KDD CUP 2012 TRACK2

Problem Description

Introduction:
Clickthrough rate (CTR) is a ratio showing how often people who see your ad end up clicking it. Clickthrough rate (CTR) can be used to gauge how well your keywords and ads are performing.

- CTR is the number of clicks that your ad receives divided by the number of times your ad is shown: clicks ÷ impressions = CTR. For example, if you had 5 clicks and 100 impressions, then your CTR would be 5%.
- Each of your ads and keywords have their own CTRs that you can see listed in your account.
- A high CTR is a good indication that users find your ads helpful and relevant. CTR also contributes to your keyword's expected CTR, which is a component of Ad Rank. Note that a good CTR is relative to what you're advertising and on which networks.

> Credits: Google (https://support.google.com/adwords/answer/2615875?hl=en (https://support.google.com/adwords/answer/2615875?hl=en))

Search advertising has been one of the major revenue sources of the Internet industry for years. A key technology behind search advertising is to predict the click-through rate (pCTR) of ads, as the economic model behind search advertising requires pCTR values to rank ads and to price clicks. **In this task, given the training instances derived from session logs of the Tencent proprietary search engine, soso.com, participants are expected to accurately predict the pCTR of ads in the testing instances.**

Source/Useful Links

Source : https://www.kaggle.com/c/kddcup2012-track2 (https://www.kaggle.com/c/kddcup2012-track2)

Dropbox Links : https://www.dropbox.com/sh/k84z8y9n387ptjb/AAA8O8IDFsSRhOhaLfXVZcJwa?dl=0 (https://www.dropbox.com/sh/k84z8y9n387ptjb/AAA8O8IDFsSRhOhaLfXVZcJwa?dl=0)

Blog :https://hivemall.incubator.apache.org/userguide/regression/kddcup12tr2_dataset.html (https://hivemall.incubator.apache.org/userguide/regression/kddcup12tr2_dataset.html)

Real-world/Business Objectives and Constraints

Objective: Given query and user information, we need to predict if the user would click the add.

Constraints: Low latency, Interpretability.

Machine Learning problem

Data Overview

Data Files

Filename	Available Format
training	.txt (9.9Gb)
queryid_tokensid	.txt (704Mb)
purchasedkeywordid_tokensid	.txt (26Mb)
titleid_tokensid	.txt (172Mb)
descriptionid_tokensid	.txt (268Mb)
userid_profile	.txt (284Mb)

training.txt

Feature	Description
UserID	The unique id for each user
AdID	The unique id for each ad
QueryID	The unique id for each Query (it is a primary key in Query table(queryid_tokensid.txt))
Depth	The number of ads impressed in a session is known as the 'depth'.
Position	The order of an ad in the impression list is known as the ‘position’ of that ad.
Impression	The number of search sessions in which the ad (AdID) was impressed by the user (UserID) who issued the query (Query).
Click	The number of times, among the above impressions, the user (UserID) clicked the ad (AdID).
TitleId	A property of ads. This is the key of 'titleid_tokensid.txt'. [An Ad, when impressed, would be displayed as a short text known as ’title’, followed by a slightly longer text known as the ’description’, and a URL (usually shortened to save screen space) known as ’display URL’.]
DescId	A property of ads. This is the key of 'descriptionid_tokensid.txt'. [An Ad, when impressed, would be displayed as a short text known as ’title’, followed by a slightly longer text known as the ’description’, and a URL (usually shortened to save screen space) known as ’display URL’.]
AdURL	The URL is shown together with the title and description of an ad. It is usually the shortened landing page URL of the ad, but not always. In the data file, this URL is hashed for anonymity.
KeyId	A property of ads. This is the key of 'purchasedkeyword_tokensid.txt'.
AdvId	a property of the ad. Some advertisers consistently optimize their ads, so the title and description of their ads are more attractive than those of others’ ads.

There are five additional data files, as mentioned in the above section:

1. queryid_tokensid.txt
2. purchasedkeywordid_tokensid.txt
3. titleid_tokensid.txt

4. descriptionid_tokensid.txt
5. userid_profile.txt

Each line of the first four files maps an id to a list of tokens, corresponding to the query, keyword, ad title, and ad description, respectively. In each line, a TAB character separates the id and the token set. A token can basically be a word in a natural language. For anonymity, each token is represented by its hash value. Tokens are delimited by the character ‘|’.

Each line of ‘userid_profile.txt’ is composed of UserID, Gender, and Age, delimited by the TAB character. Note that not every UserID in the training and the testing set will be present in ‘userid_profile.txt’. Each field is described below:

1. Gender: '1' for male, '2' for female, and '0' for unknown.
2. Age: '1' for (0, 12], '2' for (12, 18], '3' for (18, 24], '4' for (24, 30], '5' for (30, 40], and '6' for greater than 40

Example Data point

training.txt

```
Click Impression    AdURL        AdId      AdvId  Depth Pos  QId       KeyId    Tit
leId  DescId  UId
0    1   4298118681424644510    7686695 385     3     3  1601       5521     7709
576    490234
0    1   4860571499428580850    21560664    37484     2   2  2255103    317      48
989    44771  490234
0    1   9704320783495875564    21748480    36759     3   3  4532751    60721    68
5038   29681  490234
```

queryid_tokensid.txt

```
QId Query
0   12731
1   1545|75|31
2   383
3   518|1996
4   4189|75|31
```

purchasedkeywordid_tokensid.txt

titleid_tokensid.txt

```
TitleId Title
0   615|1545|75|31|1|138|1270|615|131
1   466|582|685|1|42|45|477|314
2   12731|190|513|12731|677|183
3   2371|3970|1|2805|4340|3|2914|10640|3688|11|834|3
4   165|134|460|2887|50|2|17527|1|1540|592|2181|3|...
```

descriptionid_tokensid.txt

```
DescId  Description
0   1545|31|40|615|1|272|18889|1|220|511|20|5270|1...
1   172|46|467|170|5634|5112|40|155|1965|834|21|41...
2   2672|6|1159|109662|123|49933|160|848|248|207|1...
3   13280|35|1299|26|282|477|606|1|4016|1671|771|1...
4   13327|99|128|494|2928|21|26500|10|11733|10|318
```

userid_profile.txt

```
UId Gender  Age
1   1   5
2   2   3
3   1   5
4   1   3
5   2   1
```

Mapping the Real-world to a Machine Learning problem

Performance metric

Source : https://www.kaggle.com/c/kddcup2012-track2#Evaluation (https://www.kaggle.com/c/kddcup2012-track2#Evaluation)

ROC: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/)

Type of Machine Learning Problem

Classification Problem -> Given query and user information, we need to predict if the user would click the add.

Usefull links

Source : https://www.kaggle.com/c/kddcup2012-track2 (https://www.kaggle.com/c/kddcup2012-track2)

pdf : https://jyunyu.csie.org/docs/pubs/kddcup2012paper.pdf (https://jyunyu.csie.org/docs/pubs/kddcup2012paper.pdf)

In [1]:

```
# Importing Libraries

import pandas as pd
import numpy as np
from datetime import datetime
import tqdm
import re
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt

from scipy import spatial
from xgboost import XGBClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack
from scipy.sparse import csr_matrix
from sklearn.preprocessing import normalize
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics.pairwise import cosine_similarity
from gensim.models import Word2Vec
from prettytable import PrettyTable
from tqdm import tqdm

import warnings
warnings.filterwarnings("ignore")
```

```
C:\Users\Administrator\Anaconda3\lib\site-packages\gensim\utils.py:1209: User
Warning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
# Loading training data...

# Here we took  1 million datapoints due to lack of computational resources

column  = ['clicks', 'impressions', 'AdURL', 'AdId', 'AdvId', 'Depth', 'Pos',
'QId', 'KeyId', 'TitleId', 'DescId', 'UId']
train   = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_
Track 2/training.txt', sep='\t', header=None, names=column,nrows = 1000000)
train.head()

# print(type(train['AdURL'][0])) -> <class 'numpy.uint64'>
# print(type(train['UId'][0])) -> <class 'numpy.int64'>
# train.shape -> (100000, 12)
```

Out[2]:

	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	QId	K
0	0	1	4298118681424644510	7686695	385	3	3	1601	5
1	0	1	4860571499428580850	21560664	37484	2	2	2255103	3
2	0	1	9704320783495875564	21748480	36759	3	3	4532751	6
3	0	1	13677630321509009335	3517124	23778	3	1	1601	2
4	0	1	3284760244799604489	20758093	34535	1	1	4532751	7

In [16]:
```
train  = train.reset_index()
```

In [27]:
```
# For each training instance, we first split it into (#click) positive samples
and

# (#impression-#click) negative samples. Then we train a classifier to discrim
inate

# positive samples from negative ones.
```

In [6]:
```
# Replicating each instance by (#impressions)

start = datetime.now()


for index, row in train.iterrows():
    if row['impressions'] > 1:
        train = train.append([row]*(int(row['impressions']-1)),ignore_index=Tr
ue)

    if index%10000 == 0:
        print("Rows completed : ", index)

end =datetime.now()
print("Time taken to run this cell: ",end-start)

# train.shape -> (121760, 15)
```

```
Rows completed :  0
Rows completed :  10000
Rows completed :  20000
Rows completed :  30000
Rows completed :  40000
Rows completed :  50000
Rows completed :  60000
Rows completed :  70000
Rows completed :  80000
Rows completed :  90000
Rows completed :  100000
Rows completed :  110000
Rows completed :  120000
Rows completed :  130000
Rows completed :  140000
Rows completed :  150000
Rows completed :  160000
Rows completed :  170000
Rows completed :  180000
Rows completed :  190000
Rows completed :  200000
Rows completed :  210000
Rows completed :  220000
Rows completed :  230000
Rows completed :  240000
Rows completed :  250000
Rows completed :  260000
Rows completed :  270000
Rows completed :  280000
Rows completed :  290000
Rows completed :  300000
Rows completed :  310000
Rows completed :  320000
Rows completed :  330000
Rows completed :  340000
Rows completed :  350000
Rows completed :  360000
Rows completed :  370000
Rows completed :  380000
Rows completed :  390000
Rows completed :  400000
Rows completed :  410000
Rows completed :  420000
Rows completed :  430000
Rows completed :  440000
Rows completed :  450000
Rows completed :  460000
Rows completed :  470000
Rows completed :  480000
Rows completed :  490000
Rows completed :  500000
Rows completed :  510000
Rows completed :  520000
Rows completed :  530000
Rows completed :  540000
Rows completed :  550000
Rows completed :  560000
```

```
Rows completed :  570000
Rows completed :  580000
Rows completed :  590000
Rows completed :  600000
Rows completed :  610000
Rows completed :  620000
Rows completed :  630000
Rows completed :  640000
Rows completed :  650000
Rows completed :  660000
Rows completed :  670000
Rows completed :  680000
Rows completed :  690000
Rows completed :  700000
Rows completed :  710000
Rows completed :  720000
Rows completed :  730000
Rows completed :  740000
Rows completed :  750000
Rows completed :  760000
Rows completed :  770000
Rows completed :  780000
Rows completed :  790000
Rows completed :  800000
Rows completed :  810000
Rows completed :  820000
Rows completed :  830000
Rows completed :  840000
Rows completed :  850000
Rows completed :  860000
Rows completed :  870000
Rows completed :  880000
Rows completed :  890000
Rows completed :  900000
Rows completed :  910000
Rows completed :  920000
Rows completed :  930000
Rows completed :  940000
Rows completed :  950000
Rows completed :  960000
Rows completed :  970000
Rows completed :  980000
Rows completed :  990000
Rows completed :  1000000
Time taken to run this cell:  4:21:54.495156
```

In [15]:

```
# total no of rows: 1235092
```

In [8]:

```
print(type(train['AdURL'][0]))
```

```
<class 'numpy.float64'>
```

In [9]:
```
# Saving to csv file

train.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/train_10L.csv',index=False)
```

In [ ]:
```
# Loading the data in original format

data = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/train_10L.csv",
                 dtype={'index':np.int64,'AdURL': np.uint64,'AdId':np.int64,'AdvId':np.int64,'Depth':np.int64,'Pos':np.int64,'QId':np.int64,'KeyId':np.int64,'TitleId':np.int64,'DescId':np.int64,'UId':np.int64})
```

In [ ]:
```
data['class_label'] = 0
data_n = data   # storing dataframe in a temporary variable which is used in the below snippet
data.head()
```

