: ",X_events_num_standard_train.s
# print("Test Event Latitude Standardized  Shape: ",X_events_num_standard_test.sh
```

4.3.12. Final Features

In [162]:
```
#Merging all the data sets as per train and test data for devices with events
X_having_events_device_brand_ohe_train = X_having_events_device_brand_ohe_train.
X_having_events_device_model_ohe_train = X_having_events_device_model_ohe_train.
X_events_app_train = X_events_app_train.tocsr()
X_events_label_train = X_events_label_train.tocsr()

X_having_events_device_brand_ohe_test = X_having_events_device_brand_ohe_test.to
X_having_events_device_model_ohe_test = X_having_events_device_model_ohe_test.to
X_events_app_test = X_events_app_test.tocsr()
X_events_label_test = X_events_label_test.tocsr()


X_having_events_train = hstack((X_having_events_device_brand_ohe_train,X_having_
X_having_events_test = hstack((X_having_events_device_brand_ohe_test,X_having_ev
```

In [8]:
```
# #Saving sparse Matrices for devices with events
# save_sparse_matrix('sparse/X_having_events_train.npz',X_having_events_train.to
# save_sparse_matrix('sparse/X_having_events_test.npz',X_having_events_test.tocs
```

5. Models

Confusion Matrix

In [95]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))

    labels = ['F23-', 'F24-26', 'F27-28' ,'F29-32' ,'F33-42','F43+' ,'M22-' ,'M2
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [176]:
```
#Reading train data for actual class labels
train_data = pd.read_csv("Datasets/gender_age_train.csv")
targetencoder = LabelEncoder()
targetencoder.fit(train_data["group"])
targetencoder.classes_
```

Out[176]:
```
array(['F23-', 'F24-26', 'F27-28', 'F29-32', 'F33-42', 'F43+', 'M22-',
       'M23-26', 'M27-28', 'M29-31', 'M32-38', 'M39+'], dtype=object)
```

5.1. No Events Device

In [97]:
```
#Fetching Class Labels
Y_all = gender_age_data_train['group'].values
```

In [98]:
```
#Performing Train, test and cv split on the data
X_all_train , X_all_cv , Y_all_train , Y_all_cv = train_test_split(x_ohe_train, 
X_no_events_test = X_no_events_ohe_test
```

5.1.1. K Nearest Neighbour

In [204]:
```
from sklearn.neighbors import KNeighborsClassifier
k_size = [5,10,50,100,150]
for k_neighbour in k_size:
    knn_no_events_clf = KNeighborsClassifier(n_neighbors=k_neighbour, n_jobs=-1)
    knn_no_events_clf.fit(X_all_train, Y_all_train)
    #Using Model Calibration
    knn_no_events_sig_clf = CalibratedClassifierCV(knn_no_events_clf, method="si
    knn_no_events_sig_clf.fit(X_all_train, Y_all_train)
    knn_no_events_predict_y = knn_no_events_sig_clf.predict_proba(X_all_cv)
    print('For values of K = ', k_neighbour, "The log loss is:",log_loss(Y_all_c
```

```
For values of K =  5 The log loss is: 2.4199942074440424
For values of K =  10 The log loss is: 2.416517351519975
For values of K =  50 The log loss is: 2.405082175808167
For values of K =  100 The log loss is: 2.400830923414351
For values of K =  150 The log loss is: 2.3996708934507853
```

In [205]:
```
#Training with best hyper parameters
knn_no_events_clf = KNeighborsClassifier(n_neighbors=50, n_jobs=-1)
knn_no_events_clf.fit(X_all_train, Y_all_train)
#Using Model Calibration
knn_no_events_sig_clf = CalibratedClassifierCV(knn_no_events_clf, method="sigmoi
knn_no_events_sig_clf.fit(X_all_train, Y_all_train)
knn_no_events_predict_y = knn_no_events_sig_clf.predict_proba(X_all_cv)
print('For values of K = ', k_neighbour, "The log loss is:",log_loss(Y_all_cv, kn
```

```
For values of K =  150 The log loss is: 2.405082175808167
```

In [206]:
```
#Printing Log loss
knn_no_events_pred_train = knn_no_events_sig_clf.predict_proba(X_all_train)
print("Train Log-Loss is",log_loss(Y_all_train, knn_no_events_pred_train, labels
knn_no_events_pred_cv = knn_no_events_sig_clf.predict_proba(X_all_cv)
print("CV Log-Loss is",log_loss(Y_all_cv, knn_no_events_pred_cv, labels=knn_no_e
```

```
Train Log-Loss is 2.3891547112027545
CV Log-Loss is 2.405082175808167
```

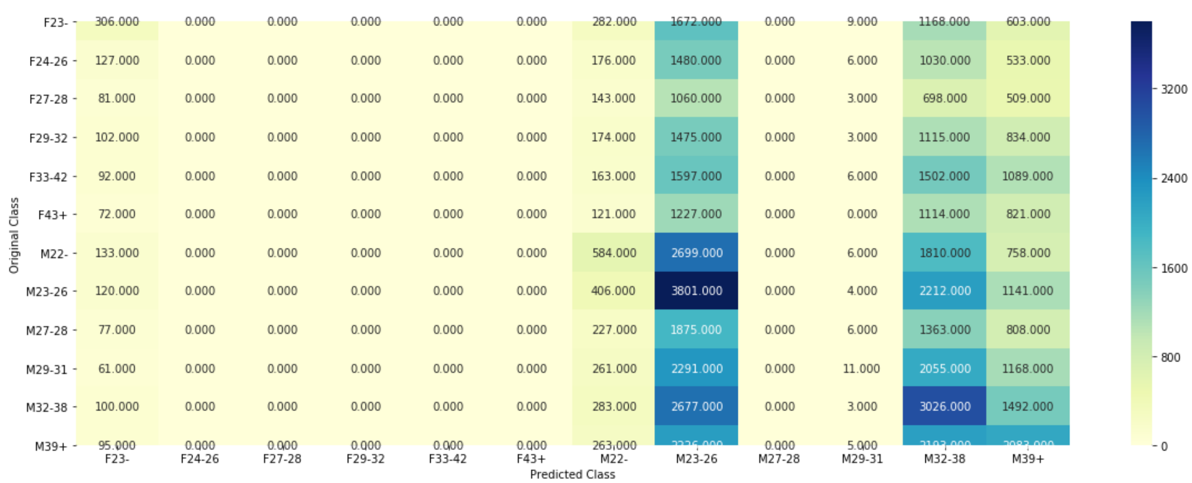
In [207]:
```
y_pred_no_events_train_classes_knn = np.argmax(knn_no_events_pred_train, axis=1)
y_pred_no_events_cv_classes_knn = np.argmax(knn_no_events_pred_cv, axis=1)
```

In [208]:
```
#Printing confusion matrix
print("Train Log Loss :",log_loss(Y_all_train, knn_no_events_pred_train))
plot_confusion_matrix(Y_all_train,y_pred_no_events_train_classes_knn)

print("="*60)

print("CV Log Loss :",log_loss(Y_all_cv, knn_no_events_pred_cv))
plot_confusion_matrix(Y_all_cv,y_pred_no_events_cv_classes_knn)
```

```
Train Log Loss : 2.3891547112027545
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [209]:
```
y_no_events_pred_k_nearest_neighbours = knn_no_events_sig_clf.predict_proba(X_no_
```

In [282]:
```
# Predicting multiclass labels for test data
prediction_no_events_knn = pd.DataFrame(y_no_events_pred_k_nearest_neighbours,in
prediction_no_events_knn.head()
prediction_no_events_knn.to_csv("Predictions/prediction_no_events_knn.csv")
```

5.1.2. SVM

In [216]:
```
from sklearn.linear_model import SGDClassifier
hyperparameters = [10**-4, 10**-3,10**-2,10**-1,10**0, 10**1, 10**2,10**3,10**4]
for c in hyperparameters:
    svm_no_events_clf = SGDClassifier(loss='hinge',alpha=c, class_weight='balanc
    svm_no_events_clf.fit(X_all_train, Y_all_train)
    #Using Model Calibration
    svm_no_events_sig_clf = CalibratedClassifierCV(svm_no_events_clf, method="si
    svm_no_events_sig_clf.fit(X_all_train, Y_all_train)
    svm_no_events_predict_y = svm_no_events_sig_clf.predict_proba(X_all_cv)
    print('For values of alpha = ', c, "The log loss is:",log_loss(Y_all_cv, svm_
```

```
For values of alpha =  0.0001 The log loss is: 2.4216545233211333
For values of alpha =  0.001 The log loss is: 2.4185207695217543
For values of alpha =  0.01 The log loss is: 2.4278811571743164
For values of alpha =  0.1 The log loss is: 2.4267097373587743
For values of alpha =  1 The log loss is: 2.4238633856291263
For values of alpha =  10 The log loss is: 2.423863380592673
For values of alpha =  100 The log loss is: 2.4238633808488466
For values of alpha =  1000 The log loss is: 2.4238633808468855
For values of alpha =  10000 The log loss is: 2.4238633809211887
```

In [217]:
```
#Training with best hyperparameters
svm_no_events_clf = SGDClassifier(loss='hinge',alpha=10, class_weight='balanced'
svm_no_events_clf.fit(X_all_train, Y_all_train)
#Using Model Calibration
svm_no_events_sig_clf = CalibratedClassifierCV(svm_no_events_clf, method="sigmoi
svm_no_events_sig_clf.fit(X_all_train, Y_all_train)
svm_no_events_predict_y = svm_no_events_sig_clf.predict_proba(X_all_cv)
print('For values of alpha = ', c, "The log loss is:",log_loss(Y_all_cv, svm_no_
```

```
For values of alpha =  10000 The log loss is: 2.423863380592673
```

In [218]:
```
#printing log loss
svm_no_events_pred_train = svm_no_events_sig_clf.predict_proba(X_all_train)
print("Train Log-Loss is",log_loss(Y_all_train, svm_no_events_pred_train, labels
svm_no_events_pred_cv = svm_no_events_sig_clf.predict_proba(X_all_cv)
print("CV Log-Loss is",log_loss(Y_all_cv, svm_no_events_pred_cv, labels=svm_no_e
```

```
Train Log-Loss is 2.4244531036160546
CV Log-Loss is 2.423863380592673
```

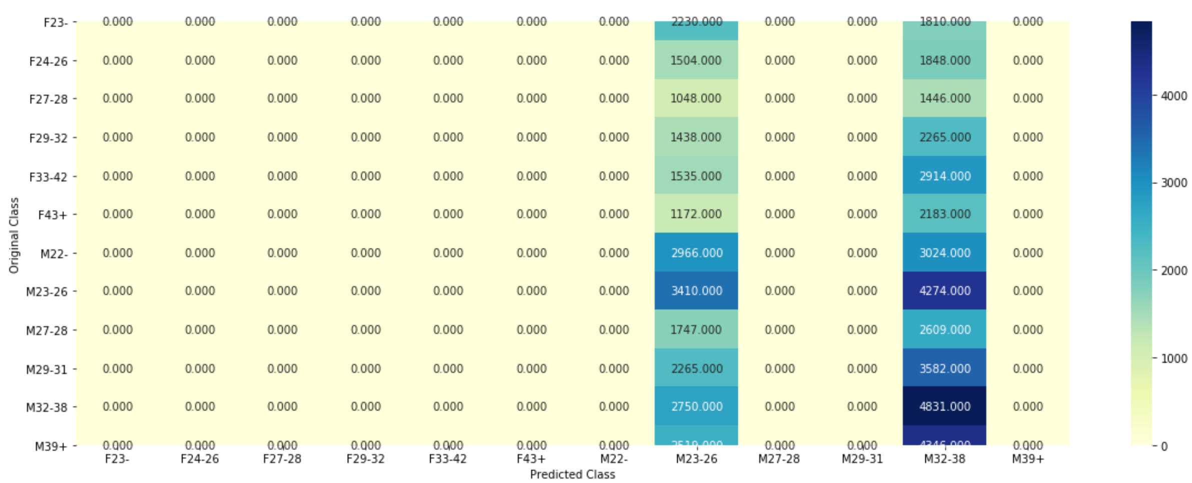
In [219]:
```
y_pred_no_events_train_classes_svm = np.argmax(svm_no_events_pred_train, axis=1)
y_pred_no_events_cv_classes_svm = np.argmax(svm_no_events_pred_cv, axis=1)
```

In [220]:
```
#printing comfusion matrix
print("Train Log Loss :",log_loss(Y_all_train, svm_no_events_pred_train))
plot_confusion_matrix(Y_all_train,y_pred_no_events_train_classes_svm)

print("="*60)

print("CV Log Loss :",log_loss(Y_all_cv, svm_no_events_pred_cv))
plot_confusion_matrix(Y_all_cv,y_pred_no_events_cv_classes_svm)
```

```
Train Log Loss : 2.4244531036160546
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [221]:
```
y_no_events_pred_svm = svm_no_events_sig_clf.predict_proba(X_no_events_test)
```

In [283]:
```
# Predicting Multiclass labels with test data
prediction_no_events_svm = pd.DataFrame(y_no_events_pred_svm,index = no_events_da
prediction_no_events_svm.head()
prediction_no_events_svm.to_csv("Predictions/prediction_no_events_svm.csv")
```

5.1.3. Logistic Regression

In [99]:
```
# # Loading the Logistic Regression model
# log_reg_no_events_sig_clf = joblib.load('Models/log_reg_no_events/no_events_log
```

In [172]:
```
hyperparameters = [10**-4, 10**-3,10**-2,10**-1,10**0, 10**1, 10**2,10**3,10**4]
for c in hyperparameters:
    log_reg_no_events_clf = SGDClassifier(alpha=c, class_weight='balanced', loss=
    log_reg_no_events_clf.fit(X_all_train, Y_all_train)
    #Using Model Calibration
    log_reg_no_events_sig_clf = CalibratedClassifierCV(log_reg_no_events_clf, me
    log_reg_no_events_sig_clf.fit(X_all_train, Y_all_train)
    log_reg_no_events_predict_y = log_reg_no_events_sig_clf.predict_proba(X_all_
    print('For values of C = ', c, "The log loss is:",log_loss(Y_all_cv, log_reg_
```

```
For values of C =  0.0001 The log loss is: 2.4037770982196105
For values of C =  0.001 The log loss is: 2.4015651343122024
For values of C =  0.01 The log loss is: 2.3952670464124797
For values of C =  0.1 The log loss is: 2.388942777120694
For values of C =  1 The log loss is: 2.390846335435144
For values of C =  10 The log loss is: 2.3988918239964168
For values of C =  100 The log loss is: 2.401727758751017
For values of C =  1000 The log loss is: 2.4019927710476208
For values of C =  10000 The log loss is: 2.401976758489355
```

In [287]:
```
# Training with best hyper parameters
best_c = 10**-1
log_reg_no_events_clf = SGDClassifier(alpha=c, class_weight='balanced', loss='lo
log_reg_no_events_clf.fit(X_all_train, Y_all_train)
log_reg_no_events_sig_clf = CalibratedClassifierCV(log_reg_no_events_clf, method
log_reg_no_events_sig_clf.fit(X_all_train, Y_all_train)
```

Out[287]:
```
CalibratedClassifierCV(base_estimator=LogisticRegression(C=0.1,
                                                         class_weight='balance
d',
                                                         dual=False,
                                                         fit_intercept=True,
                                                         intercept_scaling=1,
                                                         l1_ratio=None,
                                                         max_iter=100,
                                                         multi_class='multinomi
al',
                                                         n_jobs=None,
                                                         penalty='l2',
                                                         random_state=42,
                                                         solver='lbfgs',
                                                         tol=0.0001, verbose=0,
                                                         warm_start=False),
                       cv='warn', method='sigmoid')
```

In [11]:
```
# #Saving the best model
# jobl.dump(log_reg_no_events_sig_clf,'Models/log_reg_no_events/no_events_logist
```

In [100]:
```
#printing log loss
log_reg_no_events_pred_train = log_reg_no_events_sig_clf.predict_proba(X_all_tra
print("Train Log-Loss is",log_loss(Y_all_train, log_reg_no_events_pred_train, la
log_reg_no_events_pred_cv = log_reg_no_events_sig_clf.predict_proba(X_all_cv)
print("CV Log-Loss is",log_loss(Y_all_cv, log_reg_no_events_pred_cv, labels=log_
```

```
Train Log-Loss is 2.3676627576203098
CV Log-Loss is 2.389722153770035
```

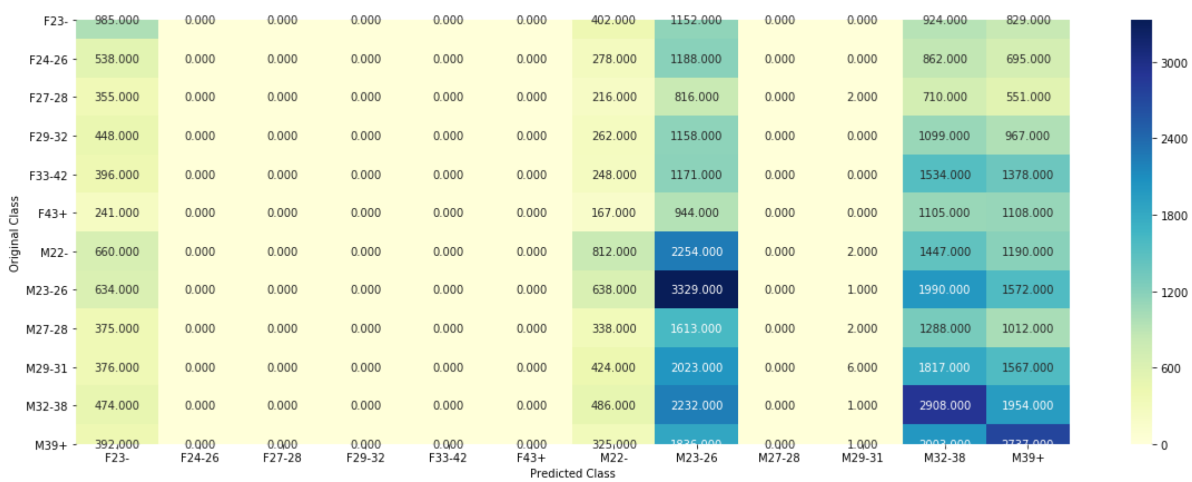
In [101]:
```
y_pred_no_events_train_classes_log_reg = np.argmax(log_reg_no_events_pred_train,
y_pred_no_events_cv_classes_log_reg = np.argmax(log_reg_no_events_pred_cv, axis=
```

In [102]:
```
#printing confusion matrix
print("Train Log Loss :",log_loss(Y_all_train, log_reg_no_events_pred_train))
plot_confusion_matrix(Y_all_train,y_pred_no_events_train_classes_log_reg)

print("="*60)

print("CV Log Loss :",log_loss(Y_all_cv, log_reg_no_events_pred_cv))
plot_confusion_matrix(Y_all_cv,y_pred_no_events_cv_classes_log_reg)
```

```
Train Log Loss : 2.3676627576203098
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [103]:
```
y_no_events_pred_logistic_regression = log_reg_no_events_sig_clf.predict_proba(X_
```