In [15]:

```
# labelling the datapoints

start = datetime.now()

indeces = list()

for i, row in data.iterrows():
    if i<1000000:   # no.of original rows(before replication)
        data_n = data
        if row['clicks'] == 0 and row['impressions'] == 1:
            data['class_label'] = 0
        elif row['clicks'] == 1 and row['impressions'] == 1:
            data['class_label'] = 1
        else:
            data_n = data_n.loc[data_n['index']==row['index']]
            if row['clicks']>=1:
                clicks_num = row['clicks']
                data_n = data_n.sample(int(clicks_num))
                ind = data_n.index  # Example: data_n.index -> Int64Index([100
000, 30], dtype='int64')
                indeces.extend(ind)
        if i%10000 == 0:
            print("Rows completed : ", i)

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Rows completed :  0
Rows completed :  10000
Rows completed :  20000
Rows completed :  30000
Rows completed :  40000
Rows completed :  50000
Rows completed :  60000
Rows completed :  70000
Rows completed :  80000
Rows completed :  90000
Rows completed :  100000
Rows completed :  110000
Rows completed :  120000
Rows completed :  130000
Rows completed :  140000
Rows completed :  150000
Rows completed :  160000
Rows completed :  170000
Rows completed :  180000
Rows completed :  190000
Rows completed :  200000
Rows completed :  210000
Rows completed :  220000
Rows completed :  230000
Rows completed :  240000
Rows completed :  250000
Rows completed :  260000
Rows completed :  270000
Rows completed :  280000
Rows completed :  290000
Rows completed :  300000
Rows completed :  310000
Rows completed :  320000
Rows completed :  330000
Rows completed :  340000
Rows completed :  350000
Rows completed :  360000
Rows completed :  370000
Rows completed :  380000
Rows completed :  390000
Rows completed :  400000
Rows completed :  410000
Rows completed :  420000
Rows completed :  430000
Rows completed :  440000
Rows completed :  450000
Rows completed :  460000
Rows completed :  470000
Rows completed :  480000
Rows completed :  490000
Rows completed :  500000
Rows completed :  510000
Rows completed :  520000
Rows completed :  530000
Rows completed :  540000
Rows completed :  550000
Rows completed :  560000
```

```
Rows completed :  570000
Rows completed :  580000
Rows completed :  590000
Rows completed :  600000
Rows completed :  610000
Rows completed :  620000
Rows completed :  630000
Rows completed :  640000
Rows completed :  650000
Rows completed :  660000
Rows completed :  670000
Rows completed :  680000
Rows completed :  690000
Rows completed :  700000
Rows completed :  710000
Rows completed :  720000
Rows completed :  730000
Rows completed :  740000
Rows completed :  750000
Rows completed :  760000
Rows completed :  770000
Rows completed :  780000
Rows completed :  790000
Rows completed :  800000
Rows completed :  810000
Rows completed :  820000
Rows completed :  830000
Rows completed :  840000
Rows completed :  850000
Rows completed :  860000
Rows completed :  870000
Rows completed :  880000
Rows completed :  890000
Rows completed :  900000
Rows completed :  910000
Rows completed :  920000
Rows completed :  930000
Rows completed :  940000
Rows completed :  950000
Rows completed :  960000
Rows completed :  970000
Rows completed :  980000
Rows completed :  990000
Time taken to run this cell:  3:03:43.298530
```

In [16]:
```
indeces = list(dict.fromkeys(indeces)) # to remove duplicate indeces if any
indeces
```

```
Out[16]: [1228023,
 33,
 1001305,
 1228148,
 1000093,
 1228209,
 494,
 1228242,
 1228243,
 1001447,
 1228345,
 1228388,
 1228394,
 1000196,
 1228468,
 1228479,
 1228509,
 1228446,
 1228493,
 1228511,
 1228475,
 1081,
 1105,
 1000258,
 1485,
 1229061,
 1229356,
 1000425,
 2440,
 1229514,
 1229699,
 1000596,
 1230210,
 1001942,
 1232384,
 1232385,
 1000714,
 1232396,
 1232395,
 1000721,
 1002016,
 1232520,
 1000792,
 1232536,
 1232557,
 1002154,
 1002169,
 1232685,
 1002172,
 1232707,
 1233335,
 1000988,
 1233333,
 1233575,
 1234314,
 1233853,
 1234209,
```

```
1233942,
1234224,
1233842,
1233813,
1233763,
1234001,
1233386,
4718,
1234434,
1001045,
1234580,
1001110,
1002431,
1001169,
1002471,
1001176,
1234854,
1001263,
1235079,
1002580,
1002597,
1002605,
1002650,
1002671,
6965,
7012,
1002682,
7057,
7431,
7479,
1002795,
7918,
1002819,
1002854,
1002859,
8047,
8129,
1002887,
1002944,
1002957,
1002986,
1003001,
9079,
1003029,
1003041,
1003043,
1003044,
9199,
1003059,
9249,
1003071,
9254,
1003082,
1003095,
1003112,
1003160,
1003161,
```

```
1003163,
10072,
10161,
10471,
1003314,
10844,
11184,
1003407,
11300,
1003440,
1003560,
11939,
1003642,
1003678,
12255,
12313,
12350,
12496,
1003721,
12506,
12507,
1003746,
12602,
12628,
1003763,
1003771,
1003801,
12856,
13001,
1003813,
13017,
1003825,
1003850,
13216,
1003867,
1003866,
1003868,
13424,
1003890,
13535,
1003904,
1003919,
1003943,
13970,
1004038,
1004055,
1004060,
14174,
14848,
1004183,
1004184,
1004185,
1004187,
1004234,
1004236,
1004235,
1004285,
```

```
1004288,
15163,
1004298,
15279,
15299,
1004317,
1004328,
15369,
15406,
15492,
15701,
15830,
15843,
1004413,
15989,
16001,
1004497,
16511,
16638,
16831,
16870,
1004580,
1004583,
1004635,
17166,
1004643,
1004678,
1004689,
17346,
1004690,
1004745,
17527,
17792,
17860,
1004833,
1004894,
18566,
1004905,
1004923,
18777,
1004959,
18891,
1004984,
18917,
1004997,
1005030,
1005033,
19229,
1005039,
19275,
1005084,
1005085,
1005101,
1005103,
1005138,
19631,
1005141,
```

```
1005159,
1005175,
1005185,
19992,
20034,
1005211,
20066,
1005263,
20458,
1005278,
1005324,
20847,
21111,
1005449,
1005489,
1005495,
1005494,
21484,
1005511,
1005524,
1005563,
1005621,
1005701,
1005700,
22339,
1005704,
1005747,
23137,
1005816,
1005856,
23330,
1005870,
1005887,
1005939,
1005944,
24063,
1005964,
24251,
24354,
1006048,
1006051,
1006054,
24631,
1006055,
1006057,
24667,
1006067,
1006115,
1006113,
1006112,
25003,
1006114,
1006116,
1006117,
25051,
1006130,
25058,
```

```
25127,
1006175,
1006237,
1006333,
1006354,
1006383,
1006388,
1006409,
1006410,
26331,
1006445,
1006455,
1006473,
1006475,
26816,
1006502,
1006504,
1006521,
1006546,
1006542,
1006547,
1006551,
27462,
27533,
1006628,
1006754,
1006758,
1006759,
28260,
1006765,
28277,
28977,
1006907,
1006908,
1006910,
29404,
1007032,
29525,
1007055,
29532,
1007062,
1007064,
1007067,
1007087,
1007086,
29726,
1007100,
1007098,
1007099,
29763,
1007128,
1007157,
1007184,
30485,
1007185,
30537,
1007196,
```

```
1007260,
1007303,
1007328,
31302,
1007349,
1007426,
1007428,
1007433,
1007442,
1007444,
1007447,
1007449,
1007501,
32213,
1007520,
1007518,
1007526,
1007536,
1007537,
32358,
1007540,
1007557,
1007565,
1007563,
32563,
32894,
32961,
33044,
1007952,
1007964,
1007998,
33526,
1008015,
33755,
34138,
34585,
34816,
35423,
1008356,
1008392,
1008426,
1008427,
1008494,
1008507,
1008523,
36389,
1008539,
36773,
1008590,
36819,
1008595,
1008602,
1008657,
37498,
37605,
1008690,
37964,
```

```
1008746,
37984,
1008754,
1008755,
1008762,
1008812,
1008851,
38500,
1008879,
39056,
1008996,
1008998,
39266,
1009000,
39742,
1009070,
40033,
1009104,
40037,
40324,
40400,
40753,
1009243,
1009270,
1009274,
41087,
41141,
1009300,
41507,
1009431,
1009432,
41544,
41546,
1009435,
41549,
41562,
41631,
41709,
41710,
41802,
42038,
1009600,
1009601,
1009603,
1009624,
1009630,
1009629,
42375,
42426,
1009646,
42465,
42552,
42599,
42695,
1009684,
1009704,
42806,
```

```
                1009809,
                43456,
                43479,
                1009871,
                1009870,
                43709,
                43778,
                1009891,
                1009892,
                1009905,
                43929,
                1009913,
                1009943,
                1009944,
                1009947,
                43952,
                43956,
                43993,
                44021,
                1009962,
                1010008,
                1010020,
                44432,
                1010046,
                44571,
                1010060,
                44711,
                45052,
                45078,
                1010121,
                45081,
                1010186,
                45502,
                45637,
                1010253,
                45998,
                1010302,
                1010305,
                1010319,
                46473,
                46505,
                1010353,
                1010354,
                46506,
                1010379,
                1010473,
                1010504,
                47216,
                47218,
                1010505,
                1010534,
                47539,
                1010553,
                1010581,
                1010584,
                1010576,
                1010588,
```

```
1010589,
1010630,
1010634,
1010690,
48330,
1010786,
48335,
48336,
1010788,
48341,
1010792,
1010823,
48765,
49200,
1010915,
1010914,
49207,
1010918,
1010913,
1010924,
1010935,
49373,
49377,
1010951,
1010949,
1010948,
1010952,
1010950,
1010953,
49433,
49596,
1011013,
1011052,
1011053,
1011054,
50051,
1011184,
50524,
1011205,
1011213,
1011214,
1011249,
1011254,
51006,
1011287,
51279,
51318,
1011339,
1011345,
51977,
51985,
52401,
52572,
1011446,
52606,
52825,
1011490,
```

```
52985,
1011521,
1011545,
1011547,
53239,
1011599,
1011682,
1011728,
54418,
54526,
55009,
1012155,
1012154,
55157,
1012172,
1012195,
55421,
1012235,
1012267,
55643,
1012506,
1012521,
55840,
55902,
56016,
1012595,
56335,
1012666,
1012691,
1012713,
56798,
1012819,
1012841,
57324,
1012862,
57326,
57329,
1013004,
1013039,
1013117,
58179,
1013390,
1013659,
59115,
59421,
1013741,
59849,
59943,
1013779,
59968,
1013844,
1013854,
1014208,
1014214,
60480,
1014235,
60582,
```

```
60658,
60675,
60849,
1014277,
1014286,
1014300,
60853,
61071,
1014370,
1014578,
62314,
62317,
1014600,
1014595,
1014596,
1014604,
1014601,
1014602,
1014607,
1014605,
62324,
1014609,
1014610,
62341,
1014659,
1014671,
62809,
1014673,
62903,
63470,
1014769,
1014770,
63517,
1014771,
63532,
1014781,
64077,
64081,
64367,
64755,
1015061,
1015067,
1015072,
1015085,
65042,
1015102,
1015103,
65043,
65044,
65090,
1015142,
1015214,
1015225,
1015252,
1015279,
66096,
1015318,
```

```
66482,
1015356,
66643,
66645,
1015380,
66701,
1015399,
1015400,
67099,
1015436,
67129,
1015462,
1015468,
1015482,
67209,
68034,
1015754,
1015770,
68381,
1015792,
68824,
68858,
1015955,
68881,
1015967,
1015968,
1015971,
68972,
69009,
69116,
1016095,
69564,
1016117,
1016118,
1016119,
69977,
70038,
1016227,
70584,
1016271,
1016273,
1016339,
70886,
71083,
71147,
71187,
1016435,
1016464,
71404,
1016465,
1016467,
1016468,
71424,
1016516,
1016549,
1016548,
1016550,
```

```
71955,
1016607,
1016613,
72291,
1016626,
1016627,
1016628,
1016641,
72489,
1016671,
1016706,
72938,
1016736,
73153,
73265,
1016760,
73707,
1016928,
1016930,
73864,
73866,
1016933,
1016936,
73928,
74147,
1017014,
1017019,
1017023,
74515,
1017047,
1017050,
1017079,
1017085,
74762,
1017091,
1017102,
1017116,
1017134,
75087,
1017133,
1017163,
1017176,
1017216,
1017313,
1017344,
1017348,
76323,
76976,
1017468,
1017471,
1017551,
77930,
78006,
1017617,
1017637,
78272,
1017678,
```

```
1017755,
1017770,
1018305,
1018314,
1018321,
79438,
1018360,
1018362,
79601,
1018399,
80128,
80184,
80253,
1018529,
80403,
1018577,
1018583,
1018612,
1018651,
81107,
81131,
1018670,
1018669,
1018671,
1018697,
1018723,
1018733,
81450,
1018752,
81921,
1018818,
81937,
81989,
1018837,
1018840,
82310,
1018962,
82620,
1019017,
1019018,
1019045,
1019061,
82970,
83058,
1019074,
83369,
1019199,
83769,
83783,
83791,
1019252,
1019254,
1019324,
1019333,
84078,
84209,
1019377,
```

```
1019466,
1019467,
1019476,
84497,
1019504,
84629,
84646,
1019522,
84759,
1019533,
84761,
1019552,
84968,
1019637,
1019648,
85354,
85671,
1019762,
85881,
1019777,
86061,
1019809,
86204,
1019901,
1019903,
1019904,
1019902,
1019900,
1019920,
86316,
1019961,
1020048,
1020063,
1020076,
1020087,
1020086,
1020093,
1020079,
1020080,
1020089,
1020073,
1020075,
1020077,
1020083,
1020095,
1020091,
1020074,
1020081,
1020084,
1020072,
1020078,
1020094,
1020085,
1020090,
1020082,
1020092,
1020088,
```

```
86740,
86807,
1020130,
1020140,
87134,
1020185,
1020323,
1020328,
1020348,
1020351,
87503,
87994,
88002,
1020423,
1020452,
88395,
88415,
1020474,
1020480,
1020490,
88721,
1020503,
88752,
1020505,
88758,
88798,
88880,
88893,
1020538,
88903,
1020587,
89115,
1020712,
1020711,
89608,
1020726,
89869,
89870,
1020771,
89903,
1020802,
89962,
1020826,
1020892,
1020944,
1020954,
90253,
1020950,
1020952,
1020956,
1020953,
1020955,
1020951,
1020949,
90383,
90476,
90493,
```

```
1021031,
90801,
90901,
90973,
1021152,
1021149,
1021154,
91064,
1021173,
1021201,
1021217,
91246,
91338,
1021321,
91642,
1021356,
91831,
1021384,
1021390,
1021411,
1021417,
1021455,
92199,
1021477,
92206,
92224,
92868,
1021656,
1021661,
1021679,
93473,
...]
```