In [104]:
```
#Predicting multiclass label on test data
prediction_no_events_log_reg = pd.DataFrame(y_no_events_pred_logistic_regression
prediction_no_events_log_reg.head()
prediction_no_events_log_reg.to_csv("Predictions/prediction_no_events_log_reg.cs
```

5.1.4. XGBoost

In [171]:
```
max_depth= [3,4,5,6,7]
n_estimator = [250,350,450,550]
for depth in max_depth:
    for estimator in n_estimator:
        xgb_no_events_clf=XGBClassifier(n_estimators=estimator,n_jobs=-1,learnin
        xgb_no_events_clf.fit(X_all_train, Y_all_train)
        xgb_no_events_sig_clf = CalibratedClassifierCV(xgb_no_events_clf)
        xgb_no_events_sig_clf.fit(X_all_train, Y_all_train)
        xgb_no_events_predict_y = xgb_no_events_sig_clf.predict_proba(X_all_cv)
        print ('log_loss for best parameter is',log_loss(Y_all_cv, xgb_no_events_
```

```
log_loss for n_estimator =  250 and maxdepth = 3 is 2.395826959726325
log_loss for n_estimator =  350 and maxdepth = 3 is 2.3953403486348774
log_loss for n_estimator =  450 and maxdepth = 3 is 2.3952853908747875
log_loss for n_estimator =  550 and maxdepth = 3 is 2.395239102536255
log_loss for n_estimator =  250 and maxdepth = 4 is 2.395420964942357
log_loss for n_estimator =  350 and maxdepth = 4 is 2.3952504654943163
log_loss for n_estimator =  450 and maxdepth = 4 is 2.3953852872052015
log_loss for n_estimator =  550 and maxdepth = 4 is 2.3955115497125967
log_loss for n_estimator =  250 and maxdepth = 5 is 2.3952598872177115
log_loss for n_estimator =  350 and maxdepth = 5 is 2.3953507968319134
log_loss for n_estimator =  450 and maxdepth = 5 is 2.3956529895062033
log_loss for n_estimator =  550 and maxdepth = 5 is 2.395849367668471
log_loss for n_estimator =  250 and maxdepth = 6 is 2.395245538225763
log_loss for n_estimator =  350 and maxdepth = 6 is 2.3955112088336312
log_loss for n_estimator =  450 and maxdepth = 6 is 2.395922944688156
log_loss for n_estimator =  550 and maxdepth = 6 is 2.3961348776325466
log_loss for n_estimator =  250 and maxdepth = 7 is 2.3953494625334297
log_loss for n_estimator =  350 and maxdepth = 7 is 2.395726749815855
log_loss for n_estimator =  450 and maxdepth = 7 is 2.3961430327310773
log_loss for n_estimator =  550 and maxdepth = 7 is 2.3962832515671866
```

In [293]:
```
xgb_no_events_clf=XGBClassifier(n_estimators=250,n_jobs=-1,learning_rate=0.05, c
xgb_no_events_clf.fit(X_all_train, Y_all_train)
xgb_no_events_sig_clf = CalibratedClassifierCV(xgb_no_events_clf)
xgb_no_events_sig_clf.fit(X_all_train, Y_all_train)
xgb_no_events_predict_y = xgb_no_events_sig_clf.predict_proba(X_all_cv)
print ('log_loss for n_estimator = ',350,'and maxdepth =',5,'is',log_loss(Y_all_
```

```
log_loss for n_estimator =  350 and maxdepth = 5 is 2.395120537544309
```

In [294]:
```
xgb_no_events_pred_train = xgb_no_events_sig_clf.predict_proba(X_all_train)
print("Train Log-Loss is",log_loss(Y_all_train, xgb_no_events_pred_train, labels
xgb_no_events_pred_cv = xgb_no_events_sig_clf.predict_proba(X_all_cv)
print("CV Log-Loss is",log_loss(Y_all_cv, xgb_no_events_pred_cv, labels=xgb_no_e
```

```
Train Log-Loss is 2.3756413248365487
CV Log-Loss is 2.395120537544309
```

In [295]:
```
y_pred_no_events_train_classes_xgb = np.argmax(xgb_no_events_pred_train, axis=1)
y_pred_no_events_cv_classes_xgb = np.argmax(xgb_no_events_pred_cv, axis=1)
```

In [296]:
```
print("Train Log Loss :",log_loss(Y_all_train, xgb_no_events_pred_train))
plot_confusion_matrix(Y_all_train,y_pred_no_events_train_classes_xgb)

print("="*60)

print("CV Log Loss :",log_loss(Y_all_cv, xgb_no_events_pred_cv))
plot_confusion_matrix(Y_all_cv,y_pred_no_events_cv_classes_xgb)
```

```
Train Log Loss : 2.3756413248365487
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [297]:
```
y_no_events_pred_xgboost = xgb_no_events_sig_clf.predict_proba(X_no_events_test)
```

In [298]:
```
# Multilabel class probalities are predicted
prediction_no_events_xgb = pd.DataFrame(y_no_events_pred_xgboost,index = no_even
prediction_no_events_xgb.head()
prediction_no_events_xgb.to_csv("Predictions/prediction_no_events_xgb.csv")
```

5.1.5. Random Forest

In [181]:
```
from sklearn.ensemble import RandomForestClassifier
max_depth = [3, 4, 5, 6]
n_estimator = [350,450,550]
for depth in max_depth:
    for estimator in n_estimator:
        rf_no_events_clf=RandomForestClassifier(n_estimators=estimator,n_jobs=-1
        rf_no_events_clf.fit(X_all_train, Y_all_train)
        rf_no_events_sig_clf = CalibratedClassifierCV(rf_no_events_clf, method="
        rf_no_events_sig_clf.fit(X_all_train, Y_all_train)
        rf_no_events_predict_y = rf_no_events_sig_clf.predict_proba(X_all_cv)
        print ('log_loss for n_estimator = ',estimator,'and maxdepth =',depth,'i
```

```
log_loss for n_estimator =  350 and maxdepth = 3 is 2.397361243391592
log_loss for n_estimator =  450 and maxdepth = 3 is 2.3969469106584156
log_loss for n_estimator =  550 and maxdepth = 3 is 2.3964033137049454
log_loss for n_estimator =  350 and maxdepth = 4 is 2.3970872500512916
log_loss for n_estimator =  450 and maxdepth = 4 is 2.396206333390335
log_loss for n_estimator =  550 and maxdepth = 4 is 2.396180574192794
log_loss for n_estimator =  350 and maxdepth = 5 is 2.3960099566379136
log_loss for n_estimator =  450 and maxdepth = 5 is 2.395699885064684
log_loss for n_estimator =  550 and maxdepth = 5 is 2.3959056435771715
log_loss for n_estimator =  350 and maxdepth = 6 is 2.3955135928858557
log_loss for n_estimator =  450 and maxdepth = 6 is 2.3952124925686045
log_loss for n_estimator =  550 and maxdepth = 6 is 2.3956913386286205
```

In [184]:
```
rf_no_events_clf=RandomForestClassifier(n_estimators=350,n_jobs=-1, max_depth=5)
rf_no_events_clf.fit(X_all_train, Y_all_train)
rf_no_events_sig_clf = CalibratedClassifierCV(rf_no_events_clf, method="sigmoid"
rf_no_events_sig_clf.fit(X_all_train, Y_all_train)
rf_no_events_predict_y = rf_no_events_sig_clf.predict_proba(X_all_cv)
print ('log_loss for n_estimator = ',estimator,'and maxdepth =',depth,'is',log_l
```

```
log_loss for n_estimator =  550 and maxdepth = 6 is 2.3958202185243205
```

In [185]:
```
rf_no_events_pred_train = rf_no_events_sig_clf.predict_proba(X_all_train)
print("Train Log-Loss is",log_loss(Y_all_train, rf_no_events_pred_train, labels=
rf_no_events_pred_cv = rf_no_events_sig_clf.predict_proba(X_all_cv)
print("CV Log-Loss is",log_loss(Y_all_cv, rf_no_events_pred_cv, labels=rf_no_eve
```

```
Train Log-Loss is 2.387445232067769
CV Log-Loss is 2.3958202185243205
```

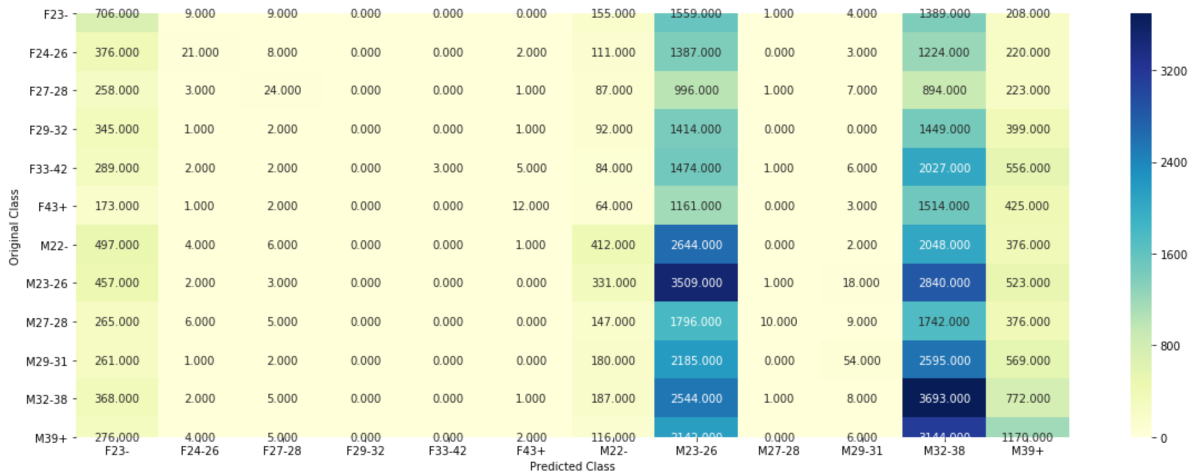
In [189]:
```
y_pred_no_events_train_classes_rf = np.argmax(rf_no_events_pred_train, axis=1)
y_pred_no_events_cv_classes_rf = np.argmax(rf_no_events_pred_cv, axis=1)
```

In [190]:
```
print("Train Log Loss :",log_loss(Y_all_train, rf_no_events_pred_train))
plot_confusion_matrix(Y_all_train,y_pred_no_events_train_classes_rf)

print("="*60)

print("CV Log Loss :",log_loss(Y_all_cv, rf_no_events_pred_cv))
plot_confusion_matrix(Y_all_cv,y_pred_no_events_cv_classes_rf)
```

```
Train Log Loss : 2.387445232067769
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Column Sum=1) --------------------
```

In [191]:
```
y_no_events_pred_random_forest = rf_no_events_sig_clf.predict_proba(X_no_events_
```

In [299]:
```
# Multilabel class probalities are predicted
prediction_no_events_rf = pd.DataFrame(y_no_events_pred_random_forest,index = no_
prediction_no_events_rf.head()
prediction_no_events_rf.to_csv("Predictions/prediction_no_events_rf.csv")
```

5.1.6. Decision Tree

In [197]:
```
from sklearn.tree import DecisionTreeClassifier
max_depth = [3, 4, 5, 6]
min_samples_split_values = [5,10,100,500]
for depth in max_depth:
    for min_sample in min_samples_split_values:
        dt_no_events_clf=DecisionTreeClassifier(min_samples_split=min_sample,max_
        dt_no_events_clf.fit(X_all_train, Y_all_train)
        dt_no_events_sig_clf = CalibratedClassifierCV(dt_no_events_clf, method="
        dt_no_events_sig_clf.fit(X_all_train, Y_all_train)
        dt_no_events_predict_y = dt_no_events_sig_clf.predict_proba(X_all_cv)
        print ('log_loss for min_sample_split = ',min_sample,'and maxdepth =',de
```

```
log_loss for min_sample_split =  5 and maxdepth = 3 is 2.410057339012505
log_loss for min_sample_split =  10 and maxdepth = 3 is 2.410057339012505
log_loss for min_sample_split =  100 and maxdepth = 3 is 2.410057339012505
log_loss for min_sample_split =  500 and maxdepth = 3 is 2.410057339012505
log_loss for min_sample_split =  5 and maxdepth = 4 is 2.4063630303784396
log_loss for min_sample_split =  10 and maxdepth = 4 is 2.4063630303784396
log_loss for min_sample_split =  100 and maxdepth = 4 is 2.4063630303784396
log_loss for min_sample_split =  500 and maxdepth = 4 is 2.4063630303784396
log_loss for min_sample_split =  5 and maxdepth = 5 is 2.4040081530200292
log_loss for min_sample_split =  10 and maxdepth = 5 is 2.4040081530200292
log_loss for min_sample_split =  100 and maxdepth = 5 is 2.4040081530200292
log_loss for min_sample_split =  500 and maxdepth = 5 is 2.4040081530200292
log_loss for min_sample_split =  5 and maxdepth = 6 is 2.402481529173549
log_loss for min_sample_split =  10 and maxdepth = 6 is 2.402481529173549
log_loss for min_sample_split =  100 and maxdepth = 6 is 2.402481529173549
log_loss for min_sample_split =  500 and maxdepth = 6 is 2.402481529173549
```

In [199]:
```
dt_no_events_clf=DecisionTreeClassifier(min_samples_split=100,max_depth=5,class_
dt_no_events_clf.fit(X_all_train, Y_all_train)
dt_no_events_sig_clf = CalibratedClassifierCV(dt_no_events_clf, method="sigmoid"
dt_no_events_sig_clf.fit(X_all_train, Y_all_train)
dt_no_events_predict_y = dt_no_events_sig_clf.predict_proba(X_all_cv)
print ('log_loss for min_sample_split = ',100,'and maxdepth =',5,'is',log_loss(Y_
```

```
log_loss for min_sample_split =  100 and maxdepth = 5 is 2.4040081530200292
```

In [200]:
```
dt_no_events_pred_train = dt_no_events_sig_clf.predict_proba(X_all_train)
print("Train Log-Loss is",log_loss(Y_all_train, dt_no_events_pred_train, labels=
dt_no_events_pred_cv = dt_no_events_sig_clf.predict_proba(X_all_cv)
print("CV Log-Loss is",log_loss(Y_all_cv, dt_no_events_pred_cv, labels=dt_no_eve
```

```
Train Log-Loss is 2.402364665110147
CV Log-Loss is 2.4040081530200292
```

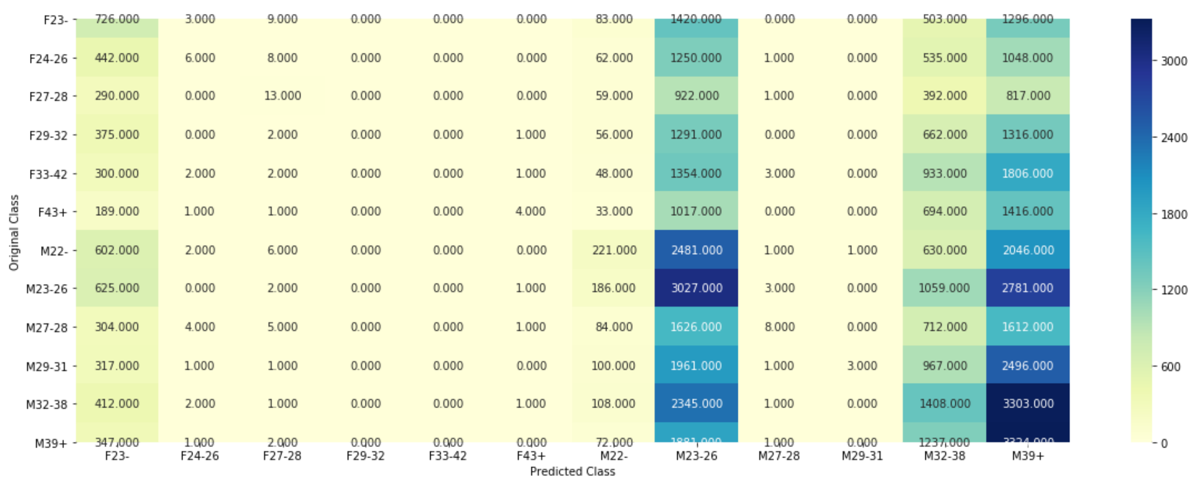
In [201]:
```
y_pred_no_events_train_classes_dt = np.argmax(dt_no_events_pred_train, axis=1)
y_pred_no_events_cv_classes_dt = np.argmax(dt_no_events_pred_cv, axis=1)
```

In [202]:
```
print("Train Log Loss :",log_loss(Y_all_train, dt_no_events_pred_train))
plot_confusion_matrix(Y_all_train,y_pred_no_events_train_classes_dt)

print("="*60)

print("CV Log Loss :",log_loss(Y_all_cv, dt_no_events_pred_cv))
plot_confusion_matrix(Y_all_cv,y_pred_no_events_cv_classes_dt)
```

```
Train Log Loss : 2.402364665110147
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [269]:
```
y_no_events_pred_decision_tree = dt_no_events_sig_clf.predict_proba(X_no_events_
```

In [300]:
```
# Multilabel class probalities are predicted
prediction_no_events_dt = pd.DataFrame(y_no_events_pred_decision_tree,index = no_
prediction_no_events_dt.head()
prediction_no_events_dt.to_csv("Predictions/prediction_no_events_dt.csv")
```

5.1.7. Deep Learning - Neural Networks

Neural network 1

The train data is trained with the below neural network model and validate against cv with different seed for train and test split. For prediction we are considering the average of model for each seed

In [106]:
```
# #Loading the models
# model_1 = []
# for i in range(5):
#     model_1.append(load_model('Models/no_events_1/no_events_nn_1_'+str(i+1)+'.l
```

In [512]:
```
%load_ext tensorboard
early_stop_1=EarlyStopping(monitor='val_loss',patience=5,restore_best_weights=Tru
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [506]:
```
#https://www.kaggle.com/c/talkingdata-mobile-user-demographics/discussion/23424
#Network architecture is refered from the above kaggle link
#Defining the model
def model_no_events_nn_1(input_shape):
    model = Sequential()
    model.add(Dense(256, input_dim=input_shape))
    model.add(PReLU())
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(64))
    model.add(PReLU())
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(12))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
    return model
```

In [507]:
```
#Model Summary
model_no_events_1=model_no_events_nn_1(x_ohe_train.shape[1])
model_no_events_1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 256)               443136    
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 256)               256       
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                16448     
_________________________________________________________________
p_re_lu_2 (PReLU)            (None, 64)                64        
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256       
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 12)                780       
_________________________________________________________________
activation_1 (Activation)    (None, 12)                0         
=================================================================
Total params: 461,964
Trainable params: 461,324
Non-trainable params: 640
_________________________________________________________________
```

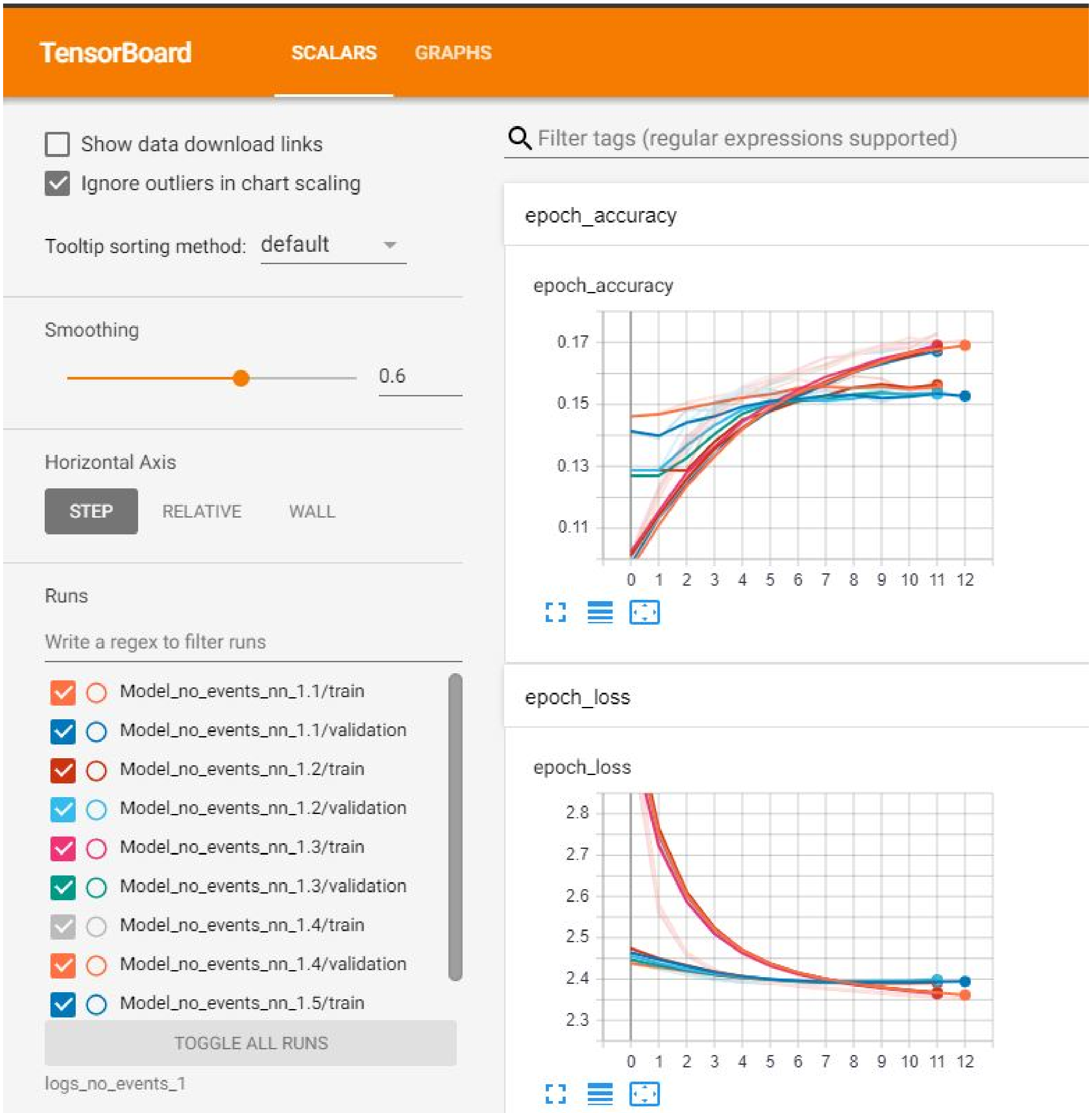
In [513]:
```
#Running the model with different seeds
seeds=[18,7,36,57,73]
model_1=[]
avg_cv_loss=0
for i in range(len(seeds)):
    train, cv, y_train, y_cv = train_test_split(x_ohe_train,Y_all,stratify=Y_all
    y_train_nn=np_utils.to_categorical(y_train)
    y_cv_nn=np_utils.to_categorical(y_cv)
    model=model_no_events_nn_1(train.shape[1])
    logdir = os.path.join("logs_no_events_1","Model_no_events_1."+str(i+1))
    t_callback=TensorBoard(log_dir=logdir)
    model.fit(train, y_train_nn, batch_size=350, epochs=30, verbose=1, validatio
    model_cv_prediction=model.predict_proba(cv)
    cv_loss=log_loss(y_cv, model_cv_prediction)
    print("CV Log Loss of Best Weights Model in Current Run: ",cv_loss)
    model_1.append(model)
    avg_cv_loss+=cv_loss
avg_cv_loss/=len(seeds)
print("Average CV Loss of "+str(len(seeds))+" Runs :",avg_cv_loss)
```

```
Train on 63448 samples, validate on 11197 samples
Epoch 1/30
63448/63448 [==============================] - ETA: 1:53 - loss: 3.9864 - acc
uracy: 0.06 - ETA: 43s - loss: 3.8690 - accuracy: 0.0657 - ETA: 23s - loss:
3.7506 - accuracy: 0.077 - ETA: 18s - loss: 3.7320 - accuracy: 0.080 - ETA: 1
4s - loss: 3.6947 - accuracy: 0.082 - ETA: 12s - loss: 3.6691 - accuracy: 0.0
82 - ETA: 10s - loss: 3.6305 - accuracy: 0.085 - ETA: 9s - loss: 3.6101 - acc
uracy: 0.084 - ETA: 8s - loss: 3.5836 - accuracy: 0.08 - ETA: 7s - loss: 3.55
20 - accuracy: 0.08 - ETA: 6s - loss: 3.5325 - accuracy: 0.08 - ETA: 6s - los
s: 3.5132 - accuracy: 0.08 - ETA: 6s - loss: 3.4906 - accuracy: 0.08 - ETA: 5
s - loss: 3.4709 - accuracy: 0.08 - ETA: 5s - loss: 3.4581 - accuracy: 0.08 -
ETA: 5s - loss: 3.4470 - accuracy: 0.08 - ETA: 4s - loss: 3.4284 - accuracy:
0.08 - ETA: 4s - loss: 3.4129 - accuracy: 0.09 - ETA: 4s - loss: 3.3933 - acc
uracy: 0.09 - ETA: 4s - loss: 3.3771 - accuracy: 0.09 - ETA: 4s - loss: 3.359
7 - accuracy: 0.09 - ETA: 3s - loss: 3.3445 - accuracy: 0.09 - ETA: 3s - los
s: 3.3322 - accuracy: 0.09 - ETA: 3s - loss: 3.3164 - accuracy: 0.09 - ETA: 3
s - loss: 3.3036 - accuracy: 0.09 - ETA: 3s - loss: 3.2880 - accuracy: 0.09 -
ETA: 3s - loss: 3.2750 - accuracy: 0.09 - ETA: 3s - loss: 3.2609 - accuracy:
0.09 - ETA: 2s - loss: 3.2448 - accuracy: 0.09 - ETA: 2s - loss: 3.2322 - acc
```

In [ ]:
```
# #Saving the models
# for i in range(len(model_list_1)):
#     model_list_1[i].save('Models/no_events_1/no_events_nn_1_'+str(i+1)+'.h5')
```

In [180]:
```
Image('Pred/tensorboard1.jpg')
```

Out[180]:



Train Prediction

In [107]:
```
#Predicting Train class labels probality
#Taking average of value predicted for each seed
train_pred_avg_no_events_1=np.zeros((X_all_train.shape[0],12))
for i in range(len(model_1)):
    train_pred=model_1[i].predict_proba(X_all_train)
    train_pred_avg_no_events_1+=train_pred
train_pred_avg_no_events_1/=len(model_1)
```

CV Prediction

In [110]:
```
#Predicting CV class labels probalities
#Taking average of value predicted for each seed
cv_pred_avg_no_events_1=np.zeros((X_all_cv.shape[0],12))
for i in range(len(model_1)):
    cv_pred=model_1[i].predict_proba(X_all_cv)
    cv_pred_avg_no_events_1+=cv_pred
cv_pred_avg_no_events_1/=len(model_1)
```

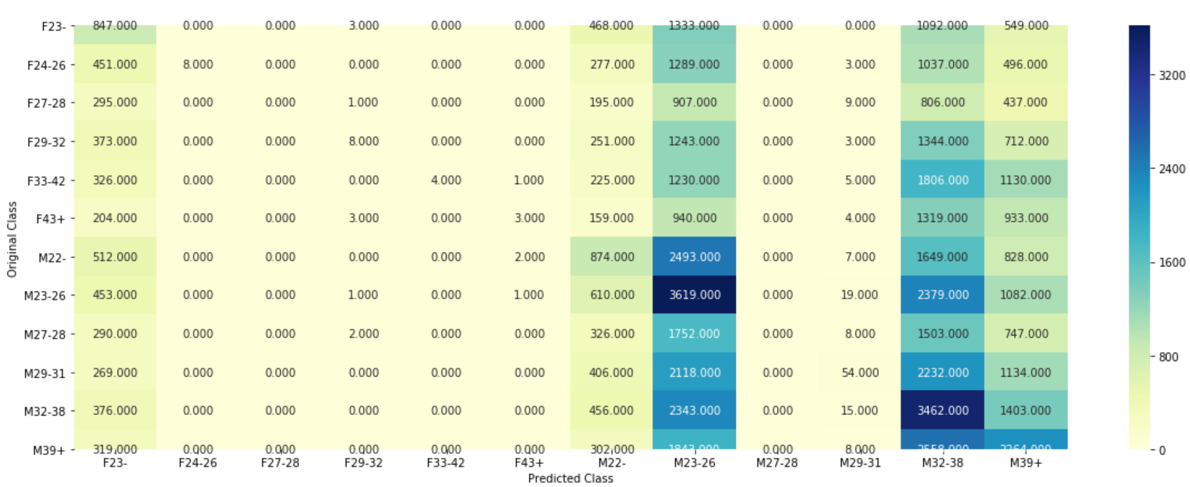
Confusion Matrix

In [111]:
```
#Printing Confusion Matrix For Train and CV preditcions
print("Train Average Log-Loss: ",log_loss(Y_all_train, train_pred_avg_no_events_1
no_events_nn_1_avg_train_predicted_classes=np.argmax(train_pred_avg_no_events_1,
plot_confusion_matrix(Y_all_train,no_events_nn_1_avg_train_predicted_classes)

print(60*'=')

print("CV Average Log-Loss: ",log_loss(Y_all_cv, cv_pred_avg_no_events_1))
no_events_nn_1_avg_cv_predicted_classes=np.argmax(cv_pred_avg_no_events_1, axis=
plot_confusion_matrix(Y_all_cv,no_events_nn_1_avg_cv_predicted_classes)
```

```
Train Average Log-Loss:  2.3575881615884535
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

Test Prediction

In [112]:
```
#Predicting the probabilities of class labels for Test data
#Taking average of value predicted for each seed
test_pred_avg_no_events_1=np.zeros((X_no_events_test.shape[0],12))
for i in range(len(model_1)):
    test_pred=model_1[i].predict_proba(X_no_events_test.tocsr())
    test_pred_avg_no_events_1+=test_pred
test_pred_avg_no_events_1/=len(model_1)
```

Neural network 2

In [113]:
```
#Loading NN model
model_no_events_2 = load_model('Models/no_events_2/no_events_nn_2.h5')
```

In [514]:
```
#https://www.kaggle.com/c/talkingdata-mobile-user-demographics/discussion/23424
#Refered from the above kaggle discussion
#Defining model
def model_no_events_nn_2(input_dim,output_dim, learRate=0.0025):

    model = Sequential()
    model.add(Dense(500, input_shape=(input_dim,), init='uniform'))
    model.add(PReLU(init='zero'))
    model.add(Dropout(0.82))
    model.add(Dense(output_dim, init='uniform'))
    model.add(Activation('softmax'))
    opt = Adagrad(lr=learRate, epsilon=1e-08)
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
    return model
```

In [515]:
```
model_no_events_2=model_no_events_nn_1(x_ohe_train.shape[1])
model_no_events_2.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 256)               443136    
_________________________________________________________________
p_re_lu_17 (PReLU)           (None, 256)               256       
_________________________________________________________________
batch_normalization_17 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_26 (Dense)             (None, 64)                16448     
_________________________________________________________________
p_re_lu_18 (PReLU)           (None, 64)                64        
_________________________________________________________________
batch_normalization_18 (Batc (None, 64)                256       
_________________________________________________________________
dropout_18 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_27 (Dense)             (None, 12)                780       
_________________________________________________________________
activation_9 (Activation)    (None, 12)                0         
=================================================================
Total params: 461,964
Trainable params: 461,324
Non-trainable params: 640
_________________________________________________________________
```

In [522]:
```
#Changin class labels to categorical values for NN
y_train_nn=np_utils.to_categorical(Y_all_train)
y_cv_nn=np_utils.to_categorical(Y_all_cv)
```

In [523]:
```
#Setting Log Directory
logdir = os.path.join("logs_no_events_2","Model_no_events_2")
t_callback_2=TensorBoard(log_dir=logdir)
```

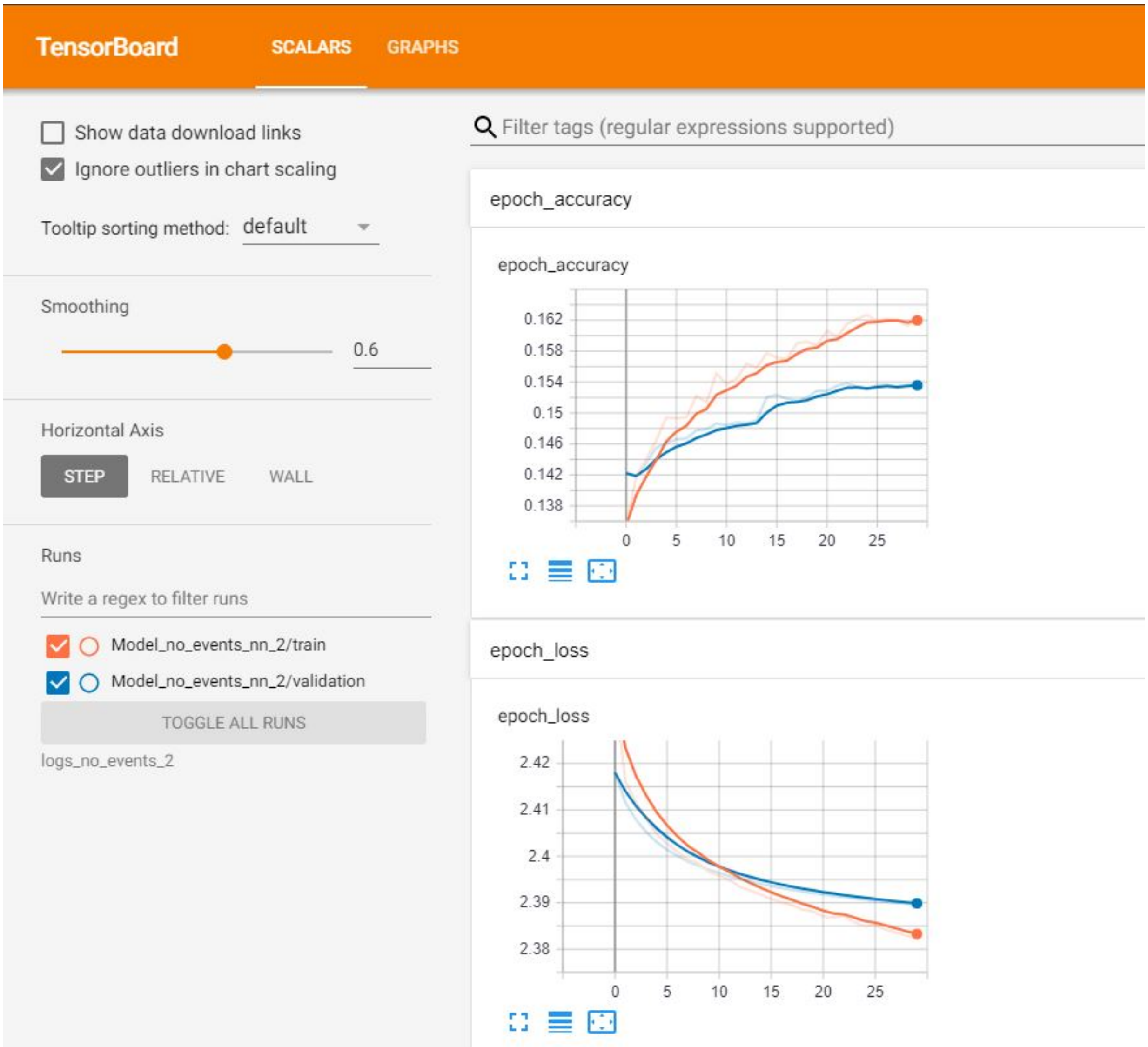
In [524]:
```
#Running model on train data and validating against test data
model_no_events_2.fit(X_all_train, y_train_nn, batch_size=350, epochs=30, verbos
```

```
Train on 63448 samples, validate on 11197 samples
Epoch 1/30
63448/63448 [==============================] - ETA: 1:37 - loss: 2.4014 - acc
uracy: 0.15 - ETA: 37s - loss: 2.3908 - accuracy: 0.1581 - ETA: 23s - loss:
2.3827 - accuracy: 0.153 - ETA: 16s - loss: 2.3744 - accuracy: 0.157 - ETA: 1
2s - loss: 2.3771 - accuracy: 0.158 - ETA: 11s - loss: 2.3746 - accuracy: 0.1
59 - ETA: 10s - loss: 2.3755 - accuracy: 0.157 - ETA: 8s - loss: 2.3735 - acc
uracy: 0.160 - ETA: 8s - loss: 2.3726 - accuracy: 0.16 - ETA: 7s - loss: 2.37
32 - accuracy: 0.15 - ETA: 7s - loss: 2.3753 - accuracy: 0.15 - ETA: 6s - los
s: 2.3743 - accuracy: 0.15 - ETA: 6s - loss: 2.3748 - accuracy: 0.15 - ETA: 6
s - loss: 2.3749 - accuracy: 0.15 - ETA: 5s - loss: 2.3762 - accuracy: 0.15 -
ETA: 5s - loss: 2.3753 - accuracy: 0.16 - ETA: 5s - loss: 2.3749 - accuracy:
0.16 - ETA: 5s - loss: 2.3754 - accuracy: 0.16 - ETA: 4s - loss: 2.3766 - acc
uracy: 0.16 - ETA: 4s - loss: 2.3765 - accuracy: 0.16 - ETA: 4s - loss: 2.377
3 - accuracy: 0.16 - ETA: 4s - loss: 2.3763 - accuracy: 0.16 - ETA: 4s - los
s: 2.3752 - accuracy: 0.16 - ETA: 4s - loss: 2.3759 - accuracy: 0.16 - ETA: 4
s - loss: 2.3759 - accuracy: 0.16 - ETA: 4s - loss: 2.3767 - accuracy: 0.16 -
ETA: 3s - loss: 2.3757 - accuracy: 0.16 - ETA: 3s - loss: 2.3762 - accuracy:
0.16 - ETA: 3s - loss: 2.3766 - accuracy: 0.16 - ETA: 3s - loss: 2.3761 - acc
```

In [ ]:
```
# #Saving the model
# model_no_events_2.save('Models/no_events_2/no_events_nn_2.h5')
```

In [181]:
```
Image('Pred/tensorboard2.jpg')
```

Out[181]:

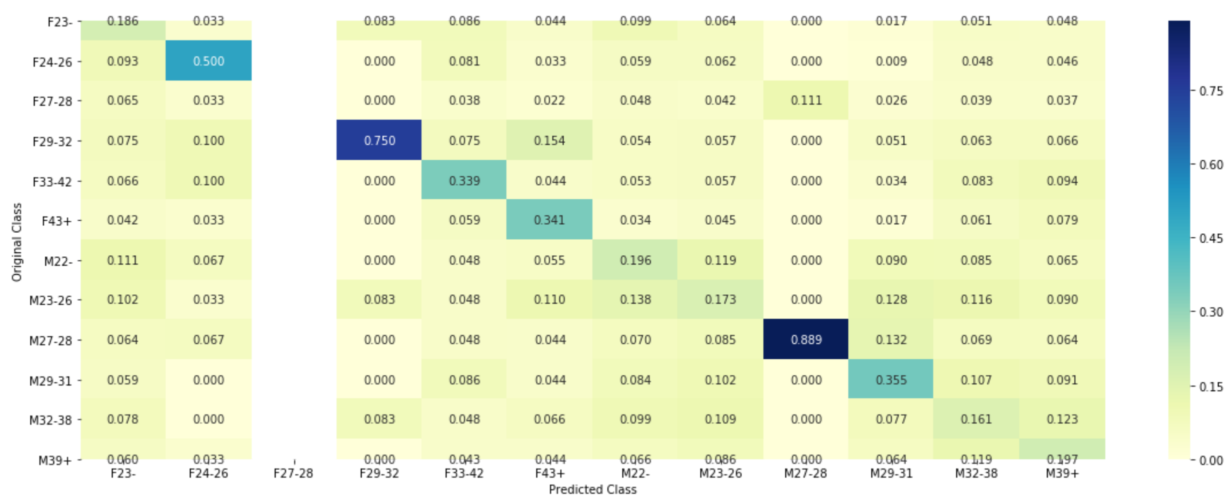


Train Prediction and Confusion Matrix

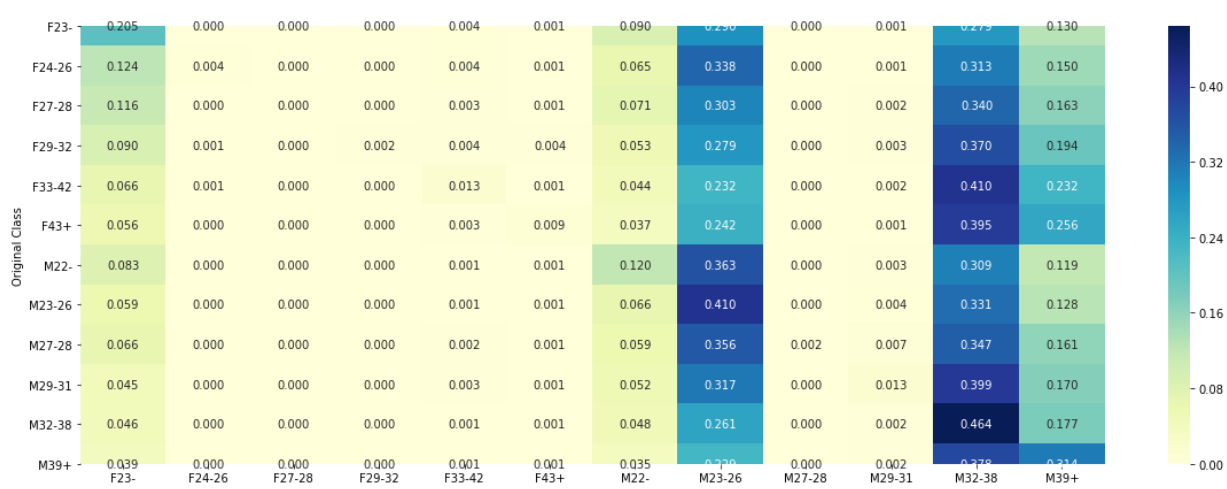
In [114]:
```
train_pred_avg_no_events_2=model_no_events_2.predict_proba(X_all_train)
print("Train Log-Loss: ",log_loss(Y_all_train, train_pred_avg_no_events_2))
no_events_nn_2_train_predicted_classes=np.argmax(train_pred_avg_no_events_2, axi
plot_confusion_matrix(Y_all_train,no_events_nn_2_train_predicted_classes)
```

```
Train Log-Loss:  2.352695338143375
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



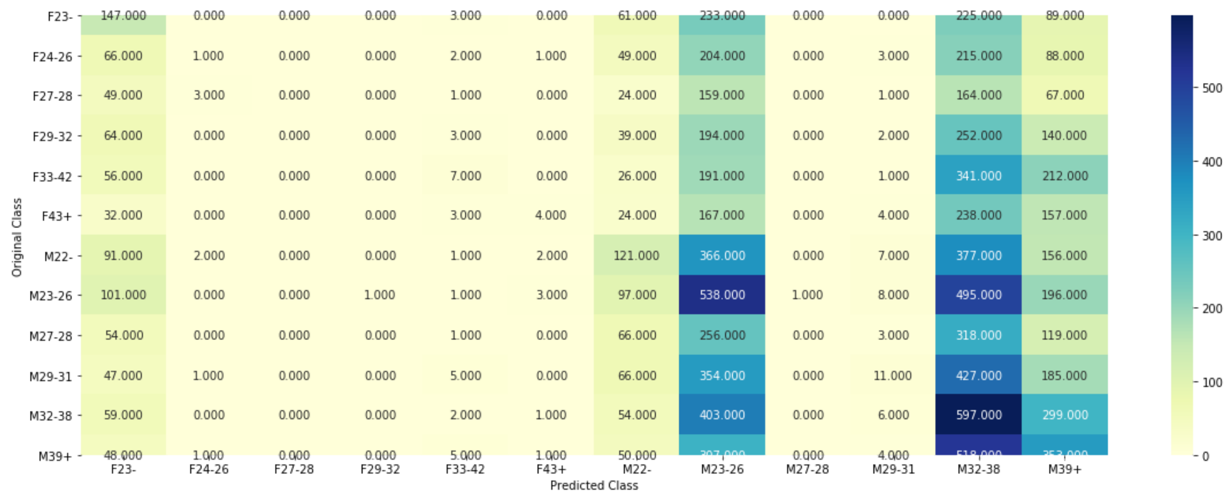

```
-------------------- Recall matrix (Row sum=1) --------------------
```



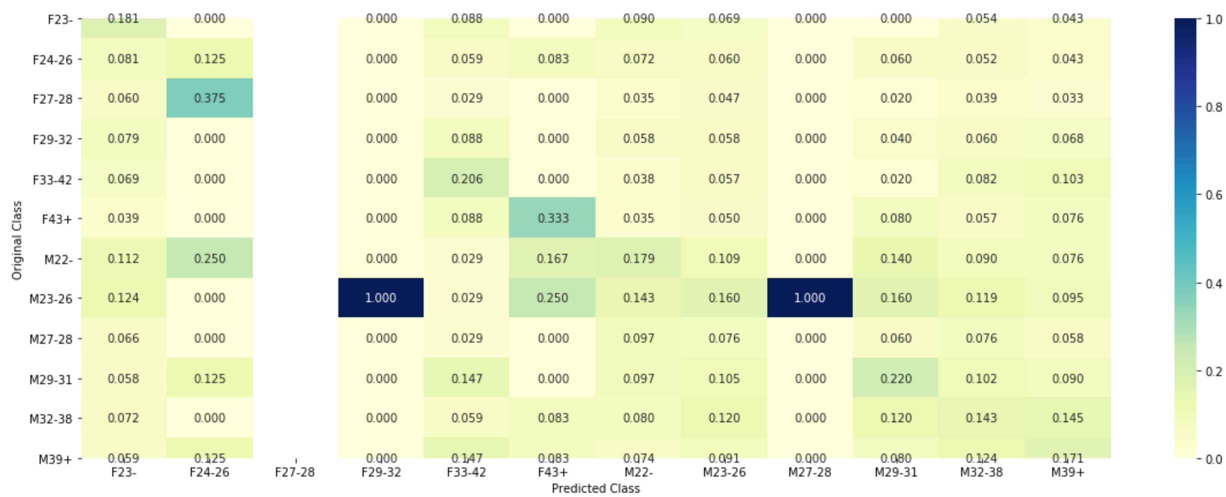
CV Prediction and Confusion Matrix

In [115]:
```
cv_pred_avg_no_events_2=model_no_events_2.predict_proba(X_all_cv)
print("Train Log-Loss: ",log_loss(Y_all_cv, cv_pred_avg_no_events_2))
no_events_nn_2_cv_predicted_classes=np.argmax(cv_pred_avg_no_events_2, axis=1)
plot_confusion_matrix(Y_all_cv,no_events_nn_2_cv_predicted_classes)
```

```
Train Log-Loss:  2.387814980548887
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

Test Prediction

In [116]:
```
test_pred_avg_no_events_2=model_no_events_2.predict_proba(X_no_events_test.tocsr
```

5.2. Events Device

In [117]:
```
#Fetching the class labels
Y_events = having_events_data_train['group'].values
```

In [118]:
```
#Performing Train Test Split
X_events_train , X_events_cv , Y_events_train , Y_events_cv = train_test_split(X_
X_events_test = X_having_events_test
```

5.2.1. KNN

In [229]:
```
from sklearn.neighbors import KNeighborsClassifier
k_size = [5,10,50,100]
for k_neighbour in k_size:
    knn_events_clf = KNeighborsClassifier(n_neighbors=k_neighbour, n_jobs=-1)
    knn_events_clf.fit(X_events_train, Y_events_train)
    #Using Model Calibration
    knn_events_sig_clf = CalibratedClassifierCV(knn_events_clf, method="sigmoid"
    knn_events_sig_clf.fit(X_events_train, Y_events_train)
    knn_events_predict_y = knn_events_sig_clf.predict_proba(X_events_cv)
    print('For values of K = ', k_neighbour, "The log loss is:",log_loss(Y_event
```

```
For values of K =  5 The log loss is: 2.343914246548062
For values of K =  10 The log loss is: 2.3143817272282066
For values of K =  50 The log loss is: 2.2430287947253147
For values of K =  100 The log loss is: 2.2120937376664918
```

In [230]:
```
knn_events_clf = KNeighborsClassifier(n_neighbors=50, n_jobs=-1)
knn_events_clf.fit(X_events_train, Y_events_train)
#Using Model Calibration
knn_events_sig_clf = CalibratedClassifierCV(knn_events_clf, method="sigmoid")
knn_events_sig_clf.fit(X_events_train, Y_events_train)
knn_events_predict_y = knn_events_sig_clf.predict_proba(X_events_cv)
print('For values of K = ', k_neighbour, "The log loss is:",log_loss(Y_events_cv
```

```
For values of K =  100 The log loss is: 2.2430287947253147
```

In [231]:
```
knn_events_pred_train = knn_events_sig_clf.predict_proba(X_events_train)
print("Train Log-Loss is",log_loss(Y_events_train, knn_events_pred_train, labels
knn_events_pred_cv = knn_events_sig_clf.predict_proba(X_events_cv)
print("CV Log-Loss is",log_loss(Y_events_cv, knn_events_pred_cv, labels=knn_even
```

```
Train Log-Loss is 2.153761278936381
CV Log-Loss is 2.2430287947253147
```

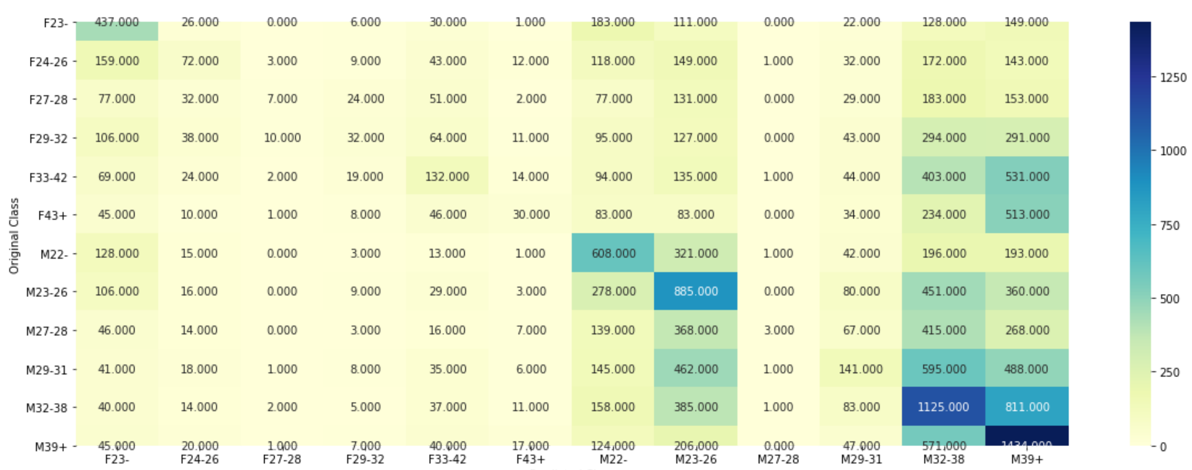
In [232]:
```
y_pred_events_train_classes_knn = np.argmax(knn_events_pred_train, axis=1)
y_pred_events_cv_classes_knn = np.argmax(knn_events_pred_cv, axis=1)
```

In [233]:
```
print("Train Log Loss :",log_loss(Y_events_train, knn_events_pred_train))
plot_confusion_matrix(Y_events_train,y_pred_events_train_classes_knn)

print("="*60)

print("CV Log Loss :",log_loss(Y_events_cv, knn_events_pred_cv))
plot_confusion_matrix(Y_events_cv,y_pred_events_cv_classes_knn)
```

```
Train Log Loss : 2.153761278936381
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [301]:
```
y_events_pred_k_nearest_neighbours = knn_events_sig_clf.predict_proba(X_events_te
# Multilabel class probalities are predicted
prediction_events_knn = pd.DataFrame(y_events_pred_k_nearest_neighbours,index = 
prediction_events_knn.head()
prediction_events_knn.to_csv("Predictions/prediction_events_knn.csv")
```

5.2.2. SVM

In [237]:
```
from sklearn.linear_model import SGDClassifier
hyperparameters = [10**-2,10**-1,10**0, 10**1]
for c in hyperparameters:
    svm_events_clf = SGDClassifier(loss='hinge',alpha=c, class_weight='balanced',
    svm_events_clf.fit(X_events_train, Y_events_train)
    #Using Model Calibration
    svm_events_sig_clf = CalibratedClassifierCV(svm_events_clf, method="sigmoid"
    svm_events_sig_clf.fit(X_events_train, Y_events_train)
    svm_events_predict_y = svm_events_sig_clf.predict_proba(X_events_cv)
    print('For values of alpha = ', c, "The log loss is:",log_loss(Y_events_cv, s
```

```
For values of alpha =  0.01 The log loss is: 2.117849528589639
For values of alpha =  0.1 The log loss is: 2.2293116608139094
For values of alpha =  1 The log loss is: 2.3672057490767573
For values of alpha =  10 The log loss is: 2.40003498589291
```

In [238]:
```
svm_events_clf = SGDClassifier(loss='hinge',alpha=10**-1, class_weight='balanced
svm_events_clf.fit(X_events_train, Y_events_train)
#Using Model Calibration
svm_events_sig_clf = CalibratedClassifierCV(svm_events_clf, method="sigmoid")
svm_events_sig_clf.fit(X_events_train, Y_events_train)
svm_events_predict_y = svm_events_sig_clf.predict_proba(X_events_cv)
print('For values of alpha = ', c, "The log loss is:",log_loss(Y_events_cv, svm_e
```

```
For values of alpha =  10 The log loss is: 2.2293116608139094
```

In [239]:
```
svm_events_pred_train = svm_events_sig_clf.predict_proba(X_events_train)
print("Train Log-Loss is",log_loss(Y_events_train, svm_events_pred_train, labels=
svm_events_pred_cv = svm_events_sig_clf.predict_proba(X_events_cv)
print("CV Log-Loss is",log_loss(Y_events_cv, svm_events_pred_cv, labels=svm_event
```

```
Train Log-Loss is 2.1680112482745835
CV Log-Loss is 2.2293116608139094
```

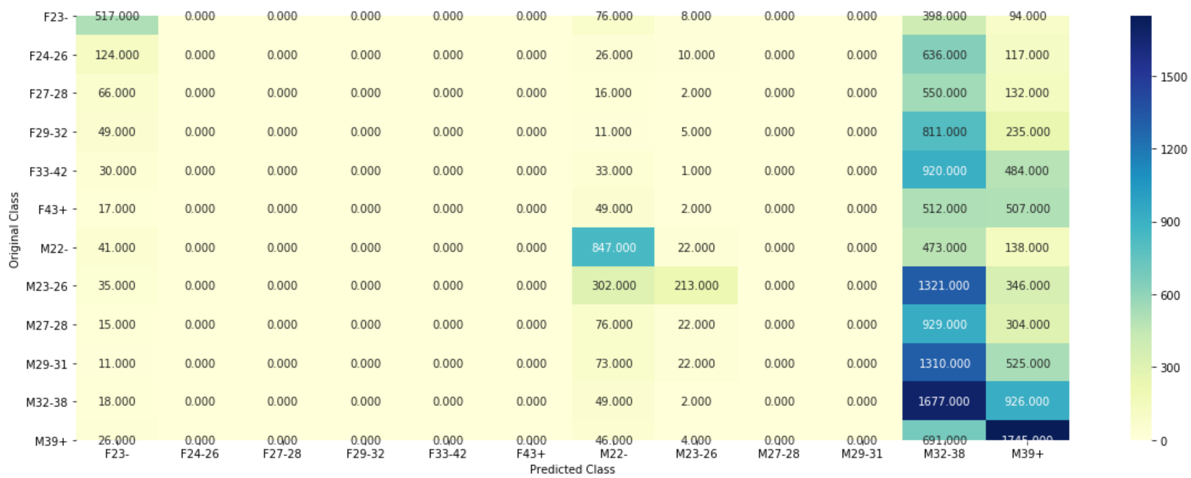
In [240]:
```
y_pred_events_train_classes_svm = np.argmax(svm_events_pred_train, axis=1)
y_pred_events_cv_classes_svm = np.argmax(svm_events_pred_cv, axis=1)
```

In [241]:
```
print("Train Log Loss :",log_loss(Y_events_train, svm_events_pred_train))
plot_confusion_matrix(Y_events_train,y_pred_events_train_classes_svm)

print("="*60)

print("CV Log Loss :",log_loss(Y_events_cv, svm_events_pred_cv))
plot_confusion_matrix(Y_events_cv,y_pred_events_cv_classes_svm)
```

```
Train Log Loss : 2.1680112482745835
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [302]:
```
y_events_pred_svm = svm_events_sig_clf.predict_proba(X_events_test)

# Multilabel class probalities are predicted
prediction_events_svm = pd.DataFrame(y_events_pred_svm,index = having_events_data
prediction_events_svm.head()
prediction_events_svm.to_csv("Predictions/prediction_events_svm.csv")
```

5.2.3 Logistic Regression

In [128]:
```
# #Loading the model
# log_reg_events_sig_clf = joblib.load('Models/log_reg_events/events_logistic_reg
```