In [17]:

```
data.iloc[indeces,16] = 1
# 16 -> "class_label" column number
```

In [18]:

```
#duplicateRowsDF = data[data.duplicated(['AdURL','AdId', 'AdvId','UId','QI
d','KeyId','TitleId','DescId','Depth','Pos','impressions'])]
#duplicateRowsDF = data[data.duplicated(['index'])]

# data_n= data_n.loc[data_n['index']==30]
# data_n = data_n.sample(2)
# data_n['label']=1
```

In [4]:

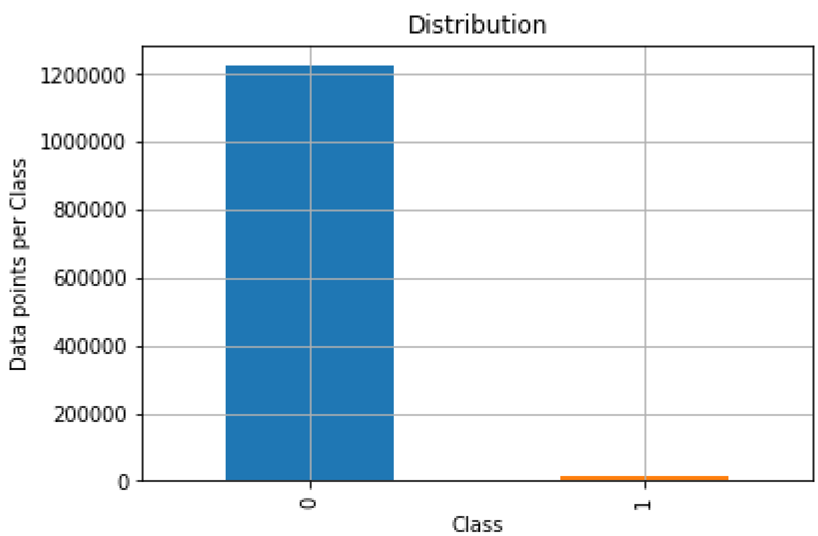
```
# Saving to csv file

data.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/tr
ain_10L_labeled.csv',index=False)
```

In [39]:
```
# Reading from csv file

data = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Tra
ck 2/train_10L_labeled.csv")

data.head()
```

Out[39]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	0	0.0	1.0	4298118681424644608	7686695	385	3	3	160
1	1	0.0	1.0	4860571499428580352	21560664	37484	2	2	225
2	2	0.0	1.0	9704320783495874560	21748480	36759	3	3	453
3	3	0.0	1.0	13677630321509009408	3517124	23778	3	1	160
4	4	0.0	1.0	3284760244799604736	20758093	34535	1	1	453

In [40]:
```
disb = data['class_label'].value_counts().sortlevel()

my_colors = 'rgbkymc'
disb.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution')
plt.grid()
plt.show()
```



In [41]:
```
data['class_label'].value_counts()
```

Out[41]:
```
0    1223664
1      11429
Name: class_label, dtype: int64
```

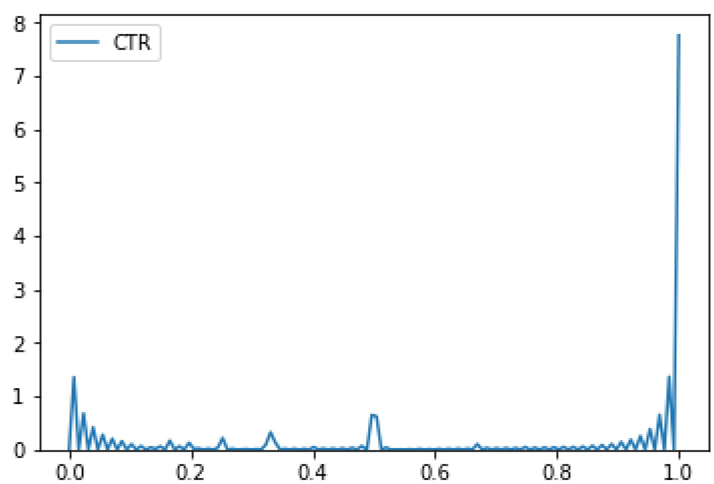
In [42]:
```
# CTR(ad) = #Clicks(ad)/#Impressions(ad)

# Calculating net CTR for our dataset...

total_impressions = data['impressions'].sum()
total_clicks      = data['clicks'].sum()
net_CTR           = total_clicks * 1.0 / total_impressions

print( ('Net CTR: {0}'.format(round(net_CTR*100,2))), '%')
```

```
Net CTR: 3.24 %
```

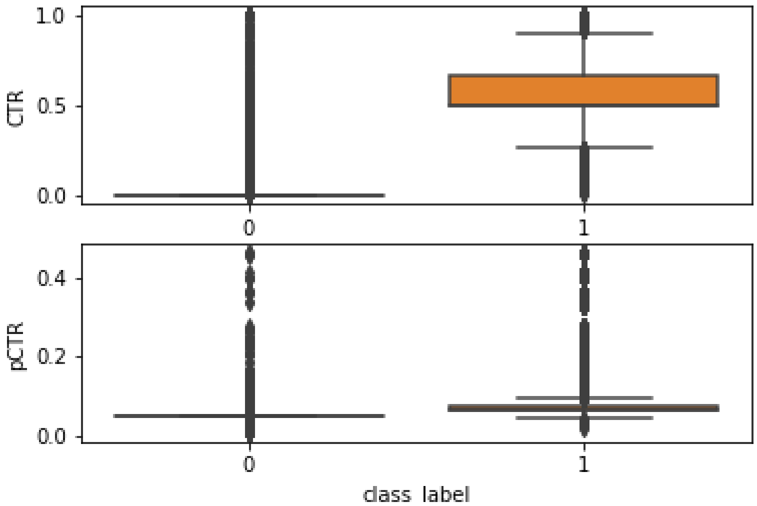
In [43]:
```
# total no. of unique users in the dataset...
print( 'Total no. of unique users:', len(data.groupby('UId')))

# total no. of unique queries in the dataset...
print( 'Total no. of unique queries:', len(data.groupby('QId')))

# total no. of unique advertisements in the dataset...
print( 'Total no. of unique ads:', len(data.groupby('AdId')))

# total no. of unique advertisers in the dataset...
print( 'Total no. of unique advertisers:', len(data.groupby('AdvId')))
```

```
Total no. of unique users: 202547
Total no. of unique queries: 279352
Total no. of unique ads: 99242
Total no. of unique advertisers: 12193
```

In [41]:
```
sns.kdeplot(data['CTR'])
plt.show()
```



In [37]:
```
f, (ax1, ax2) = plt.subplots(2)
sns.boxplot(x='class_label', y='CTR', data=data,ax=ax1)
sns.boxplot(x='class_label', y='pCTR', data=data,ax=ax2)

plt.show()
```



The adds which have high CTR got clicked

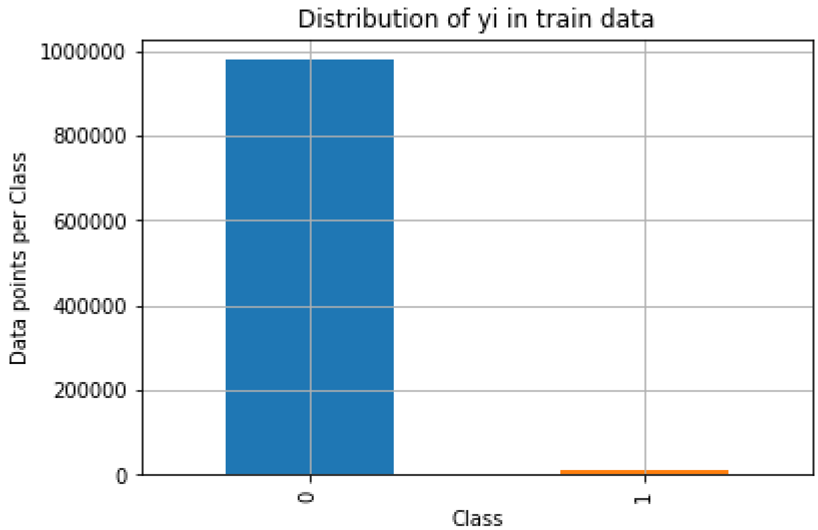
Train Test Split

In [44]:
```
y_true = data['class_label'].values

# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
x_train, x_test, y_train, y_test = train_test_split(data, y_true, stratify=y_t
rue, test_size=0.20)
```

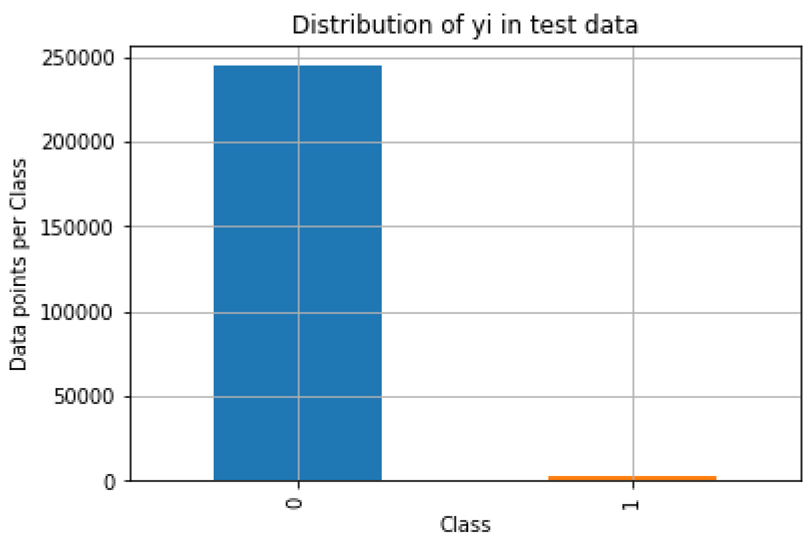
In [45]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(988074, 14)
(247019, 14)
(988074,)
(247019,)
```

In [46]:
```
train_class_distribution = x_train['class_label'].value_counts().sortlevel()
test_class_distribution = x_test['class_label'].value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-train_class_distribution.values) #Return a Numpy repre
sentation of the DataFrame
for i in sorted_yi:
    print('Number of data points in class', i, ':',train_class_distribution.va
lues[i], '(', np.round((train_class_distribution.values[i]/x_train.shape[0]*10
0), 3), '%)')


print('-'*80)


my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i, ':',test_class_distribution.val
ues[i], '(', np.round((test_class_distribution.values[i]/x_test.shape[0]*100),
3), '%)')
```




```
Number of data points in class 0 : 978931 ( 99.075 %)
Number of data points in class 1 : 9143 ( 0.925 %)
-----------------------------------------------------------------------------
---
```




```
Number of data points in class 0 : 244733 ( 99.075 %)
Number of data points in class 1 : 2286 ( 0.925 %)
```

In [51]:
```
# we observe that some categories come with only a few or even no instances.