In [424]:
```
erparameters = [10**-4, 10**-3,10**-2,10**-1,10**0, 10**1, 10**2,10**3,10**4]
log_error_array = []
 c in hyperparameters:
 log_reg_events_clf = LogisticRegression(C=c, class_weight='balanced', multi_clas
 log_reg_events_clf.fit(X_events_train, Y_events_train)
 #Using Model Calibration
 log_reg_events_sig_clf = CalibratedClassifierCV(log_reg_events_clf, method="sigm
 log_reg_events_sig_clf.fit(X_events_train, Y_events_train)
 log_reg_events_predict_y = log_reg_events_sig_clf.predict_proba(X_events_cv)
 print('For values of C = ', c, "The log loss is:",log_loss(Y_events_cv, log_reg_
```

```
For values of C =  0.0001 The log loss is: 2.195634357544072
For values of C =  0.001 The log loss is: 2.0992408439935297
For values of C =  0.01 The log loss is: 2.021074377675445
For values of C =  0.1 The log loss is: 2.0407353146729488
For values of C =  1 The log loss is: 2.1092748831094887
For values of C =  10 The log loss is: 2.126140261538226
For values of C =  100 The log loss is: 2.1273864200405233
For values of C =  1000 The log loss is: 2.1348130714885483
For values of C =  10000 The log loss is: 2.1349921423751335
```

In [124]:
```
#Training Model With best Hyperparameters
best_c = 10**-2
log_reg_events_clf = LogisticRegression(C=best_c, class_weight='balanced', multi_
log_reg_events_clf.fit(X_events_train, Y_events_train)
log_reg_events_sig_clf = CalibratedClassifierCV(log_reg_events_clf, method="sigm
log_reg_events_sig_clf.fit(X_events_train, Y_events_train)
```

Out[124]:
```
CalibratedClassifierCV(base_estimator=LogisticRegression(C=0.01,
                                                         class_weight='balance
d',
                                                         dual=False,
                                                         fit_intercept=True,
                                                         intercept_scaling=1,
                                                         l1_ratio=None,
                                                         max_iter=100,
                                                         multi_class='multinomi
al',
                                                         n_jobs=None,
                                                         penalty='l2',
                                                         random_state=42,
                                                         solver='lbfgs',
                                                         tol=0.0001, verbose=0,
                                                         warm_start=False),
                       cv='warn', method='sigmoid')
```

In [127]:
```
# #Saving the model
# jobl.dump(log_reg_events_sig_clf,'Models/log_reg_events/events_logistic_regres
```

In [129]:
```
#Printing log loss
log_reg_events_pred_train = log_reg_events_sig_clf.predict_proba(X_events_train)
print("Train Log-Loss is",log_loss(Y_events_train, log_reg_events_pred_train, la
log_reg_events_pred_cv = log_reg_events_sig_clf.predict_proba(X_events_cv)
print("CV Log-Loss is",log_loss(Y_events_cv, log_reg_events_pred_cv, labels=log_
```

```
Train Log-Loss is 1.88945546600425
CV Log-Loss is 2.021074377675445
```

In [130]:
```
y_pred_events_train_classes_log_reg = np.argmax(log_reg_events_pred_train, axis=
y_pred_events_cv_classes_log_reg = np.argmax(log_reg_events_pred_cv, axis=1)
```

In [131]:
```
#Printing confusion matrix
print("Train Log Loss :",log_loss(Y_events_train, log_reg_events_pred_train))
plot_confusion_matrix(Y_events_train,y_pred_events_train_classes_log_reg)

print("="*60)

print("CV Log Loss :",log_loss(Y_events_cv, log_reg_events_pred_cv))
plot_confusion_matrix(Y_events_cv,y_pred_events_cv_classes_log_reg)
```

```
Train Log Loss : 1.88945546600425
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [132]:
```
#Predicting Class Labels probalities for test data
y_events_pred_logistic_regression = log_reg_events_sig_clf.predict_proba(X_event
prediction_events_log_reg = pd.DataFrame(y_events_pred_logistic_regression,index
prediction_events_log_reg.to_csv("Predictions/prediction_events_log_reg.csv")
```

5.2.4. XGBoost

In [431]:
```
# https://www.kaggle.com/c/talkingdata-mobile-user-demographics/discussion/23424
# Best max_depth will be 5(Got from the above link discussion)
max_depth= [2,3,4,5]
n_estimator = [250,350,450]
for depth in max_depth:
    for estimator in n_estimator:
        xgb_events_clf=XGBClassifier(n_estimators=estimator,n_jobs=-1,learning_ra
        xgb_events_clf.fit(X_events_train, Y_events_train)
        xgb_events_sig_clf = CalibratedClassifierCV(xgb_events_clf)
        xgb_events_sig_clf.fit(X_events_train, Y_events_train)
        predict_events_xgb_y = xgb_events_sig_clf.predict_proba(X_events_cv)
        print ('log_loss for n_estimator = ',estimator,'and maxdepth =',depth,'i
```

In [252]:
```
xgb_events_clf=XGBClassifier(n_estimators=350,n_jobs=-1,learning_rate=0.05, cols
xgb_events_clf.fit(X_events_train, Y_events_train)
xgb_events_sig_clf = CalibratedClassifierCV(xgb_events_clf)
xgb_events_sig_clf.fit(X_events_train, Y_events_train)
xgb_events_predict_y = xgb_events_sig_clf.predict_proba(X_events_cv)
print ('log_loss for n_estimator = ',350,'and maxdepth =',4,'is',log_loss(Y_even
```

```
log_loss for n_estimator =  350 and maxdepth = 4 is 2.065546582627574
```

In [253]:
```
xgb_events_pred_train = xgb_events_sig_clf.predict_proba(X_events_train)
print("Train Log-Loss is",log_loss(Y_events_train, xgb_events_pred_train, labels
xgb_events_pred_cv = xgb_events_sig_clf.predict_proba(X_events_cv)
print("CV Log-Loss is",log_loss(Y_events_cv, xgb_events_pred_cv, labels=xgb_even
```

```
Train Log-Loss is 1.4814399080847296
CV Log-Loss is 2.065546582627574
```

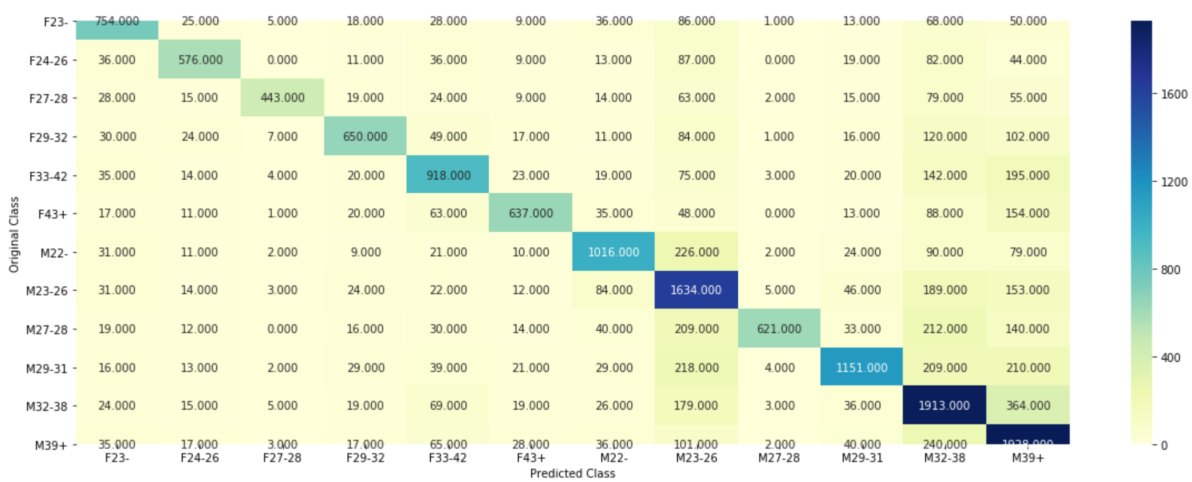
In [254]:
```
y_pred_events_train_classes_xgb = np.argmax(xgb_events_pred_train, axis=1)
y_pred_events_cv_classes_xgb = np.argmax(xgb_events_pred_cv, axis=1)
```

In [255]:
```
print("Train Log Loss :",log_loss(Y_events_train, xgb_events_pred_train))
plot_confusion_matrix(Y_events_train,y_pred_events_train_classes_xgb)

print("="*60)

print("CV Log Loss :",log_loss(Y_events_cv, xgb_events_pred_cv))
plot_confusion_matrix(Y_events_cv,y_pred_events_cv_classes_xgb)
```

```
Train Log Loss : 1.4814399080847296
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [308]:
```
y_events_pred_xgboost = xgb_events_sig_clf.predict_proba(X_events_test)

# Multilabel class probalities are predicted
prediction_events_xgb = pd.DataFrame(y_events_pred_xgboost,index = having_events_
prediction_events_xgb.head()
prediction_events_xgb.to_csv("Predictions/prediction_events_xgb.csv")
```

5.2.5. Random Forest

In [258]:
```
from sklearn.ensemble import RandomForestClassifier
max_depth = [3, 4, 5, 6]
n_estimator = [350,450,550]
for depth in max_depth:
    for estimator in n_estimator:
        rf_events_clf=RandomForestClassifier(n_estimators=estimator,n_jobs=-1, ma
        rf_events_clf.fit(X_events_train, Y_events_train)
        rf_events_sig_clf = CalibratedClassifierCV(rf_events_clf, method="sigmoi
        rf_events_sig_clf.fit(X_events_train, Y_events_train)
        rf_events_predict_y = rf_events_sig_clf.predict_proba(X_events_cv)
        print ('log_loss for n_estimator = ',estimator,'and maxdepth =',depth,'i
```

```
log_loss for n_estimator =  350 and maxdepth = 3 is 2.171986027538312
log_loss for n_estimator =  450 and maxdepth = 3 is 2.1707237438136486
log_loss for n_estimator =  550 and maxdepth = 3 is 2.1666960153128603
log_loss for n_estimator =  350 and maxdepth = 4 is 2.1587664328560816
log_loss for n_estimator =  450 and maxdepth = 4 is 2.159469938622786
log_loss for n_estimator =  550 and maxdepth = 4 is 2.158241615093273
log_loss for n_estimator =  350 and maxdepth = 5 is 2.1499067761911177
log_loss for n_estimator =  450 and maxdepth = 5 is 2.1517681616612356
log_loss for n_estimator =  550 and maxdepth = 5 is 2.150181618268085
log_loss for n_estimator =  350 and maxdepth = 6 is 2.1440397949344283
log_loss for n_estimator =  450 and maxdepth = 6 is 2.144281228180726
log_loss for n_estimator =  550 and maxdepth = 6 is 2.1448643367241362
```

In [259]:
```
rf_events_clf=RandomForestClassifier(n_estimators=350,n_jobs=-1, max_depth=5)
rf_events_clf.fit(X_events_train, Y_events_train)
rf_events_sig_clf = CalibratedClassifierCV(rf_events_clf, method="sigmoid")
rf_events_sig_clf.fit(X_events_train, Y_events_train)
rf_events_predict_y = rf_events_sig_clf.predict_proba(X_events_cv)
print ('log_loss for n_estimator = ',estimator,'and maxdepth =',depth,'is',log_lo
```

```
log_loss for n_estimator =  550 and maxdepth = 6 is 2.1516574269127413
```

In [260]:
```
rf_events_pred_train = rf_events_sig_clf.predict_proba(X_events_train)
print("Train Log-Loss is",log_loss(Y_events_train, rf_events_pred_train, labels=r
rf_events_pred_cv = rf_events_sig_clf.predict_proba(X_events_cv)
print("CV Log-Loss is",log_loss(Y_events_cv, rf_events_pred_cv, labels=rf_events_
```

```
Train Log-Loss is 2.065628925771852
CV Log-Loss is 2.1516574269127404
```

In [261]:
```
y_pred_events_train_classes_rf = np.argmax(rf_events_pred_train, axis=1)
y_pred_events_cv_classes_rf = np.argmax(rf_events_pred_cv, axis=1)
```

In [262]:
```
print("Train Log Loss :",log_loss(Y_events_train, rf_events_pred_train))
plot_confusion_matrix(Y_events_train,y_pred_events_train_classes_rf)

print("="*60)

print("CV Log Loss :",log_loss(Y_events_cv, rf_events_pred_cv))
plot_confusion_matrix(Y_events_cv,y_pred_events_cv_classes_rf)
```

```
Train Log Loss : 2.065628925771852
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [309]:
```
y_events_pred_random_forest = rf_events_sig_clf.predict_proba(X_events_test)
# Multilabel class probalities are predicted
prediction_events_rf = pd.DataFrame(y_events_pred_random_forest,index = having_ev
prediction_events_rf.head()
prediction_events_rf.to_csv("Predictions/prediction_events_rf.csv")
```

5.2.6. Decision tree

In [264]:
```
from sklearn.tree import DecisionTreeClassifier
max_depth = [3, 4, 5, 6]
min_samples_split_values = [5,10,100,500]
for depth in max_depth:
    for min_sample in min_samples_split_values:
        dt_events_clf=DecisionTreeClassifier(min_samples_split=min_sample,max_de
        dt_events_clf.fit(X_events_train, Y_events_train)
        dt_events_sig_clf = CalibratedClassifierCV(dt_events_clf, method="sigmoi
        dt_events_sig_clf.fit(X_events_train, Y_events_train)
        dt_events_predict_y = dt_events_sig_clf.predict_proba(X_events_cv)
        print ('log_loss for min_sample_split = ',min_sample,'and maxdepth =',de
```

```
log_loss for min_sample_split =  5 and maxdepth = 3 is 2.315839347233516
log_loss for min_sample_split =  10 and maxdepth = 3 is 2.315839347233516
log_loss for min_sample_split =  100 and maxdepth = 3 is 2.315839347233516
log_loss for min_sample_split =  500 and maxdepth = 3 is 2.31707491638645
log_loss for min_sample_split =  5 and maxdepth = 4 is 2.30180234699013
log_loss for min_sample_split =  10 and maxdepth = 4 is 2.3018589548413813
log_loss for min_sample_split =  100 and maxdepth = 4 is 2.3013630958317686
log_loss for min_sample_split =  500 and maxdepth = 4 is 2.3020708580201035
log_loss for min_sample_split =  5 and maxdepth = 5 is 2.290405248033235
log_loss for min_sample_split =  10 and maxdepth = 5 is 2.2899796601859848
log_loss for min_sample_split =  100 and maxdepth = 5 is 2.2857231291819793
log_loss for min_sample_split =  500 and maxdepth = 5 is 2.286207462685353
log_loss for min_sample_split =  5 and maxdepth = 6 is 2.2902936563875276
log_loss for min_sample_split =  10 and maxdepth = 6 is 2.289661641627657
log_loss for min_sample_split =  100 and maxdepth = 6 is 2.2803035978749784
log_loss for min_sample_split =  500 and maxdepth = 6 is 2.276983724237959
```

In [265]:
```
dt_events_clf=DecisionTreeClassifier(min_samples_split=100,max_depth=5,class_wei
dt_events_clf.fit(X_events_train, Y_events_train)
dt_events_sig_clf = CalibratedClassifierCV(dt_events_clf, method="sigmoid")
dt_events_sig_clf.fit(X_events_train, Y_events_train)
dt_events_predict_y = dt_events_sig_clf.predict_proba(X_events_cv)
print ('log_loss for min_sample_split = ',100,'and maxdepth =',5,'is',log_loss(Y
```

```
log_loss for min_sample_split =  100 and maxdepth = 5 is 2.2857231291819793
```

In [266]:
```
dt_events_pred_train = dt_events_sig_clf.predict_proba(X_events_train)
print("Train Log-Loss is",log_loss(Y_events_train, dt_events_pred_train, labels=
dt_events_pred_cv = dt_events_sig_clf.predict_proba(X_events_cv)
print("CV Log-Loss is",log_loss(Y_events_cv, dt_events_pred_cv, labels=dt_events
```

```
Train Log-Loss is 2.2673973721615237
CV Log-Loss is 2.2857231291819793
```

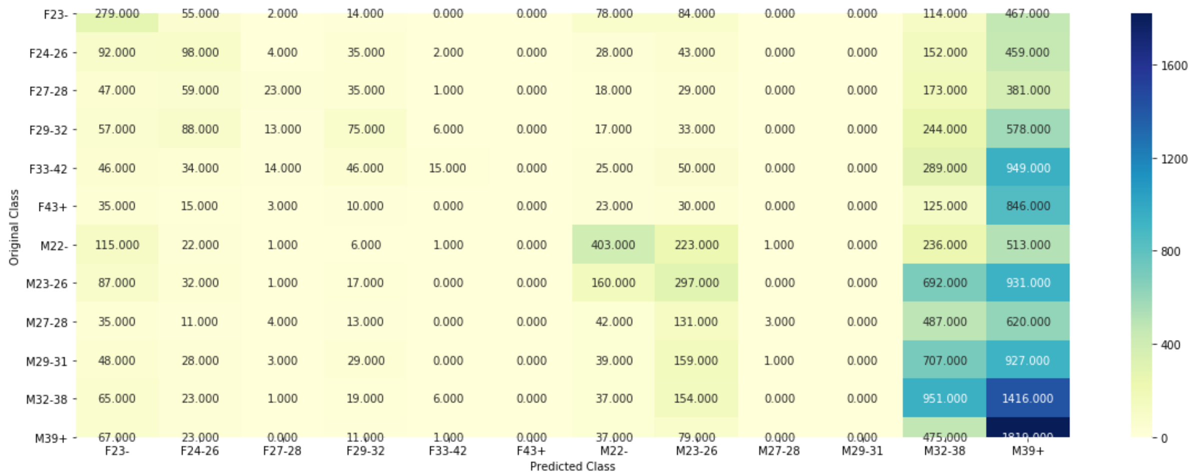
In [267]:
```
y_pred_events_train_classes_dt = np.argmax(dt_events_pred_train, axis=1)
y_pred_events_cv_classes_dt = np.argmax(dt_events_pred_cv, axis=1)
```

In [268]:
```
print("Train Log Loss :",log_loss(Y_events_train, dt_events_pred_train))
plot_confusion_matrix(Y_events_train,y_pred_events_train_classes_dt)

print("="*60)

print("CV Log Loss :",log_loss(Y_events_cv, dt_events_pred_cv))
plot_confusion_matrix(Y_events_cv,y_pred_events_cv_classes_dt)
```

```
Train Log Loss : 2.2673973721615237
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

In [271]:
```
y_events_pred_decision_tree = dt_events_sig_clf.predict_proba(X_events_test)
```

In [310]:
```
# Multilabel class probalities are predicted
prediction_events_dt = pd.DataFrame(y_events_pred_decision_tree,index = having_ev
prediction_events_dt.head()
prediction_events_dt.to_csv("Predictions/prediction_events_dt.csv")
```

5.2.7. Deep Learning - Neural Networks

Neural Network 1

Here the train data is trained from 20 times and validated against the cv data. Then the average of all the predicted class labels are taken.

In [133]:
```
# #loading models
# model_2 = []
# for i in range(20):
#     model_2.append(load_model('Models/events_1/events_nn_1_'+str(i+1)+'.h5'))
```

```
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout()
uses dropout rate instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout()
uses dropout rate instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout()
uses dropout rate instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout()
uses dropout rate instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout()
uses dropout rate instead of keep_prob. Please ensure that this is intended.
```

In [134]:
```
#Coverting to categorical for NN
y_train_events=np_utils.to_categorical(Y_events_train)
y_cv_events=np_utils.to_categorical(Y_events_cv)
```

In [525]:
```
#https://www.kaggle.com/c/talkingdata-mobile-user-demographics/discussion/23424
#Refered From above kaggle dicussion
def model_events_nn_1(input_dim,output_dim):
    model = Sequential()
    model.add(Dropout(0.15, input_shape=(input_dim,)))
    model.add(Dense(240, init='uniform'))
    model.add(PReLU(init='zero'))
    model.add(Dropout(0.8))
    model.add(Dense(240, init='uniform'))
    model.add(PReLU(init='zero', weights=None))
    model.add(Dropout(0.35))
    model.add(Dense(260, init='uniform'))
    model.add(PReLU(init='zero', weights=None))
    model.add(Dropout(0.40))
    model.add(Dense(output_dim, init='uniform'))
    model.add(Activation('softmax'))

    opt = Adagrad(lr=0.008, epsilon=1e-08)
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
    return model
```

In [526]:
```
model_events_1=model_events_nn_1(X_events_train.shape[1],12)
model_events_1.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dropout_19 (Dropout)         (None, 21566)             0         
_________________________________________________________________
dense_28 (Dense)             (None, 240)               5176080   
_________________________________________________________________
p_re_lu_19 (PReLU)           (None, 240)               240       
_________________________________________________________________
dropout_20 (Dropout)         (None, 240)               0         
_________________________________________________________________
dense_29 (Dense)             (None, 240)               57840     
_________________________________________________________________
p_re_lu_20 (PReLU)           (None, 240)               240       
_________________________________________________________________
dropout_21 (Dropout)         (None, 240)               0         
_________________________________________________________________
dense_30 (Dense)             (None, 260)               62660     
_________________________________________________________________
p_re_lu_21 (PReLU)           (None, 260)               260       
_________________________________________________________________
dropout_22 (Dropout)         (None, 260)               0         
_________________________________________________________________
dense_31 (Dense)             (None, 12)                3132      
_________________________________________________________________
activation_10 (Activation)   (None, 12)                0         
=================================================================
Total params: 5,300,452
Trainable params: 5,300,452
Non-trainable params: 0
_________________________________________________________________
```

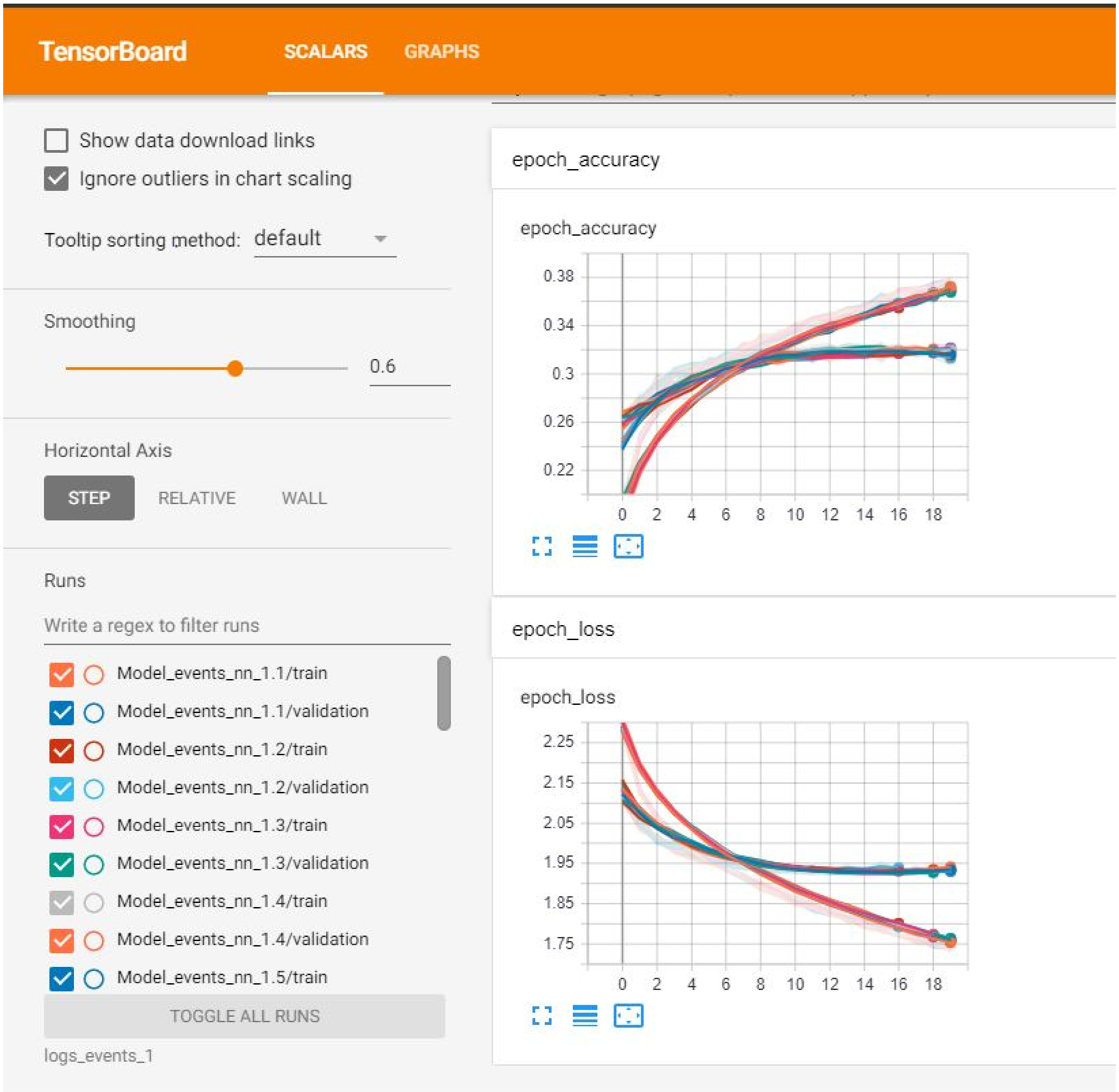
In [530]:
```
model_2=[]
avg_cv_loss=0
for i in range(20):
    model=model_events_nn_1(X_events_train.shape[1],12)
    logdir = os.path.join("logs_events_1","Model_events_1."+str(i+1))
    t_callback=TensorBoard(log_dir=logdir)
    model.fit(X_events_train, y_train_events, batch_size=149, epochs=20, verbose
    model_cv_prediction=model.predict_proba(X_events_cv)
    cv_loss=log_loss(Y_events_cv, model_cv_prediction)
    print("CV Log Loss of Best Weights Model in Current Run: ",cv_loss)
    model_2.append(model)
    avg_cv_loss+=cv_loss
avg_cv_loss/=n_models
print("Average CV Loss of "+str(n_models)+" Runs :",avg_cv_loss)
return model_2
```

```
Train on 18647 samples, validate on 4662 samples
Epoch 1/20
18647/18647 [==============================] - ETA: 41s - loss: 2.4761 - accu
racy: 0.181 - ETA: 28s - loss: 2.4803 - accuracy: 0.164 - ETA: 25s - loss: 2.
4717 - accuracy: 0.143 - ETA: 22s - loss: 2.4638 - accuracy: 0.146 - ETA: 21s
- loss: 2.4573 - accuracy: 0.146 - ETA: 19s - loss: 2.4566 - accuracy: 0.143
- ETA: 19s - loss: 2.4467 - accuracy: 0.148 - ETA: 18s - loss: 2.4363 - accur
acy: 0.148 - ETA: 17s - loss: 2.4400 - accuracy: 0.146 - ETA: 17s - loss: 2.4
381 - accuracy: 0.145 - ETA: 17s - loss: 2.4319 - accuracy: 0.144 - ETA: 16s
- loss: 2.4302 - accuracy: 0.142 - ETA: 16s - loss: 2.4210 - accuracy: 0.142
- ETA: 16s - loss: 2.4231 - accuracy: 0.143 - ETA: 16s - loss: 2.4224 - accur
acy: 0.143 - ETA: 15s - loss: 2.4185 - accuracy: 0.144 - ETA: 15s - loss: 2.4
175 - accuracy: 0.142 - ETA: 15s - loss: 2.4151 - accuracy: 0.142 - ETA: 15s
- loss: 2.4124 - accuracy: 0.143 - ETA: 14s - loss: 2.4129 - accuracy: 0.141
- ETA: 14s - loss: 2.4112 - accuracy: 0.142 - ETA: 14s - loss: 2.4122 - accur
acy: 0.142 - ETA: 14s - loss: 2.4107 - accuracy: 0.140 - ETA: 14s - loss: 2.4
107 - accuracy: 0.139 - ETA: 13s - loss: 2.4093 - accuracy: 0.139 - ETA: 13s
- loss: 2.4091 - accuracy: 0.139 - ETA: 13s - loss: 2.4093 - accuracy: 0.139
- ETA: 13s - loss: 2.4096 - accuracy: 0.139 - ETA: 13s - loss: 2.4073 - accur
```

In [ ]:
```
# #Saving Models
# for i in range(len(model_2)):
#     model_2[i].save('Models/events_1/events_nn_1_'+str(i+1)+'.h5')
```

```
In [182]: Image('Pred/tensorboard3.jpg')
```

Out[182]:



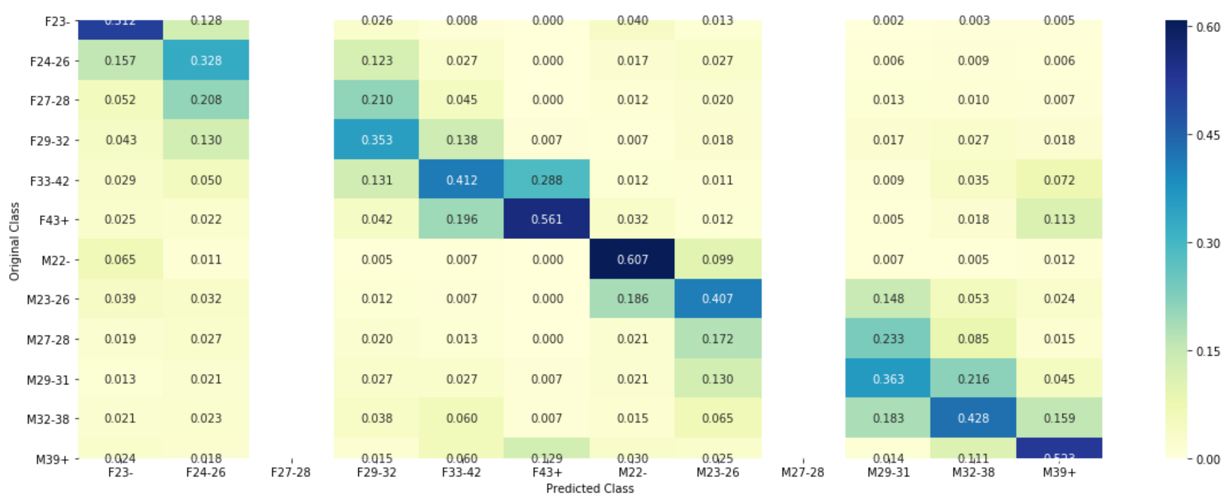
Train Prediction And Confusion matrix

In [135]:
```
train_pred_avg_events_1=np.zeros((X_events_train.shape[0],12))
#Taking average of all the model predictions
for i in range(len(model_2)):
    train_pred=model_2[i].predict_proba(X_events_train)
    train_pred_avg_events_1+=train_pred
train_pred_avg_events_1/=len(model_2)
```

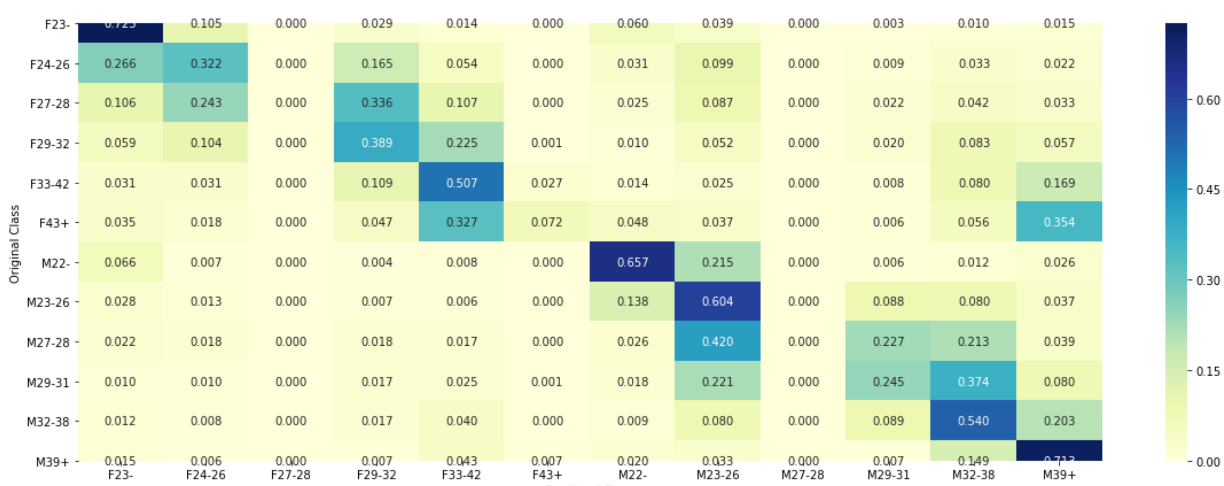
In [136]:
```
print("Train Average Log-Loss: ",log_loss(Y_events_train, train_pred_avg_events_1
events_nn_1_avg_train_predicted_classes=np.argmax(train_pred_avg_events_1, axis=1
plot_confusion_matrix(Y_events_train,events_nn_1_avg_train_predicted_classes)
```

```
Train Average Log-Loss:  1.5883830015371585
-------------------- Confusion matrix --------------------
```

```
-------------------- Precision matrix (Column Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```

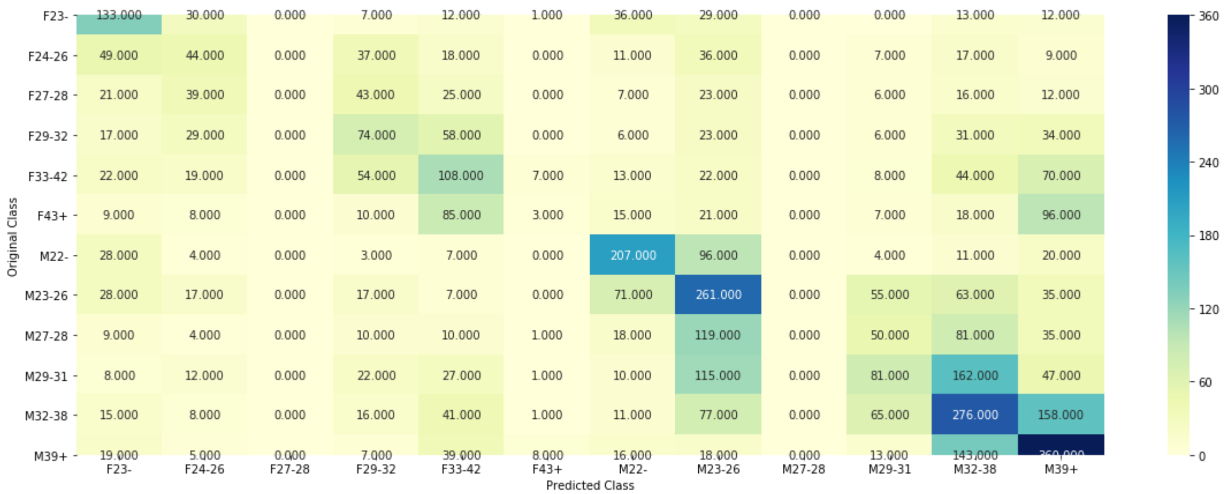


CV Prediction and Confusion matrix

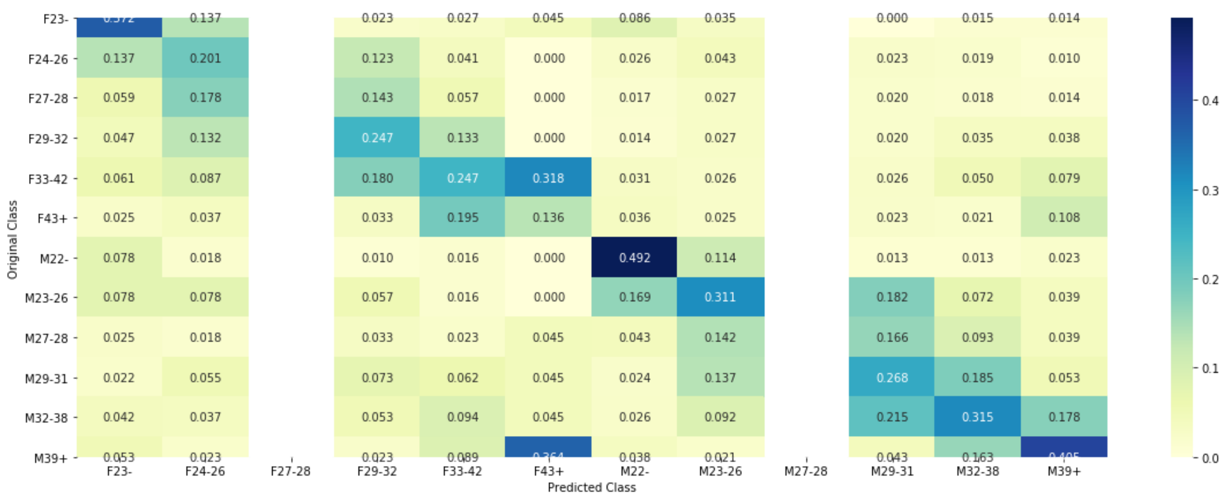
In [137]:
```
cv_pred_avg_events_1=np.zeros((X_events_cv.shape[0],12))
#Taking average of all the model predictions
for i in range(len(model_2)):
    cv_pred=model_2[i].predict_proba(X_events_cv)
    cv_pred_avg_events_1+=cv_pred
cv_pred_avg_events_1/=len(model_2)
```

In [138]:
```
print("CV Average Log-Loss: ",log_loss(Y_events_cv, cv_pred_avg_events_1))
events_nn_1_avg_cv_predicted_classes=np.argmax(cv_pred_avg_events_1, axis=1)
plot_confusion_matrix(Y_events_cv,events_nn_1_avg_cv_predicted_classes)
```

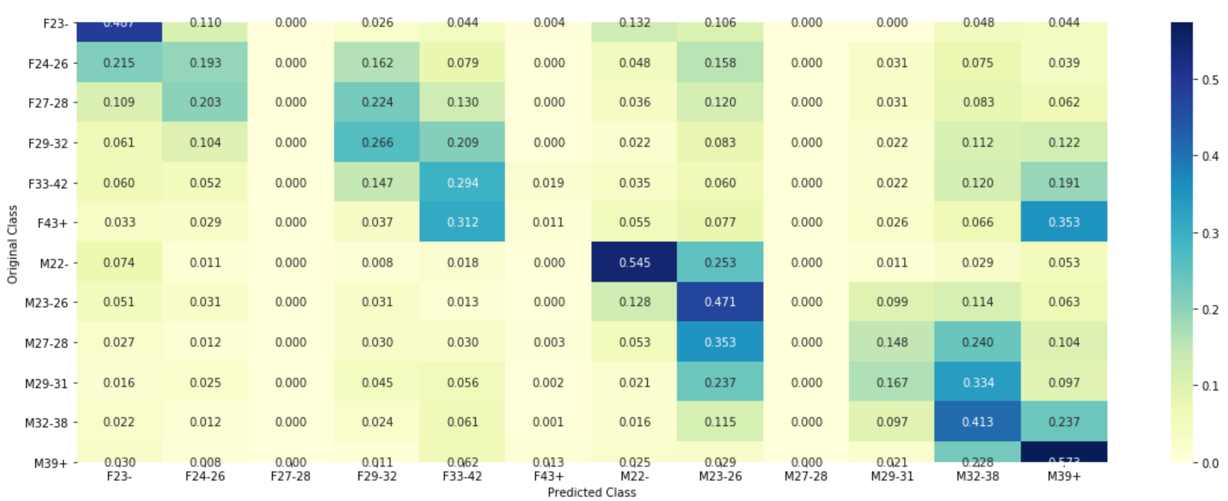
```
CV Average Log-Loss:  1.906501488636026
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Column Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Test Prediction

In [139]:
```
test_pred_avg_events_1=np.zeros((X_events_test.shape[0],12))
#Taking average of all the model predictions
for i in range(len(model_2)):
    test_pred=model_2[i].predict_proba(X_events_test.tocsr())
    test_pred_avg_events_1+=test_pred
test_pred_avg_events_1/=len(model_2)
```

Neural Network 2

Here the train data is trained from 20 times and validated against the cv data. Then the average of all the predicted class labels are taken.