# Computing the click-through rate directly for those categories would result
 in inaccurate estimations

# because of the insufficient statistics. Thus, we apply smoothing methods durin
g click-through rate estima-tion.

# We mainly use a simple additive smoothing  pseudo-CTR = click + α × β #impre
ssion + β

# and we name it pseudo click-through rate (pseudo-CTR). In our experiments, w
e set α as 0.05 and β as 75.
```

In [47]:
```
# Add target variable CTR as #clicks / #impression

x_train['CTR'] = x_train['clicks'] * 1.0 / x_train['impressions']

#adding relative position as a new feature
x_train['RPosition'] = x_train['Depth'] - x_train['Pos'] * 1.0 / x_train['Dept
h']

# Add predicted CTR as #clicks + ab / #impressions + b
x_train['pCTR'] = (1.0 * x_train['clicks'] + 0.05 * 75) / (x_train['impression
s'] + 75)

x_train.head()
```

Out[47]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pc
35803	35803	0.0	1.0	14340390157469405184	4803006	23777	1	1
760987	760987	0.0	1.0	4298118681424644608	7831406	385	2	2
360323	360323	0.0	2.0	14340390157469405184	21163923	23808	1	1
158858	158858	0.0	2.0	13021740539055374336	21163748	8222	1	1
7867	7867	1.0	1.0	2692859619851282432	8692018	2051	2	2

In [48]:
```
# Add target variable CTR as #clicks / #impression

x_test['CTR'] = x_test['clicks'] * 1.0 / x_test['impressions']

#adding relative position as a new feature
x_test['RPosition'] = x_test['Depth'] - x_test['Pos'] * 1.0 / x_test['Depth']

# Add predicted CTR as #clicks + ab / #impressions + b
x_test['pCTR'] = (1.0 * x_test['clicks'] + 0.05 * 75) / (x_test['impressions']
+ 75)

x_test.head()
```

Out[48]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	F
683475	683475	0.0	1.0	15349556856043354112	22098439	36855	3	3
769904	769904	0.0	2.0	9740463035797751808	21423590	11536	3	2
1024586	106870	1.0	2.0	14340390157469405184	20643978	23808	3	1
635839	635839	0.0	1.0	15234713295054012416	4255733	23964	2	1
1165021	725076	0.0	6.0	12402410148710070272	7817411	27726	2	1

In [49]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(988074, 17)
(247019, 17)
```

In [50]:
```
#  Now, we will load additional files provided in the problem, extract useful
 info. from them & merge
```

In [51]:
```
def count(sentence):
    '''
        (str) -> (int)
        Returns no. of words in a sentence.
    '''
    return len(str(sentence).split('|'))




# Load User Data..

user_col  = ['UId', 'Gender', 'Age']
user      = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/userid_profile.txt', sep='\t', header=None, names=user_col)

# Load Query Data..

query_col = ['QId', 'Query']
query     = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/queryid_tokensid.txt', sep='\t', header=None, names=query_col)

# Load Ad Description Data..

desc_col  = ['DescId', 'Description']
desc      = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/descriptionid_tokensid.txt', sep='\t', header=None, names=desc_col)

# Load Ad Title Data..

title_col = ['TitleId', 'Title']
title     = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/titleid_tokensid.txt', sep='\t', header=None, names=title_col)

# Load Keyword Data..

key_col  = ['KeyId', 'Keyword']
keyword  = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012
_Track 2/purchasedkeywordid_tokensid.txt', sep='\t', header=None, names=key_co
l)

# Count no. of tokens in a query issued by a user.

query['QCount'] = query['Query'].apply(count)
del query['Query']

# Count no. of tokens in title of an advertisement.

title['TCount'] = title['Title'].apply(count)
del title['Title']

# Count no. of tokens in description of an advertisement.

desc['DCount'] = desc['Description'].apply(count)
del desc['Description']
```

```
# Count no. of tokens in purchased keyword.

keyword['KCount'] = keyword['Keyword'].apply(count)
del keyword['Keyword']
```

In [52]:

```
# Merging data with user, query, title, keyword & desc on appropriate keys to
 get data..

x_train  = pd.merge(x_train, user,  on='UId')
x_train  = pd.merge(x_train, query, on='QId')
x_train  = pd.merge(x_train, title, on='TitleId')
x_train  = pd.merge(x_train, desc,  on='DescId')
x_train  = pd.merge(x_train, keyword, on='KeyId')

x_train.head()
```

Out[52]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	35803	0.0	1.0	14340390157469405184	4803006	23777	1	1	229
1	35805	0.0	2.0	14340390157469405184	4803006	23777	2	1	229
2	920342	0.0	1.0	14340390157469405184	4803006	23777	3	1	229
3	78016	0.0	1.0	14340390157469405184	4803006	23777	2	1	229
4	831444	0.0	1.0	14340390157469405184	4803006	23777	2	1	229

5 rows × 23 columns

In [53]:

```
# Merging data with user, query, title, keyword & desc on appropriate keys to
 get data..

x_test  = pd.merge(x_test, user,  on='UId')
x_test  = pd.merge(x_test, query, on='QId')
x_test  = pd.merge(x_test, title, on='TitleId')
x_test  = pd.merge(x_test, desc,  on='DescId')
x_test  = pd.merge(x_test, keyword, on='KeyId')

x_test.head()
```

Out[53]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	683475	0.0	1.0	15349556856043354112	22098439	36855	3	3	128
1	969618	0.0	1.0	15349556856043354112	22098439	36855	3	2	128
2	683473	0.0	1.0	15349556856043354112	22098439	36855	2	2	528
3	616976	0.0	1.0	15349556856043354112	22098457	36855	2	1	333
4	714143	0.0	1.0	15349556856043354112	22098457	36855	1	1	333

5 rows × 23 columns

In [54]:
```
x_train.isnull().sum()
```

Out[54]:
```
index          0
clicks         0
impressions    0
AdURL          0
AdId           0
AdvId          0
Depth          0
Pos            0
QId            0
KeyId          0
TitleId        0
DescId         0
UId            0
class_label    0
CTR            0
RPosition      0
pCTR           0
Gender         0
Age            0
QCount         0
TCount         0
DCount         0
KCount         0
dtype: int64
```

In [55]:
```
# checking the null values in every column of x_test
x_test.isnull().sum()
```

Out[55]:
```
index          0
clicks         0
impressions    0
AdURL          0
AdId           0
AdvId          0
Depth          0
Pos            0
QId            0
KeyId          0
TitleId        0
DescId         0
UId            0
class_label    0
CTR            0
RPosition      0
pCTR           0
Gender         0
Age            0
QCount         0
TCount         0
DCount         0
KCount         0
dtype: int64
```

Feature Engineering

In [56]:
```
def features(data_1,temp,key,op):
    temp_data = temp.groupby(key).agg([op])
    temp_df = pd.DataFrame()
    temp_df[key] = temp_data.index
    temp_df['values'] = temp_data.get_values()
    temp = pd.merge(temp,temp_df, on=key, how='left')
    return temp['values']
```

In [57]:
```
# For each AdID, we compute the average click-through rate for all instances w
ith the
# same AdID, and use this value as a single feature. This feature represents t
he estimated click-through rate given its
# category. We compute this kind of feature for AdID, AdvertiserID, depth, pos
ition,QID,KeyID,TitleID,
# DescID,UID,RPosition->(depth-position)/depth.

start = datetime.now()

x_train['mAdURL']     = features(x_train,x_train[['AdURL', 'CTR']],'AdURL','me
an')
x_train['mAdId']      = features(x_train,x_train[['AdId', 'CTR']],'AdId','mea
n')
x_train['mAdvId']     = features(x_train,x_train[['AdvId', 'CTR']],'AdvId','me
an')
x_train['mDepth']     = features(x_train,x_train[['Depth', 'CTR']],'Depth','me
an')
x_train['mPos']       = features(x_train,x_train[['Pos', 'CTR']],'Pos','mean')
x_train['mQId']       = features(x_train,x_train[['QId', 'CTR']],'QId','mean')
x_train['mKeyId']     = features(x_train,x_train[['KeyId', 'CTR']],'KeyId','me
an')
x_train['mTitleId']   = features(x_train,x_train[['TitleId', 'CTR']],'TitleId'
,'mean')
x_train['mDescId']    = features(x_train,x_train[['DescId', 'CTR']],'DescId',
'mean')
x_train['mUId']       = features(x_train,x_train[['UId', 'CTR']],'UId','mean')
x_train['mRPosition'] = features(x_train,x_train[['RPosition', 'CTR']],'RPosit
ion','mean')
x_train['mGender']    = features(x_train,x_train[['Gender', 'CTR']],'Gender',
'mean')
x_train['mAge']       = features(x_train,x_train[['Age', 'CTR']],'Age','mean')

x_train['pAdURL']     = features(x_train,x_train[['AdURL', 'pCTR']],'AdURL','m
ean')
x_train['pAdId']      = features(x_train,x_train[['AdId', 'pCTR']],'AdId','mea
n')
x_train['pAdvId']     = features(x_train,x_train[['AdvId', 'pCTR']],'AdvId','m
ean')
x_train['pDepth']     = features(x_train,x_train[['Depth', 'pCTR']],'Depth','m
ean')
x_train['pPos']       = features(x_train,x_train[['Pos', 'pCTR']],'Pos','mean'
)
x_train['pQId']       = features(x_train,x_train[['QId', 'pCTR']],'QId','mean'
)
x_train['pKeyId']     = features(x_train,x_train[['KeyId', 'pCTR']],'KeyId','m
ean')
x_train['pTitleId']   = features(x_train,x_train[['TitleId', 'pCTR']],'TitleI
d','mean')
x_train['pDescId']    = features(x_train,x_train[['DescId', 'pCTR']],'DescId',
'mean')
x_train['pUId']       = features(x_train,x_train[['UId', 'pCTR']],'UId','mean'
)
x_train['pRPosition'] = features(x_train,x_train[['RPosition', 'pCTR']],'RPosi
tion','mean')
x_train['pGender']    = features(x_train,x_train[['Gender', 'pCTR']],'Gender',
```

```
'mean')
x_train['pAge']       = features(x_train,x_train[['Age', 'pCTR']],'Age','mean'
)

# Test Features

x_test['mAdURL']     = features(x_test,x_train[['AdURL', 'CTR']],'AdURL','mea
n')
x_test['mAdId']      = features(x_test,x_train[['AdId', 'CTR']],'AdId','mean')
x_test['mAdvId']     = features(x_test,x_train[['AdvId', 'CTR']],'AdvId','mea
n')
x_test['mDepth']     = features(x_test,x_train[['Depth', 'CTR']],'Depth','mea
n')
x_test['mPos']       = features(x_test,x_train[['Pos', 'CTR']],'Pos','mean')
x_test['mQId']       = features(x_test,x_train[['QId', 'CTR']],'QId','mean')
x_test['mKeyId']     = features(x_test,x_train[['KeyId', 'CTR']],'KeyId','mea
n')
x_test['mTitleId']   = features(x_test,x_train[['TitleId', 'CTR']],'TitleId',
'mean')
x_test['mDescId']    = features(x_test,x_train[['DescId', 'CTR']],'DescId','me
an')
x_test['mUId']       = features(x_test,x_train[['UId', 'CTR']],'UId','mean')
x_test['mRPosition'] = features(x_test,x_train[['RPosition', 'CTR']],'RPositio
n','mean')
x_test['mGender']    = features(x_test,x_train[['Gender', 'CTR']],'Gender','me
an')
x_test['mAge']       = features(x_test,x_train[['Age', 'CTR']],'Age','mean')

x_test['pAdURL']     = features(x_test,x_train[['AdURL', 'pCTR']],'AdURL','mea
n')
x_test['pAdId']      = features(x_test,x_train[['AdId', 'pCTR']],'AdId','mean'
)
x_test['pAdvId']     = features(x_test,x_train[['AdvId', 'pCTR']],'AdvId','mea
n')
x_test['pDepth']     = features(x_test,x_train[['Depth', 'pCTR']],'Depth','mea
n')
x_test['pPos']       = features(x_test,x_train[['Pos', 'pCTR']],'Pos','mean')
x_test['pQId']       = features(x_test,x_train[['QId', 'pCTR']],'QId','mean')
x_test['pKeyId']     = features(x_test,x_train[['KeyId', 'pCTR']],'KeyId','mea
n')
x_test['pTitleId']   = features(x_test,x_train[['TitleId', 'pCTR']],'TitleId',
'mean')
x_test['pDescId']    = features(x_test,x_train[['DescId', 'pCTR']],'DescId','m
ean')
x_test['pUId']       = features(x_test,x_train[['UId', 'pCTR']],'UId','mean')
x_test['pRPosition'] = features(x_test,x_train[['RPosition', 'pCTR']],'RPositi
on','mean')
x_test['pGender']    = features(x_test,x_train[['Gender', 'pCTR']],'Gender','m
ean')
x_test['pAge']       = features(x_test,x_train[['Age', 'pCTR']],'Age','mean')

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:00:33.596918
```