In [140]:
```
# #Loading the Models
# model_3 = []
# for i in range(20):
#     model_3.append(load_model('Models/events_2/events_nn_2_'+str(i+1)+'.h5'))
```

In [531]:
```
#https://github.com/chechir/talking_data
#Refered from the above github link
def model_events_nn_2(input_dim,output_dim):
    model = Sequential()
    model.add(Dropout(0.4, input_shape=(input_dim,)))
    model.add(Dense(75))
    model.add(PReLU())
    model.add(Dropout(0.30))
    model.add(Dense(50, init='normal', activation='tanh'))
    model.add(PReLU())
    model.add(Dropout(0.20))
    model.add(Dense(output_dim, init='normal', activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics
    return model
```

In [532]:
```
model_events_2=model_events_nn_2(X_events_train.shape[1],12)
model_events_2.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dropout_35 (Dropout)         (None, 21566)             0         
_________________________________________________________________
dense_44 (Dense)             (None, 75)                1617525   
_________________________________________________________________
p_re_lu_31 (PReLU)           (None, 75)                75        
_________________________________________________________________
dropout_36 (Dropout)         (None, 75)                0         
_________________________________________________________________
dense_45 (Dense)             (None, 50)                3800      
_________________________________________________________________
p_re_lu_32 (PReLU)           (None, 50)                50        
_________________________________________________________________
dropout_37 (Dropout)         (None, 50)                0         
_________________________________________________________________
dense_46 (Dense)             (None, 12)                612       
=================================================================
Total params: 1,622,062
Trainable params: 1,622,062
Non-trainable params: 0
_________________________________________________________________
```

In [535]:
```
#Running the Model
model_3=[]
avg_cv_loss=0
for i in range(20):
    model=model_events_nn_2(X_events_train.shape[1],12)
    logdir = os.path.join("logs_events_2","Model_events_2."+str(i+1))
    t_callback=TensorBoard(log_dir=logdir)
    model.fit(X_events_train, y_train_events, batch_size=149, epochs=20, verbose
    model_cv_prediction=model.predict_proba(X_events_cv)
    cv_loss=log_loss(y_cv_events, model_cv_prediction)
    print("CV Log Loss of Best Weights Model in Current Run: ",cv_loss)
    model_3.append(model)
    avg_cv_loss+=cv_loss
avg_cv_loss/=n_models
print("Average CV Loss of "+str(n_models)+" Runs :",avg_cv_loss)
return model_3
```

```
Train on 18647 samples, validate on 4662 samples
Epoch 1/20
18647/18647 [==============================] - ETA: 41s - loss: 2.4844 - accu
racy: 0.087 - ETA: 27s - loss: 2.4837 - accuracy: 0.090 - ETA: 23s - loss: 2.
4834 - accuracy: 0.102 - ETA: 21s - loss: 2.4816 - accuracy: 0.117 - ETA: 19s
- loss: 2.4806 - accuracy: 0.114 - ETA: 18s - loss: 2.4798 - accuracy: 0.114
- ETA: 17s - loss: 2.4789 - accuracy: 0.115 - ETA: 16s - loss: 2.4777 - accur
acy: 0.115 - ETA: 16s - loss: 2.4743 - accuracy: 0.127 - ETA: 15s - loss: 2.4
706 - accuracy: 0.130 - ETA: 15s - loss: 2.4684 - accuracy: 0.131 - ETA: 15s
- loss: 2.4661 - accuracy: 0.133 - ETA: 14s - loss: 2.4620 - accuracy: 0.132
- ETA: 14s - loss: 2.4613 - accuracy: 0.133 - ETA: 14s - loss: 2.4565 - accur
acy: 0.135 - ETA: 13s - loss: 2.4553 - accuracy: 0.131 - ETA: 13s - loss: 2.4
551 - accuracy: 0.129 - ETA: 13s - loss: 2.4525 - accuracy: 0.130 - ETA: 13s
- loss: 2.4501 - accuracy: 0.132 - ETA: 12s - loss: 2.4480 - accuracy: 0.133
- ETA: 12s - loss: 2.4430 - accuracy: 0.134 - ETA: 12s - loss: 2.4402 - accur
acy: 0.135 - ETA: 12s - loss: 2.4393 - accuracy: 0.136 - ETA: 12s - loss: 2.4
374 - accuracy: 0.137 - ETA: 11s - loss: 2.4371 - accuracy: 0.137 - ETA: 11s
- loss: 2.4357 - accuracy: 0.137 - ETA: 11s - loss: 2.4331 - accuracy: 0.139
- ETA: 11s - loss: 2.4327 - accuracy: 0.138 - ETA: 11s - loss: 2.4305 - accur
```

In [ ]:
```
# #Saving Models
# for i in range(len(model_3)):
#     model_3[i].save('Models/events_2/events_nn_2_'+str(i+1)+'.h5')
```

In [ ]:
```
```

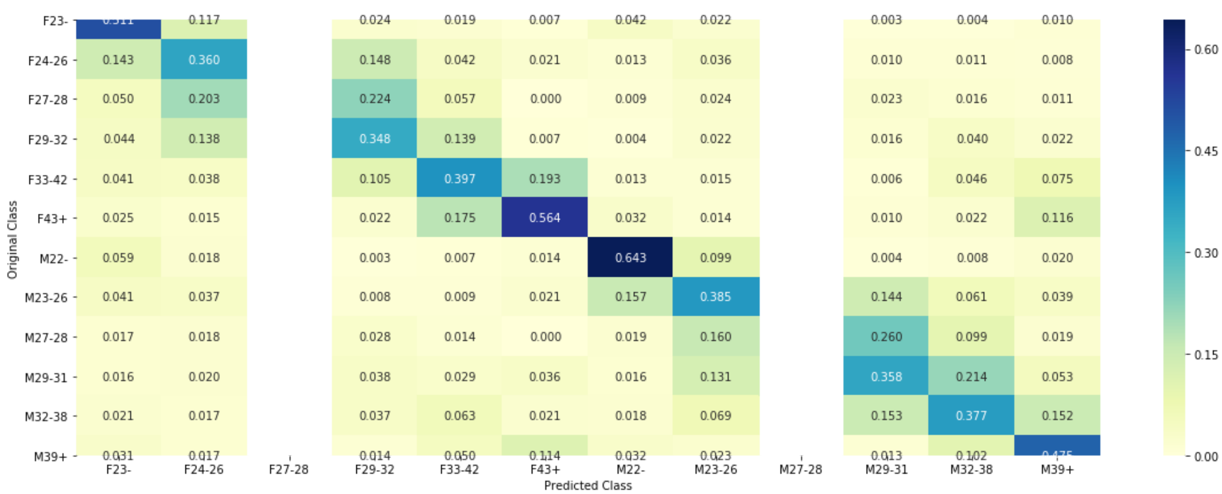
Train Prediction And Confusion Matrix

In [141]:
```
train_pred_avg_events_2=np.zeros((X_events_train.shape[0],12))
#Taking average of all the model predictions
for i in range(len(model_3)):
    train_pred=model_3[i].predict_proba(X_events_train)
    train_pred_avg_events_2+=train_pred
train_pred_avg_events_2/=len(model_3)
```

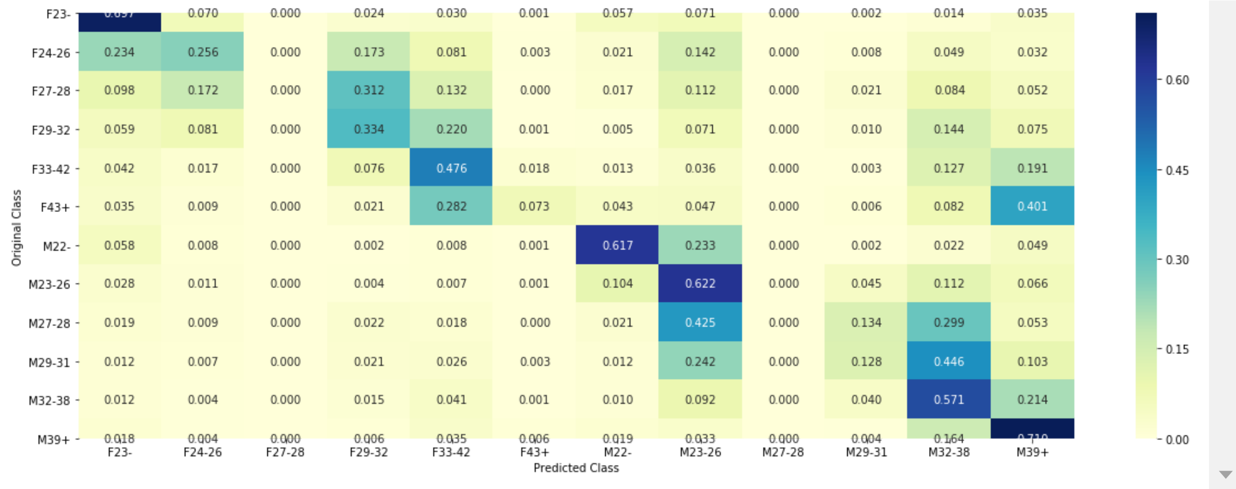
In [142]:
```
print("Train Average Log-Loss: ",log_loss(Y_events_train, train_pred_avg_events_
events_nn_2_avg_train_predicted_classes=np.argmax(train_pred_avg_events_2, axis=
plot_confusion_matrix(Y_events_train,events_nn_2_avg_train_predicted_classes)
```

```
Train Average Log-Loss:  1.6678485562072904
-------------------- Confusion matrix --------------------
```

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



CV Prediction and Confusion Matrix

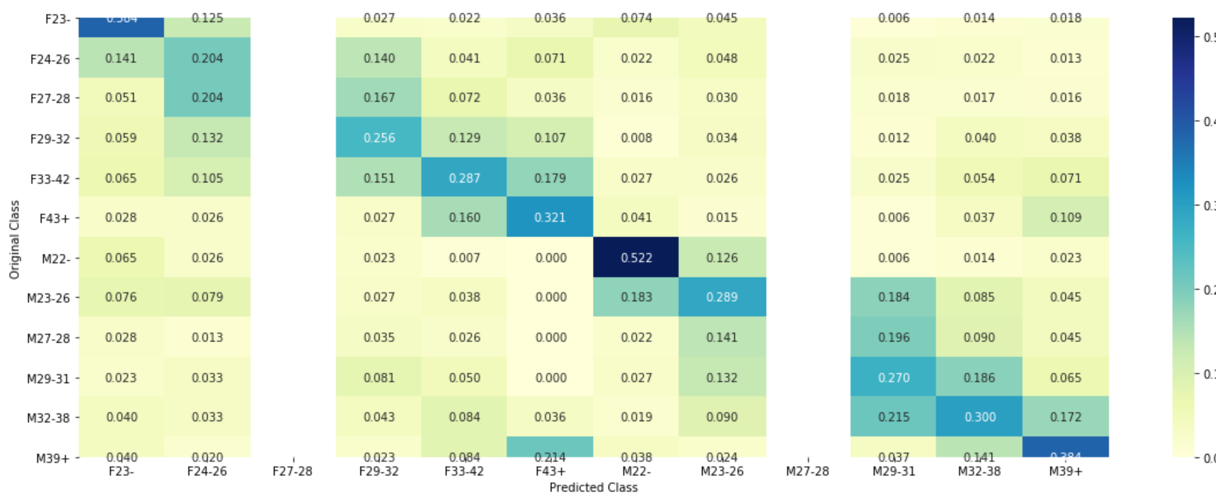
In [143]:
```
cv_pred_avg_events_2=np.zeros((X_events_cv.shape[0],12))
```

In [144]:
```
#taing average of all the model predictions
for i in range(len(model_3)):
    cv_pred=model_3[i].predict_proba(X_events_cv)
    cv_pred_avg_events_2+=cv_pred
cv_pred_avg_events_2/=len(model_3)
```

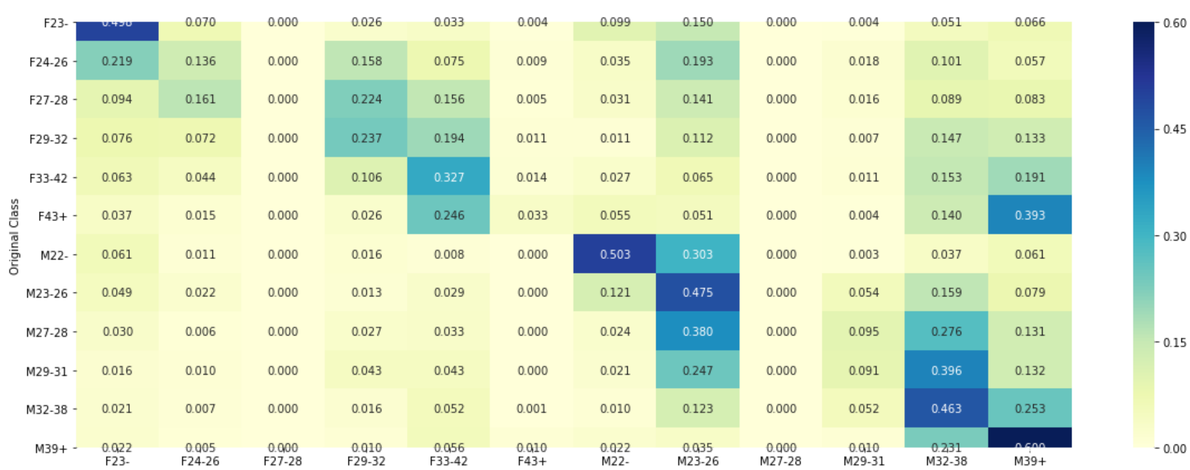
In [145]:
```
print("CV Average Log-Loss: ",log_loss(Y_events_cv, cv_pred_avg_events_2))
events_nn_2_avg_cv_predicted_classes=np.argmax(cv_pred_avg_events_2, axis=1)
plot_confusion_matrix(Y_events_cv,events_nn_2_avg_cv_predicted_classes)
```

```
CV Average Log-Loss:  1.8996915747356988
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Column Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Test Prediction

In [146]:
```
test_pred_avg_events_2=np.zeros((X_events_test.shape[0],12))
for i in range(len(model_3)):
    test_pred=model_3[i].predict_proba(X_events_test.tocsr())
    test_pred_avg_events_2+=test_pred
test_pred_avg_events_2/=len(model_3)
```

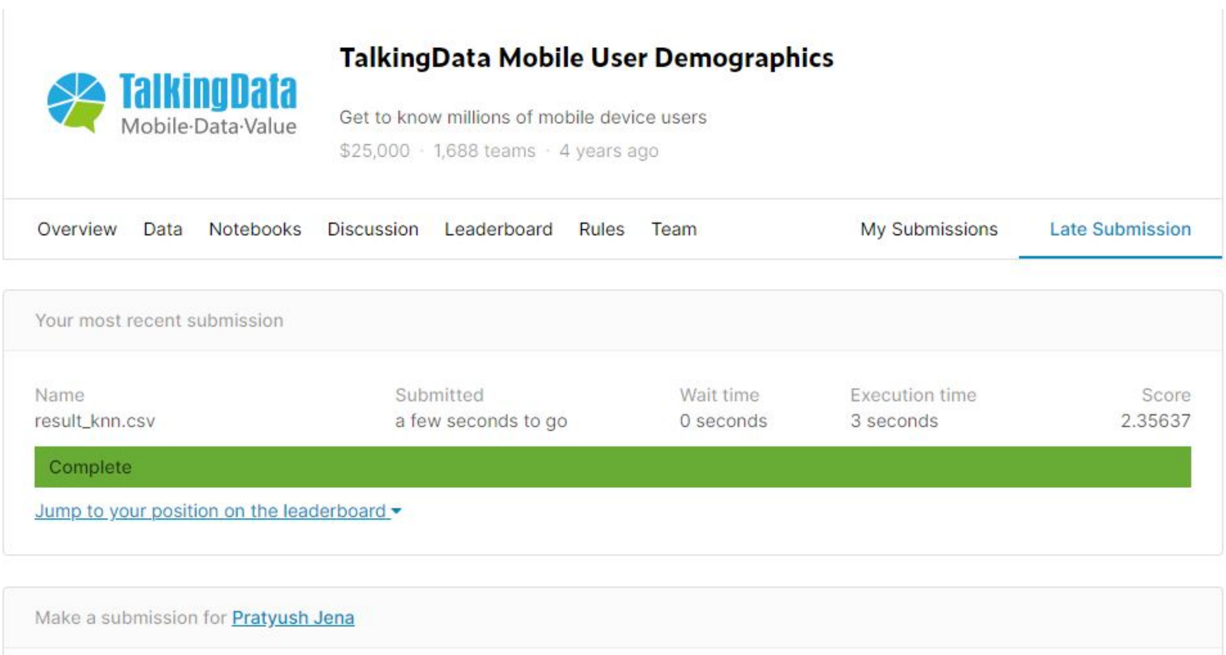
6. Predictions

6.1. KNN

In [311]:
```
predictions_knn = pd.concat([prediction_no_events_knn,prediction_events_knn],axi
predictions_knn.to_csv('Predictions/result_knn.csv')
```

In [315]:
```
Image('Pred/knn.jpg')
```

Out[315]:

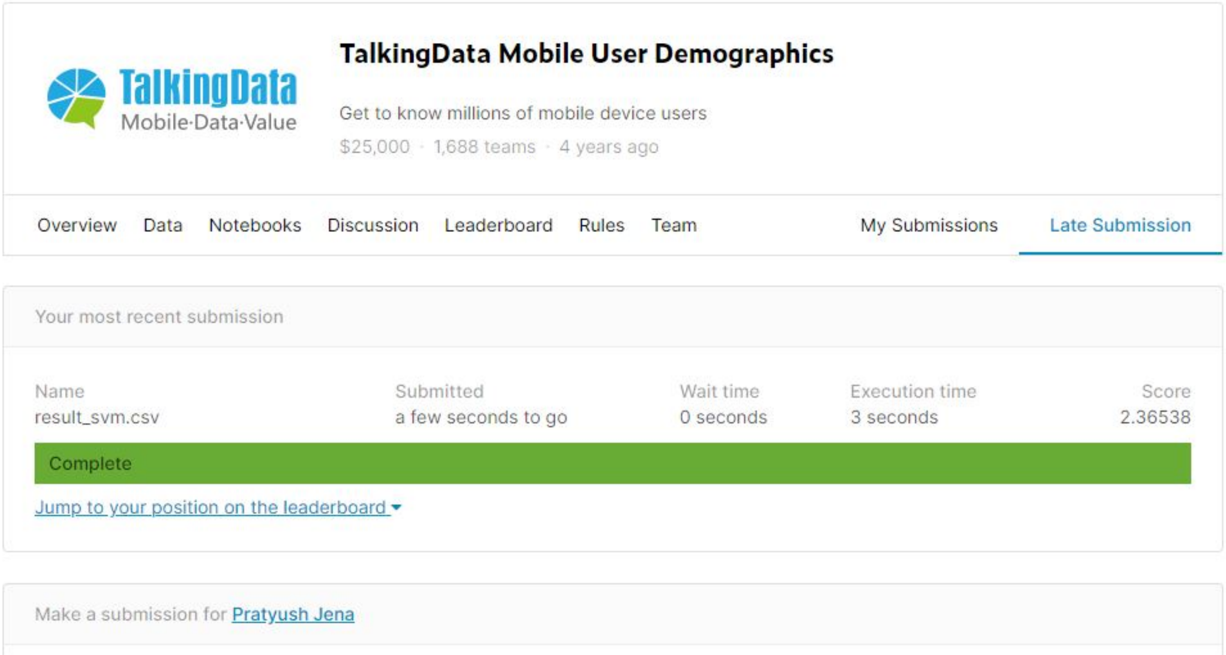


6.2. SVM

In [316]:
```
predictions_svm = pd.concat([prediction_no_events_svm,prediction_events_svm],axi
predictions_svm.to_csv('Predictions/result_svm.csv')
```

In [317]:
```
Image('Pred/svm.jpg')
```

Out[317]:

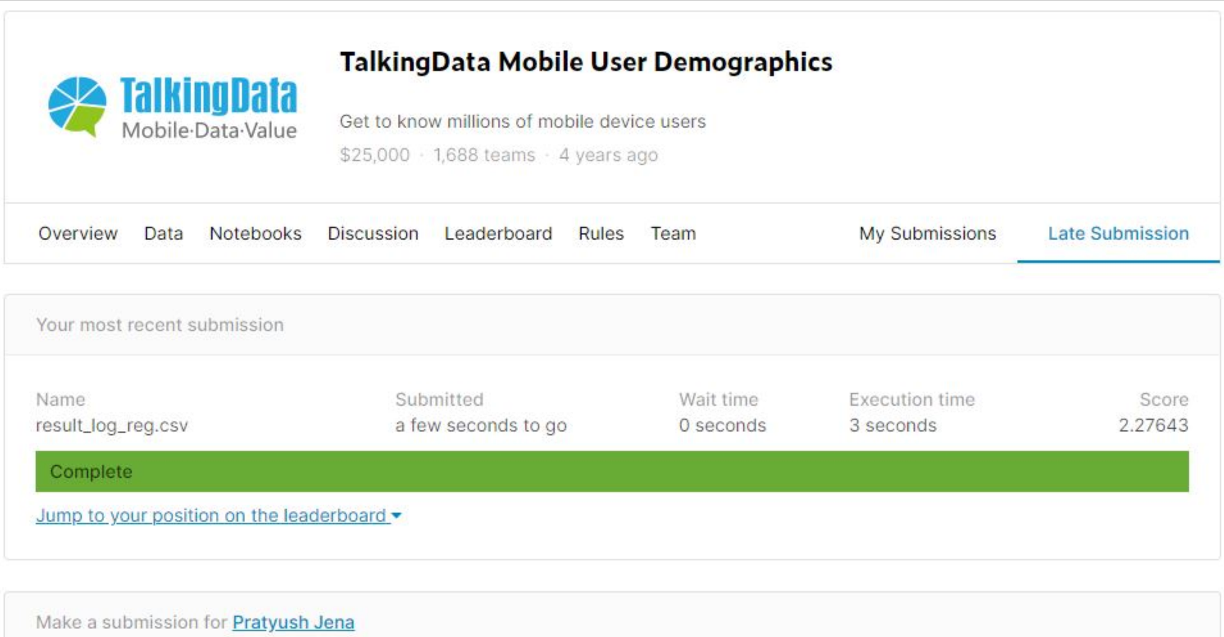


6.3. Logistic Regression

In [318]:
```
predictions_log_reg = pd.concat([prediction_no_events_log_reg,prediction_events_l
predictions_log_reg.to_csv('Predictions/result_log_reg.csv')
```

In [319]:
```
Image('Pred/log_reg.jpg')
```

Out[319]:

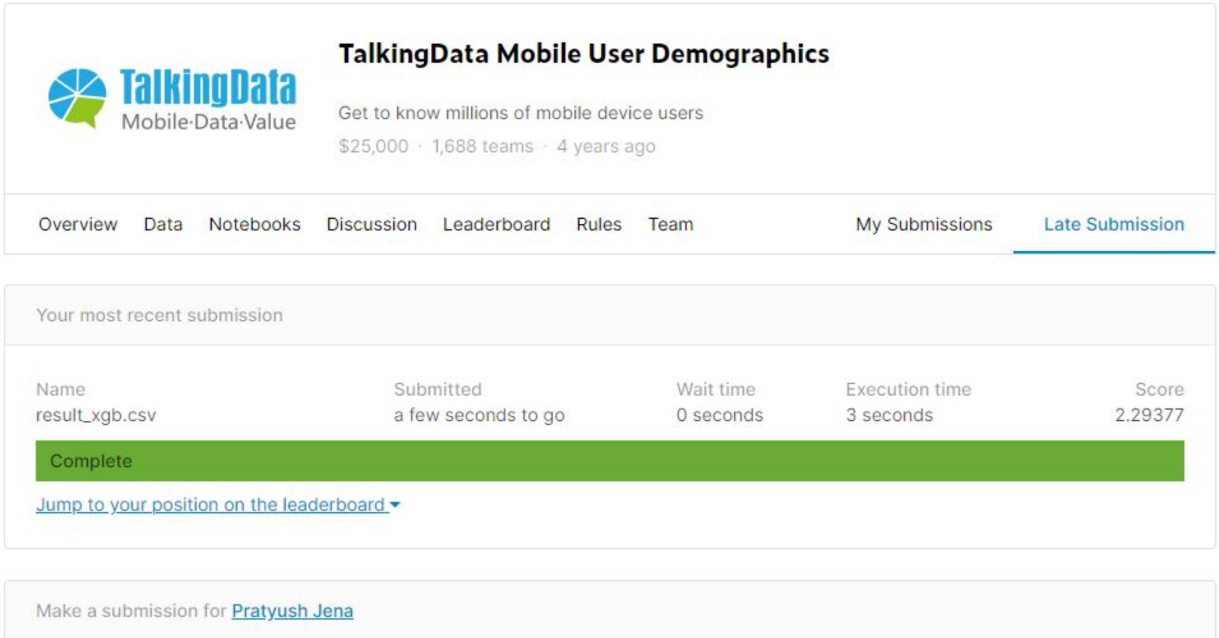


6.4. XGBoost

In [321]:
```
predictions_xgb = pd.concat([prediction_no_events_xgb,prediction_events_xgb],axi
predictions_xgb.to_csv('Predictions/result_xgb.csv')
```

In [322]:
```
Image('Pred/xgb.jpg')
```

Out[322]:

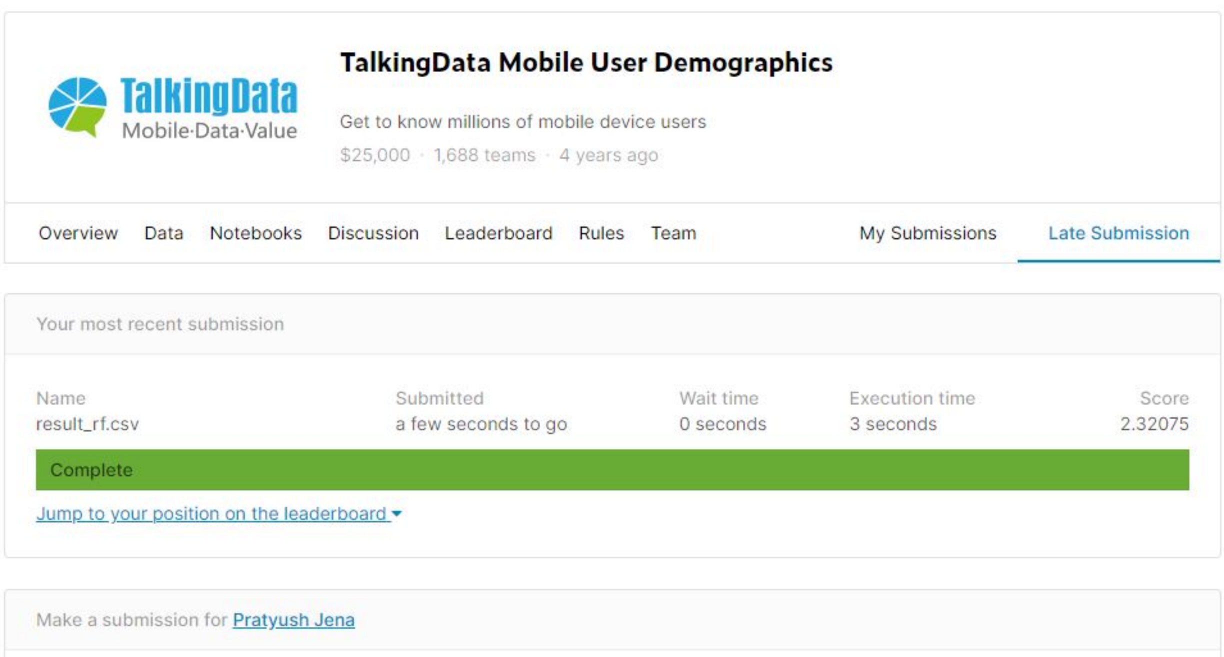


6.5. Random Forest

In [323]:
```
predictions_rf = pd.concat([prediction_no_events_rf,prediction_events_rf],axis=0
predictions_rf.to_csv('Predictions/result_rf.csv')
```

In [324]:
```
Image('Pred/rf.jpg')
```

Out[324]:

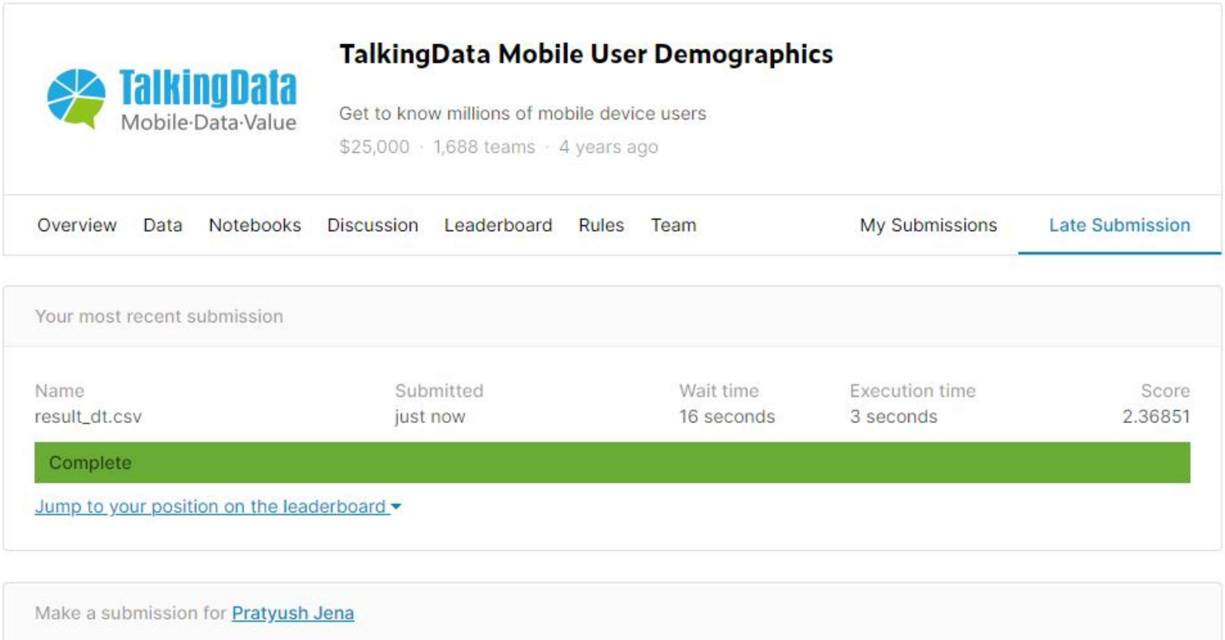


6.6. Decision Tree

In [325]:
```
predictions_rf = pd.concat([prediction_no_events_dt,prediction_events_dt],axis=0
predictions_rf.to_csv('Predictions/result_dt.csv')
```

In [327]:
```
Image('Pred/dt.jpg')
```

Out[327]:



6.7. Stacking Logistic regression with Neural Networks

From the above 6 model we can see that logistic regression gives the best result after predicting the age groups for test data with a log loss of 2.27643.

In the kaggle competition, there are total number of 1668 submissions. So in order to get a rank within first 10 percent we need to score a rank less than 168 and a log loss less than 2.24006 in private leadership board and a log loss less than 2.23592 in public leader ship board.

Private Leadership Board

In [164]:
```
Image('Pred/private.jpg')
```

Out[164]:

167	▾15	mlearn	2.24004	29	4y
168	—	Mandarin Orange	2.24006	11	4y
169	▴17	jeremie_db	2.24017	81	4y
170	▴15	Golden Bears	2.24019	94	4y
171	▴2	Arty Erokhin	2.24025	60	4y

Public Leadership Board

In [165]:
```
Image('Pred/public.jpg')
```

Out[165]:

167	Kesis	2.23579	17	4y
168	Mandarin Orange	2.23592	11	4y
169	Teléfono rojo	2.23596	84	4y
170	XiaolingChen	2.23608	10	4y
171	Nathaniel Shimoni	2.23616	19	4y

So to achieve a rank within 10 percent of the competition we will use stacking model.
Here we will stack logistic regression which has the best result in the above 6 models with neural network model that we have trained on our data. For stacking we will use average weight concept. We will multiply some weights to each of our predicted result and these in total should sum to one. In simple words we are just giving weightage or priorirtizing the models which have minimum log loss.

In [156]:
```
#Weights for both the having evenst and no events predictions
w_no_events_lr=0.15
w_no_events_nn_1=0.75
w_no_events_nn_2=0.1
w_events_lr=0.45
w_events_nn_1=0.45
w_events_nn_2=0.1
```

In [157]:
```
#Adding the predictions after multipying it by weights
pred_no_events=(w_no_events_lr*y_no_events_pred_logistic_regression)+(w_no_event
pred_events=(w_events_lr*test_pred_avg_events_1)+(w_events_nn_1*test_pred_avg_ev
```

In [159]:
```
#Converting the predictions to dataset
pred_1 = pd.DataFrame(pred_no_events, index = no_events_data_test.index,columns=
pred_2 = pd.DataFrame(pred_events, index = having_events_data_test.index,columns
```

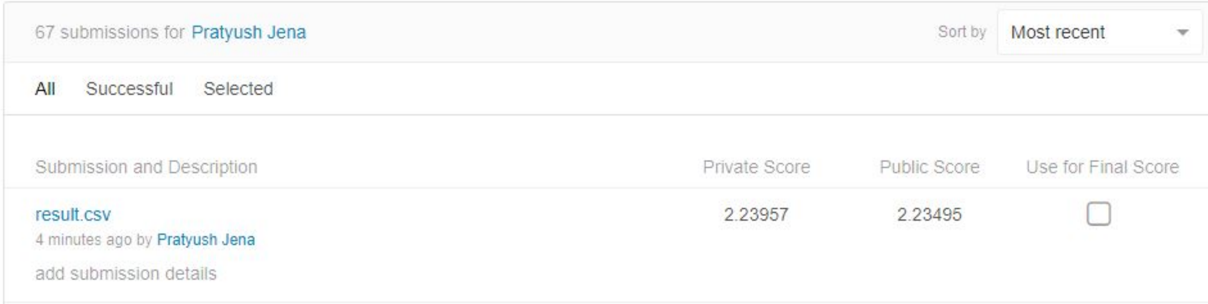
In [162]:
```
#Concaneting the predicted data sets both for having events and no events
result=pd.concat([pred_1,pred_2], axis=0)
```

In [163]:
```
#Generating the result
result.to_csv("result.csv")
```

Final Result in Kaggle

In [167]:
```
Image('Pred/result.jpg')
```

Out[167]:



Observation

After submitting the result.csv in the kaggle competition I am getting the following results:

In [168]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
```

In [169]:
```
x.field_names = ["Leadership Board", "Rank","Loss"]
x.add_row(["Public", 154,2.23495])
x.add_row(["Private", 159,2.23975])
```

In [170]:
```
print(x)
```

```
+------------------+------+---------+
| Leadership Board | Rank |   Loss  |
+------------------+------+---------+
|      Public      | 154  | 2.23495 |
|     Private      | 159  | 2.23975 |
+------------------+------+---------+
```

7. Conclusion

In [173]:
```
# Please compare all your models using Prettytable library
x = PrettyTable()
```

In [174]:
```
x.field_names = ["Feature Name", "Log Loss"]
x.add_row(["KNN", 2.35637])
x.add_row(["SVM", 2.36538])
x.add_row(["Logistic Regression", 2.27643])
x.add_row(["XGBoost", 2.29377])
x.add_row(["Random Forest", 2.32075])
x.add_row(["Decision Tree", 2.36851])
x.add_row(["Stacking Model",2.23495])
```

In [175]:
```
print(x)
```

```
+---------------------+----------+
|     Feature Name    | Log Loss |
+---------------------+----------+
|         KNN         | 2.35637  |
|         SVM         | 2.36538  |
| Logistic Regression | 2.27643  |
|       XGBoost       | 2.29377  |
|    Random Forest    | 2.32075  |
|    Decision Tree    | 2.36851  |
|    Stacking Model   | 2.23495  |
+---------------------+----------+
```

We can see that stacking model of Logistic regression and neural network gives the best result for above data set with minimum loss of 2.23495 on predicting the age group for the test data given. And hence we are able to get top 10% rank in the kaggle competition.