In [58]:
```
# only the above features gave the better results when I modelled. So saving these features for future purpose

x_train.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/train_10L_basic_feat.csv',index=False)
x_test.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/test_10L_basic_feat.csv',index=False)
```

In [74]:
```
# Reading from CSV file

x_train = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/train_10L_basic_feat.csv")
x_test = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/test_10L_basic_feat.csv")

#total 49 features
```

In [75]:
```
x_train.head()
```

Out[75]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	35803	0.0	1.0	14340390157469405184	4803006	23777	1	1	229
1	35805	0.0	2.0	14340390157469405184	4803006	23777	2	1	229
2	920342	0.0	1.0	14340390157469405184	4803006	23777	3	1	229
3	78016	0.0	1.0	14340390157469405184	4803006	23777	2	1	229
4	831444	0.0	1.0	14340390157469405184	4803006	23777	2	1	229

5 rows × 49 columns

In [76]:
```
x_test.head()
```

Out[76]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	683475	0.0	1.0	15349556856043354112	22098439	36855	3	3	128
1	969618	0.0	1.0	15349556856043354112	22098439	36855	3	2	128
2	683473	0.0	1.0	15349556856043354112	22098439	36855	2	2	528
3	616976	0.0	1.0	15349556856043354112	22098457	36855	2	1	333
4	714143	0.0	1.0	15349556856043354112	22098457	36855	1	1	333

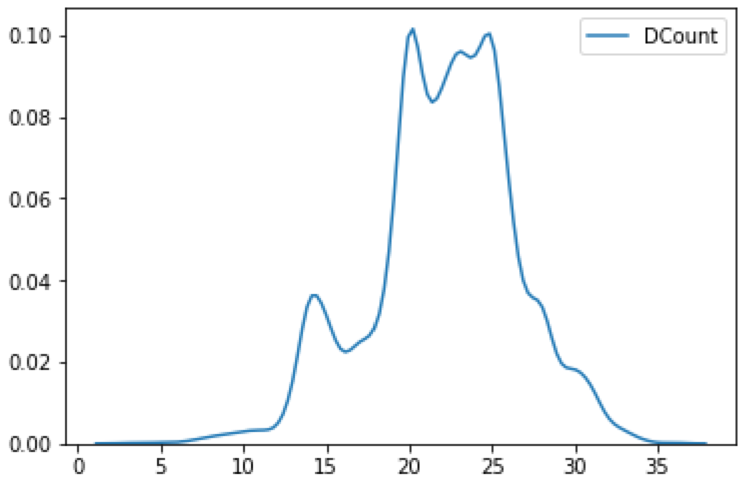
5 rows × 49 columns

In [77]:
```
temp_data_1 = x_train.loc[x_train['class_label']==1]

print( 'Maximum Length of a Desc: ', temp_data_1['DCount'].max())
print( 'Average Length of a Desc: ', temp_data_1['DCount'].mean())
print( 'Average Length of a Desc: ', temp_data_1['DCount'].min())
```

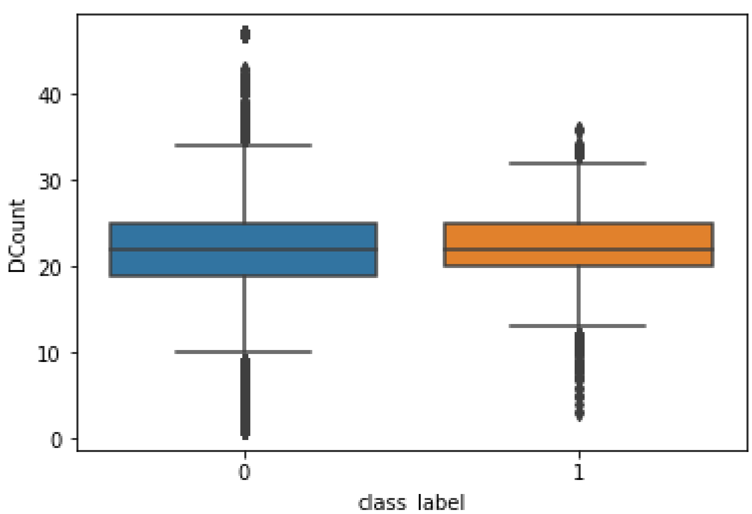
```
Maximum Length of a Desc:  36
Average Length of a Desc:  22.10348236844996
Average Length of a Desc:  3
```

In [78]:
```
sns.kdeplot(temp_data_1['DCount'])
```

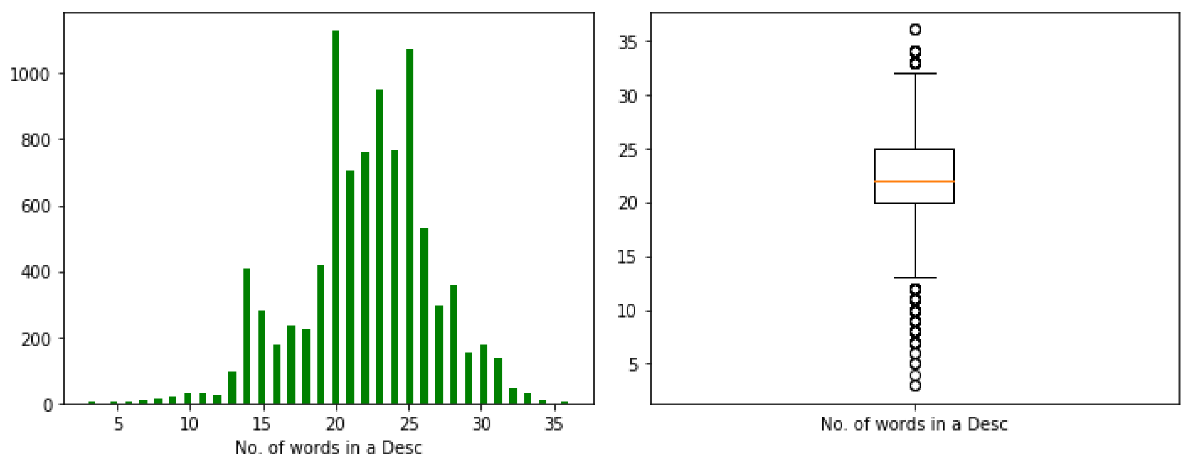
Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x22049d01f28>



In [80]:
```
sns.boxplot(x='class_label', y='DCount', data=x_train)
plt.show()
```



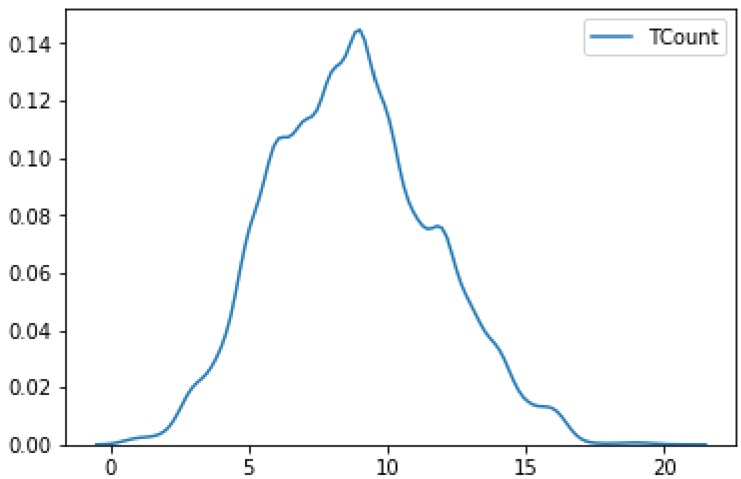
In [81]:
```
# This is the same plot as above just a little more readable ...

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

plt.subplot(1, 2, 1)
plt.hist(temp_data_1['DCount'],
         color='green',
         bins=65,
         normed=False)
plt.xlabel('No. of words in a Desc')

plt.subplot(1, 2, 2)
plt.boxplot(temp_data_1['DCount'],
            labels=['No. of words in a Desc'],
            )

plt.tight_layout()
```



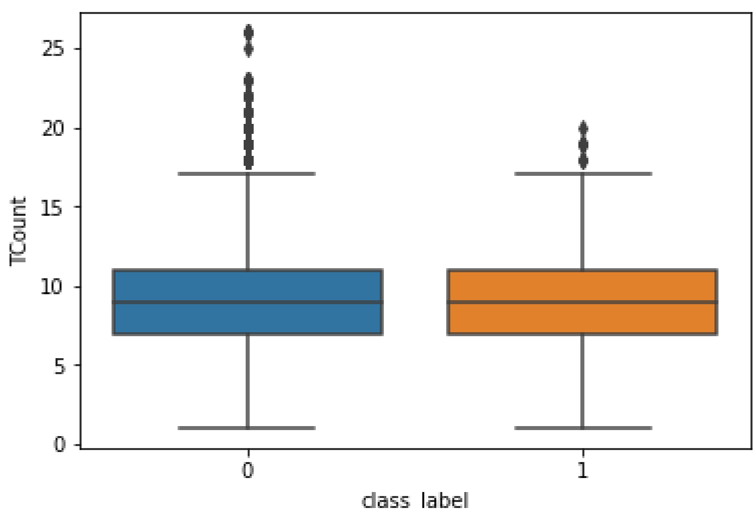
In [82]:
```
print( 'Maximum Length of a Title: ', temp_data_1['TCount'].max())
print( 'Average Length of a Title: ', temp_data_1['TCount'].mean())
print( 'Average Length of a Title: ', temp_data_1['TCount'].min())

sns.kdeplot(temp_data_1['TCount'])
```

```
Maximum Length of a Title:  20
Average Length of a Title:  8.765022520048335
Average Length of a Title:  1
```

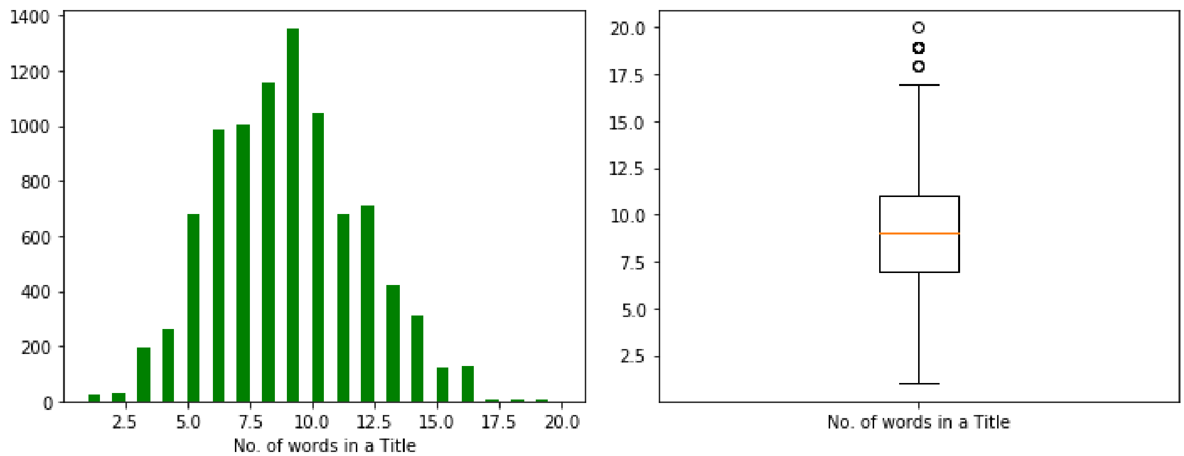
Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x22046ee82b0>



In [84]:
```
sns.boxplot(x='class_label', y='TCount', data=x_train)
plt.show()
```



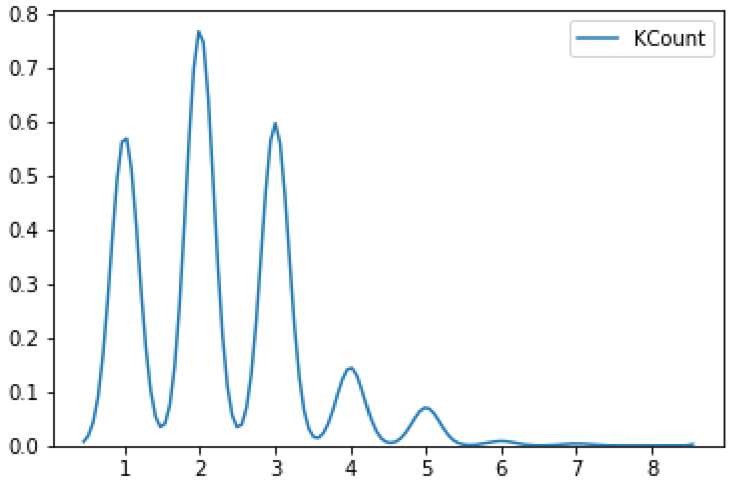
In [85]:
```
# This is the same plot as above just a little more readable ...

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

plt.subplot(1, 2, 1)
plt.hist(temp_data_1['TCount'],
         color='green',
         bins=38,
         normed=False)
plt.xlabel('No. of words in a Title')

plt.subplot(1, 2, 2)
plt.boxplot(temp_data_1['TCount'],
            labels=['No. of words in a Title'],
            )

plt.tight_layout()
```



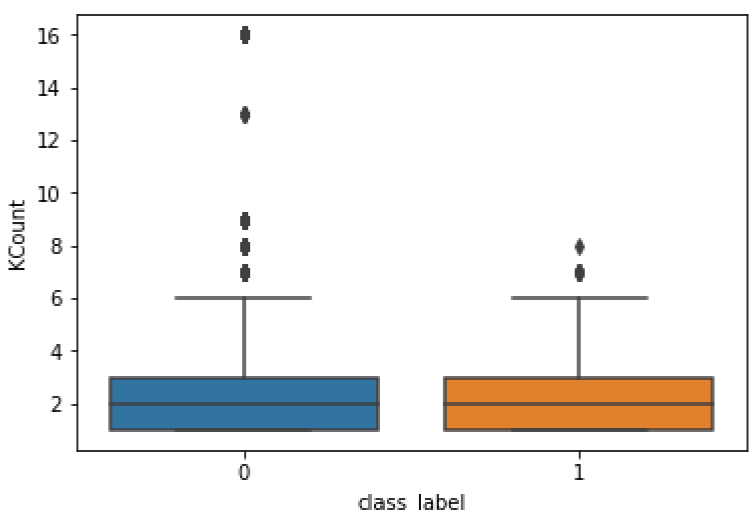
In [86]:
```
print( 'Maximum Length of a Keyword: ', temp_data_1['KCount'].max())
print( 'Average Length of a Keyword: ', temp_data_1['KCount'].mean())
print( 'Average Length of a Keyword: ', temp_data_1['KCount'].min())

sns.kdeplot(temp_data_1['KCount'])
```

```
Maximum Length of a Keyword:  8
Average Length of a Keyword:  2.263649346369329
Average Length of a Keyword:  1
```

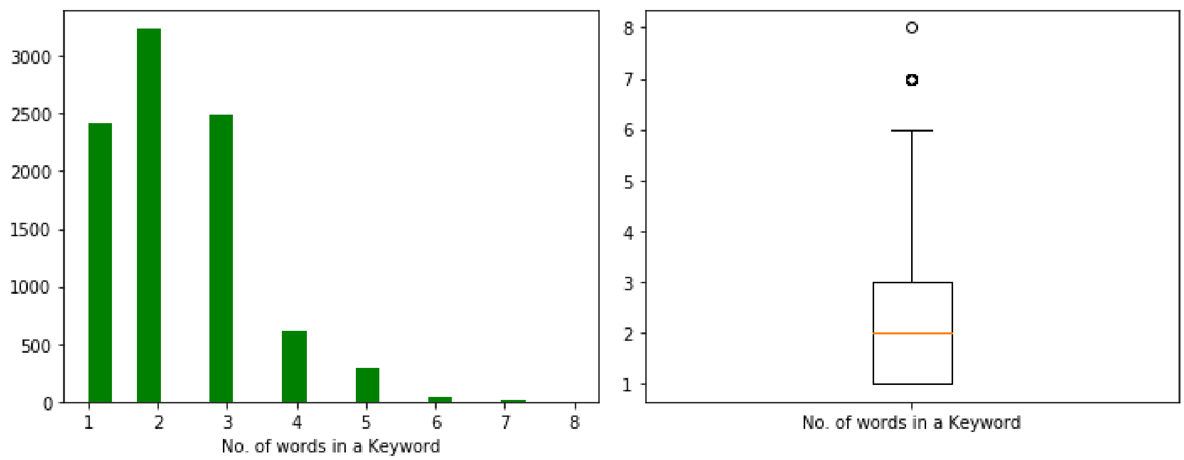
Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x22043749d68>



In [87]:
```
sns.boxplot(x='class_label', y='KCount', data=x_train)
plt.show()
```



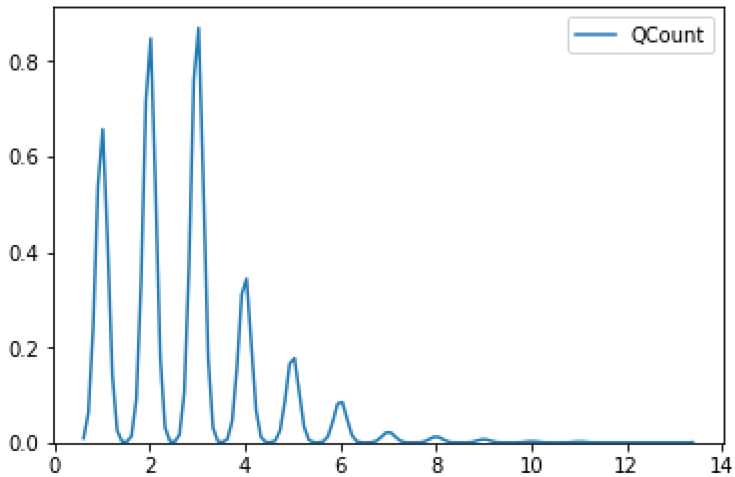
In [88]:
```
# This is the same plot as above just a little more readable ...

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

plt.subplot(1, 2, 1)
plt.hist(temp_data_1['KCount'],
         color='green',
         bins=20,
         normed=False)
plt.xlabel('No. of words in a Keyword')

plt.subplot(1, 2, 2)
plt.boxplot(temp_data_1['KCount'],
            labels=['No. of words in a Keyword'],
            )

plt.tight_layout()
```



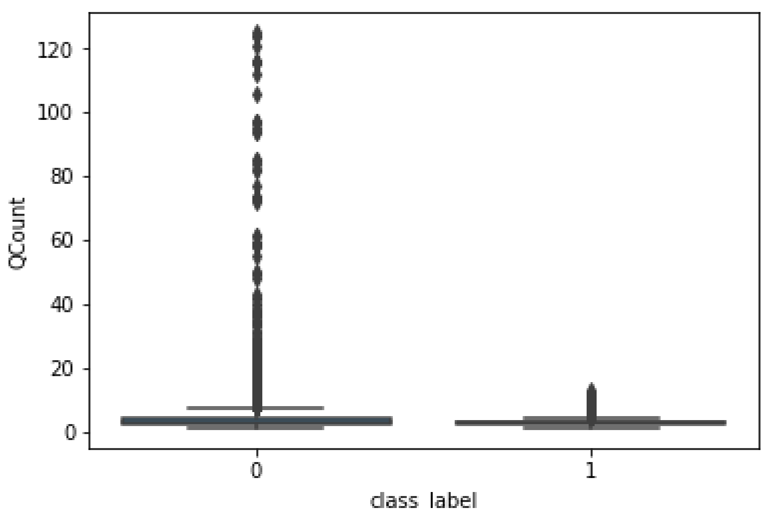
In [89]:
```
print( 'Maximum Length of a Query: ', temp_data_1['QCount'].max())
print( 'Average Length of a Query: ', temp_data_1['QCount'].mean())
print( 'Minimum Length of a Query: ', temp_data_1['QCount'].min())

sns.kdeplot(temp_data_1['QCount'])
```

```
Maximum Length of a Query:  13
Average Length of a Query:  2.7037240470174666
Minimum Length of a Query:  1
```

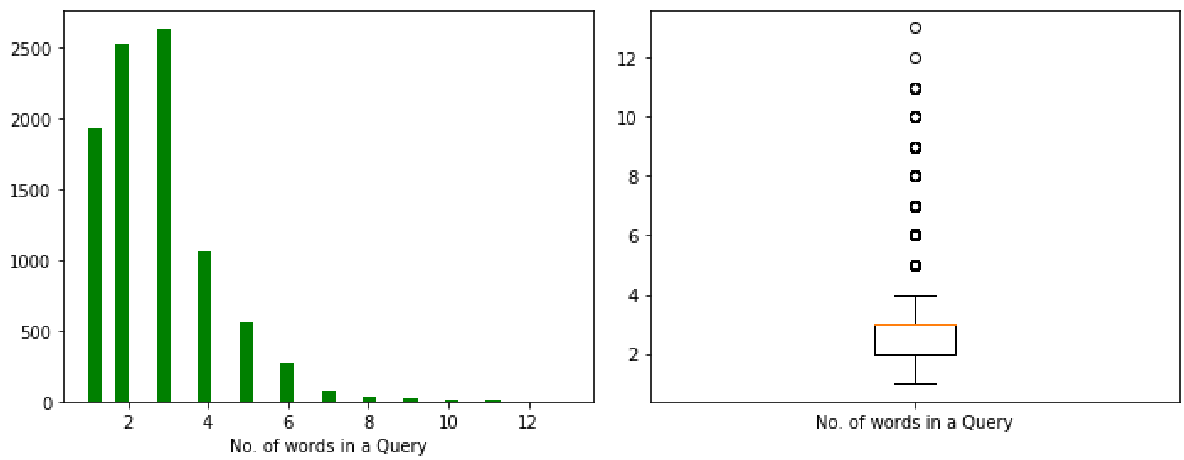
Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x21fd030a6d8>



In [91]:
```
sns.boxplot(x='class_label', y='QCount', data=x_train)
plt.show()
```



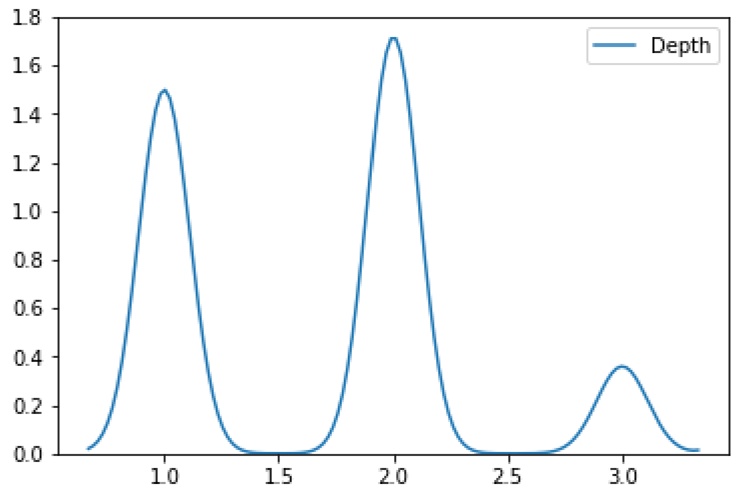
In [92]:
```
# This is the same plot as above just a little more readable ...

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

plt.subplot(1, 2, 1)
plt.hist(temp_data_1['QCount'],
         color='green',
         bins=35,
         normed=False)
plt.xlabel('No. of words in a Query')

plt.subplot(1, 2, 2)
plt.boxplot(temp_data_1['QCount'],
            labels=['No. of words in a Query'],
            )

plt.tight_layout()
```



In [93]:
```
print( 'Maximum Depth: ', temp_data_1['Depth'].max())
print( 'Minimum Depth: ', temp_data_1['Depth'].min())

sns.kdeplot(temp_data_1['Depth'])
```

```
Maximum Depth:  3
Minimum Depth:  1
```

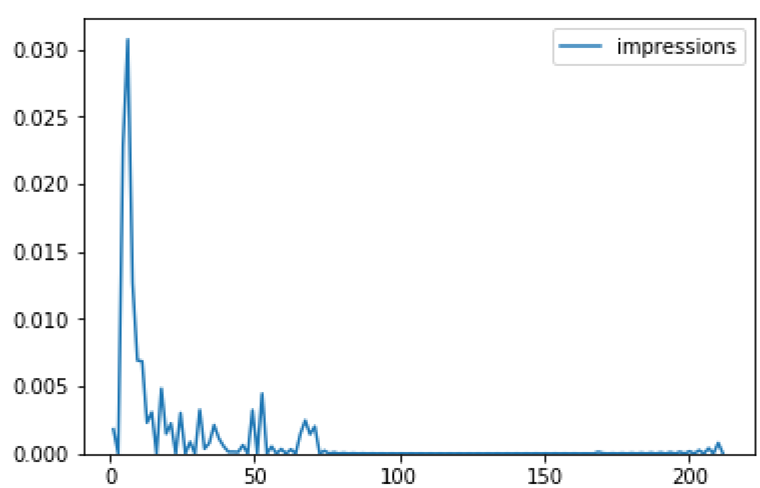
Out[93]: <matplotlib.axes._subplots.AxesSubplot at 0x2200c56ac88>



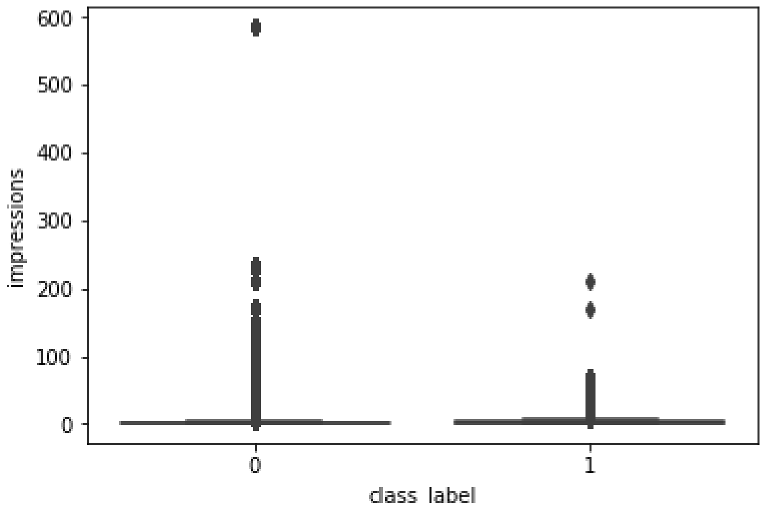
In [94]:
```
print( 'Maximum impressions: ', temp_data_1['impressions'].max())
print( 'Average impressions: ', temp_data_1['impressions'].mean())
print( 'Median impressions: ', temp_data_1['impressions'].median())
print( 'Minimum impressions: ', temp_data_1['impressions'].min())

sns.kdeplot(temp_data_1['impressions'])
```

```
Maximum impressions:  211.0
Average impressions:  5.307810611886191
Median impressions:  2.0
Minimum impressions:  2.0
```

Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x220950715f8>



In [96]:
```
sns.boxplot(x='class_label', y='impressions', data=x_train)
plt.show()
```



Tried some more feature engineering

In [31]:

```
# Here we compute total #impressions for each AdId, AdvId, Depth, Pos, Rpositi
on

start = datetime.now()

x_train['num_Imp_Ad']         = features(x_train,x_train[['AdId', 'impression
s']],'AdId','sum')
x_train['num_Imp_Advertiser'] = features(x_train,x_train[['AdvId', 'impression
s']],'AdvId','sum')
x_train['num_Imp_Depth']      = features(x_train,x_train[['Depth', 'impression
s']],'Depth','sum')
x_train['num_Imp_Position']   = features(x_train,x_train[['Pos', 'impressions'
]],'Pos','sum')
x_train['num_Imp_Rposition '] = features(x_train,x_train[['RPosition', 'impres
sions']],'RPosition','sum')
x_train['num_Imp_UId']      = features(x_train,x_train[['UId', 'impressions'
]],'UId','sum')

x_test['num_Imp_Ad']         = features(x_test,x_train[['AdId', 'impressions'
]],'AdId','sum')
x_test['num_Imp_Advertiser'] = features(x_test,x_train[['AdvId', 'impressions'
]],'AdvId','sum')
x_test['num_Imp_Depth']      = features(x_test,x_train[['Depth', 'impressions'
]],'Depth','sum')
x_test['num_Imp_Position']   = features(x_test,x_train[['Pos', 'impressions'
]],'Pos','sum')
x_test['num_Imp_Rposition '] = features(x_test,x_train[['RPosition', 'impressi
ons']],'RPosition','sum')
x_test['num_Imp_UId']      = features(x_test,x_train[['UId', 'impressions']],
'UId','sum')

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:00:05.622804
```

In [4]:
```
# Here we compute total #impressions for each AdId, AdvId, Depth, Pos, Rpositi
on

start = datetime.now()

x_train['num_Imp_count_Ad']         = features(x_train,x_train[['AdId', 'impre
ssions']],'AdId','count')
x_train['num_Imp_count_Advertiser'] = features(x_train,x_train[['AdvId', 'impr
essions']],'AdvId','count')
#x_train['num_Imp_count_Depth']      = features(x_train,x_train[['Depth', 'imp
ressions']],'Depth','count')
#x_train['num_Imp_count_Position']   = features(x_train,x_train[['Pos', 'impre
ssions']],'Pos','count')
#x_train['num_Imp_count_Rposition '] = features(x_train,x_train[['RPosition',
 'impressions']],'RPosition','count')
x_train['num_Imp_count_QId']        = features(x_train,x_train[['QId', 'impres
sions']],'QId','count')
x_train['num_Imp_count_KeyId']      = features(x_train,x_train[['KeyId', 'impr
essions']],'KeyId','count')
x_train['num_Imp_count_TitleId ']   = features(x_train,x_train[['TitleId', 'im
pressions']],'TitleId','count')
x_train['num_Imp_count_DescId']     = features(x_train,x_train[['DescId', 'imp
ressions']],'DescId','count')
x_train['num_Imp_count_UId']        = features(x_train,x_train[['UId', 'impres
sions']],'UId','count')


x_test['num_Imp_count_Ad']         = features(x_test,x_train[['AdId', 'impress
ions']],'AdId','count')
x_test['num_Imp_count_Advertiser'] = features(x_test,x_train[['AdvId', 'impres
sions']],'AdvId','count')
#x_test['num_Imp_count_Depth']      = features(x_test,x_train[['Depth', 'impre
ssions']],'Depth','count')
#x_test['num_Imp_count_Position']   = features(x_test,x_train[['Pos', 'impress
ions']],'Pos','count')
#x_test['num_Imp_count_Rposition '] = features(x_test,x_train[['RPosition', 'i
mpressions']],'RPosition','count')
x_test['num_Imp_count_QId']        = features(x_test,x_train[['QId', 'impressi
ons']],'QId','count')
x_test['num_Imp_count_KeyId']      = features(x_test,x_train[['KeyId', 'impres
sions']],'KeyId','count')
x_test['num_Imp_count_TitleId ']   = features(x_test,x_train[['TitleId', 'impr
essions']],'TitleId','count')
x_test['num_Imp_count_DescId']     = features(x_test,x_train[['DescId', 'impre
ssions']],'DescId','count')
x_test['num_Imp_count_UId']        = features(x_test,x_train[['UId', 'impressi
ons']],'UId','count')

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:00:10.010451
```

In [17]:
```
# Here we compute total #clicks for each AdId, AdvId, QId, KeyId, TitleId, Des
cId, UId, Gender

start = datetime.now()

x_train['num_click_Ad']         = features(x_train,x_train[['AdId', 'clicks'
]],'AdId','sum')
x_train['num_click_Advertiser'] = features(x_train,x_train[['AdvId', 'clicks'
]],'AdvId','sum')
x_train['num_click_QId']        = features(x_train,x_train[['QId', 'clicks']],
'QId','sum')
x_train['num_click_KeyId']      = features(x_train,x_train[['KeyId', 'clicks'
]],'KeyId','sum')
x_train['num_click_TitleId ']   = features(x_train,x_train[['TitleId', 'click
s']],'TitleId','sum')
x_train['num_click_DescId']     = features(x_train,x_train[['DescId', 'clicks'
]],'DescId','sum')
x_train['num_click_UId']        = features(x_train,x_train[['UId', 'clicks']],
'UId','sum')
x_train['num_click_Gender ']    = features(x_train,x_train[['Gender', 'clicks'
]],'Gender','sum')

x_test['num_click_Ad']          = features(x_test,x_train[['AdId', 'clicks']],
'AdId','sum')
x_test['num_click_Advertiser']  = features(x_test,x_train[['AdvId', 'clicks'
]],'AdvId','sum')
x_test['num_click_QId']         = features(x_test,x_train[['QId', 'clicks']],
'QId','sum')
x_test['num_click_KeyId']       = features(x_test,x_train[['KeyId', 'clicks'
]],'KeyId','sum')
x_test['num_click_TitleId ']    = features(x_test,x_train[['TitleId', 'clicks'
]],'TitleId','sum')
x_test['num_click_DescId']      = features(x_test,x_train[['DescId', 'clicks'
]],'DescId','sum')
x_test['num_click_UId']         = features(x_test,x_train[['UId', 'clicks']],
'UId','sum')
x_test['num_click_Gender ']     = features(x_test,x_train[['Gender', 'clicks'
]],'Gender','sum')

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:00:36.345268
```

In [18]:
```
start = datetime.now()

x_train['num_count_Ad']         = features(x_train,x_train[['AdId', 'clicks'
]],'AdId','count')
x_train['num_count_Advertiser'] = features(x_train,x_train[['AdvId', 'clicks'
]],'AdvId','count')
x_train['num_count_QId']        = features(x_train,x_train[['QId', 'clicks']],
'QId','count')
x_train['num_count_KeyId']      = features(x_train,x_train[['KeyId', 'clicks'
]],'KeyId','count')
x_train['num_count_TitleId ']   = features(x_train,x_train[['TitleId', 'click
s']],'TitleId','count')
x_train['num_count_DescId']     = features(x_train,x_train[['DescId', 'clicks'
]],'DescId','count')
x_train['num_count_UId']        = features(x_train,x_train[['UId', 'clicks']],
'UId','count')

x_test['num_count_Ad']         = features(x_test,x_train[['AdId', 'clicks']],
'AdId','count')
x_test['num_count_Advertiser'] = features(x_test,x_train[['AdvId', 'clicks']],
'AdvId','count')
x_test['num_count_QId']        = features(x_test,x_train[['QId', 'clicks']],'Q
Id','count')
x_test['num_count_KeyId']      = features(x_test,x_train[['KeyId', 'clicks']],
'KeyId','count')
x_test['num_count_TitleId ']   = features(x_test,x_train[['TitleId', 'clicks'
]],'TitleId','count')
x_test['num_count_DescId']     = features(x_test,x_train[['DescId', 'clicks'
]],'DescId','count')
x_test['num_count_UId']        = features(x_test,x_train[['UId', 'clicks']],'U
Id','count')


end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:00:28.156833
```

In [32]:
```
x_train.head()
```

Out[32]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	756057	0.0	1.0	7797031665819164672	20563185	5476	2	1	221
1	208918	0.0	1.0	7797031665819164672	20563185	5476	2	1	224
2	981049	0.0	1.0	7797031665819164672	20563185	5476	2	1	738
3	890103	0.0	1.0	7797031665819164672	20563185	5476	2	1	738
4	208919	0.0	1.0	6311739615958990848	21926844	38029	2	2	224

5 rows × 81 columns

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

Out[33]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	QId
0	933760	0.0	1.0	4151297533975027200	21437320	36604	2	2	211
1	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211
2	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211
3	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211
4	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211

5 rows × 81 columns

Weighted Features -> where each token is weighted by their idf value

In [21]:
```
start = datetime.now()

# Load Keyword Data.

key_col  = ['KeyId', 'Keyword']
keyword  = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012
_Track 2/purchasedkeywordid_tokensid.txt', sep='\t', header=None, names=key_co
l)

# one-hot encoding of Keyword feature.
key_tfidf_vectorizer = TfidfVectorizer(use_idf=True)
tfidf_keyword = key_tfidf_vectorizer.fit_transform(keyword['Keyword'])

a = list(key_tfidf_vectorizer.vocabulary_.keys())
b = list(key_tfidf_vectorizer.idf_)
#print(len(a)) -> 91482
#print(len(b)) -> 91482
dict_keyword = dict(zip(a[::], b[::]))


# Load Query Data..

query_col = ['QId', 'Query']
query     = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/queryid_tokensid.txt', sep='\t', header=None, names=query_col)

query_tfidf_vectorizer = TfidfVectorizer(use_idf=True)
tfidf_query = query_tfidf_vectorizer.fit_transform(query['Query'])

a = list(query_tfidf_vectorizer.vocabulary_.keys())
b = list(query_tfidf_vectorizer.idf_)
dict_query = dict(zip(a[::], b[::]))

# Load Ad Description Data..

desc_col  = ['DescId', 'Description']
desc      = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/descriptionid_tokensid.txt', sep='\t', header=None, names=desc_col)

desc_tfidf_vectorizer = TfidfVectorizer(use_idf=True)
tfidf_desc = desc_tfidf_vectorizer.fit_transform(desc['Description'])

a = list(desc_tfidf_vectorizer.vocabulary_.keys())
b = list(desc_tfidf_vectorizer.idf_)
dict_desc = dict(zip(a[::], b[::]))

# Load Ad Title Data..

title_col = ['TitleId', 'Title']
title     = pd.read_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_201
2_Track 2/titleid_tokensid.txt', sep='\t', header=None, names=title_col)

title_tfidf_vectorizer = TfidfVectorizer(use_idf=True)
tfidf_title = title_tfidf_vectorizer.fit_transform(title['Title'])

a = list(title_tfidf_vectorizer.vocabulary_.keys())
```

```
b = list(title_tfidf_vectorizer.idf_)
dict_title = dict(zip(a[::], b[::]))

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:27:43.512583
```

In [22]:
```
start = datetime.now()

def idf_sum(sentence,res):
    sum = 0.0
    store = str(sentence).split('|')
    for i in range(len(store)):
        val = res.get(store[i])
        if val is None:
            val = 0.0
        else:
            sum = sum+val
    return sum

keyword['num_idf_Keyword'] = keyword['Keyword'].apply((lambda x: idf_sum(x,dic
t_keyword)))
#del keyword['Keyword']

query['num_idf_query'] = query['Query'].apply((lambda x: idf_sum(x,dict_query
)))
#del query['Query']

desc['num_idf_desc'] = desc['Description'].apply((lambda x: idf_sum(x,dict_des
c)))
#del desc['Description']

title['num_idf_title'] = title['Title'].apply((lambda x: idf_sum(x,dict_title
)))
#del title['Title']

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:14:43.339205
```

In [23]:
```
del keyword['Keyword']
del query['Query']
del desc['Description']
del title['Title']
```

In [24]:
```
# Merging data with user, query, title, keyword & desc on appropriate keys to
 get data..

x_train  = pd.merge(x_train, query, on='QId')
x_train  = pd.merge(x_train, title, on='TitleId')
x_train  = pd.merge(x_train, desc,  on='DescId')
x_train  = pd.merge(x_train, keyword, on='KeyId')

x_train.head()
```

Out[24]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	756057	0.0	1.0	7797031665819164672	20563185	5476	2	1	221
1	208918	0.0	1.0	7797031665819164672	20563185	5476	2	1	224
2	981049	0.0	1.0	7797031665819164672	20563185	5476	2	1	738
3	890103	0.0	1.0	7797031665819164672	20563185	5476	2	1	738
4	208919	0.0	1.0	6311739615958990848	21926844	38029	2	2	224

5 rows × 73 columns

In [25]:
```
# Merging data with user, query, title, keyword & desc on appropriate keys to
 get data..

x_test  = pd.merge(x_test, query, on='QId')
x_test  = pd.merge(x_test, title, on='TitleId')
x_test  = pd.merge(x_test, desc,  on='DescId')
x_test  = pd.merge(x_test, keyword, on='KeyId')

x_test.head()
```

Out[25]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	QId
0	933760	0.0	1.0	4151297533975027200	21437320	36604	2	2	211
1	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211
2	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211
3	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211
4	607979	0.0	210.0	4151297533975027200	21437320	36604	2	2	211

5 rows × 73 columns

In [26]:
```
# Saving to CSV file

x_train.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track
 2/train_10L_num_feats_mod.csv',index=False)
x_test.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/
test_10L_num_feats_mod.csv',index=False)
```

In [35]:
```
x_train.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track
 2/train_10L_num_feats_mod_1.csv',index=False)
x_test.to_csv('C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_Track 2/
test_10L_num_feats_mod_1.csv',index=False)
```

In [30]:
```
# Reading from CSV file

x_train = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_
Track 2/train_10L_num_feats_mod_1.csv")
x_test = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_T
rack 2/test_10L_num_feats_mod_1.csv")

#x_train.head() -> total 81 features.
```

In [62]:
```
# Reading from CSV file

x_train = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_
Track 2/train_10L_basic_feat.csv")
x_test = pd.read_csv("C:/Users/Administrator/Documents/Datasets/KDD_Cup_2012_T
rack 2/test_10L_basic_feat.csv")

#total 49 features
```

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

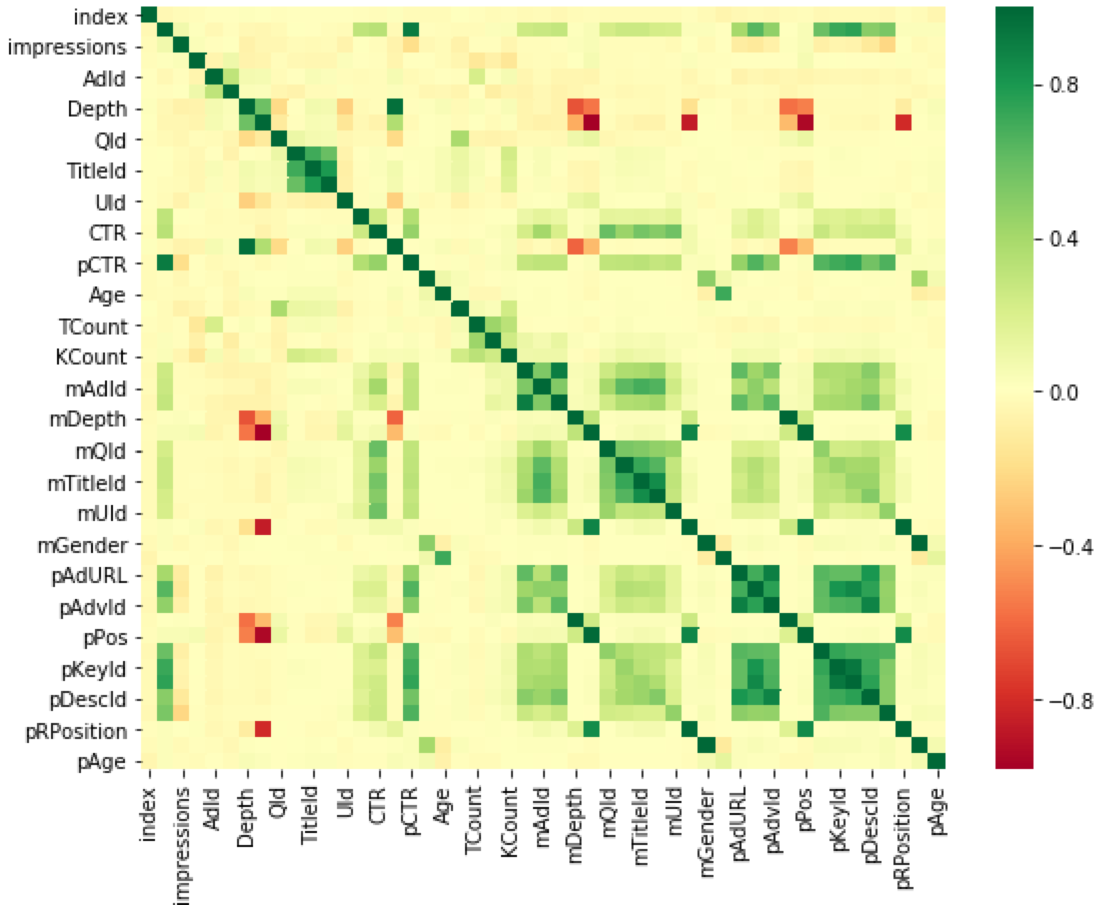
Out[63]:

	index	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	
0	35803	0.0	1.0	14340390157469405184	4803006	23777	1	1	229
1	35805	0.0	2.0	14340390157469405184	4803006	23777	2	1	229
2	920342	0.0	1.0	14340390157469405184	4803006	23777	3	1	229
3	78016	0.0	1.0	14340390157469405184	4803006	23777	2	1	229
4	831444	0.0	1.0	14340390157469405184	4803006	23777	2	1	229

5 rows × 49 columns

CORRELATION CHECK

In [65]:
```
# Heatmap of correlation plot
plt.figure(figsize=(9,7))
sns.heatmap(x_train.corr(), cmap='RdYlGn')
plt.show()
```



In [66]:
```
y_train = x_train['class_label'].values
y_test = x_test['class_label'].values
```

In [67]:
```
# Dropping off the class labels and index columns

x_train = x_train.drop('class_label', axis=1)
x_train = x_train.drop('index', axis=1)

x_test = x_test.drop('class_label', axis=1)
x_test = x_test.drop('index', axis=1)

print(x_train.shape)
print(x_test.shape)
```

```
(981617, 47)
(245352, 47)
```

Binary Sparse features

In [29]:
```
# We also expand query’s tokens, title’s tokens, description’s tokens and keyw
ord’s tokens into binary features.
# That is, if a token occurs in title, query, description or keyword, the corr
esponding value in the feature vector will be
# 1, or 0 otherwise

start = datetime.now()


# one-hot encoding of Keyword feature.
train = [str (item) for item in x_train['KeyId']]
test = [str (item) for item in x_test['KeyId']]
keyword_vectorizer = CountVectorizer(binary=True)
bow_train_keyword = keyword_vectorizer.fit_transform(train)
bow_test_keyword = keyword_vectorizer.transform(test)

#train = [str (item) for item in x_train['Gender']]
#test = [str (item) for item in x_test['Gender']]
#user_gender_vectorizer = CountVectorizer(binary=True)
#bow_train_gender = user_gender_vectorizer.fit_transform(train)
#bow_test_gender = user_gender_vectorizer.transform(test)

#train = [str (item) for item in x_train['Age']]
#test = [str (item) for item in x_test['Age']]
#user_age_vectorizer = CountVectorizer(binary=True)
#bow_train_age = user_age_vectorizer.fit_transform(train)
#bow_test_age = user_age_vectorizer.transform(test)

train = [str (item) for item in x_train['QId']]
test = [str (item) for item in x_test['QId']]
query_vectorizer = CountVectorizer(binary=True)
bow_train_query = query_vectorizer.fit_transform(train)
bow_test_query = query_vectorizer.transform(test)

train = [str (item) for item in x_train['DescId']]
test = [str (item) for item in x_test['DescId']]
desc_vectorizer = CountVectorizer(binary=True)
bow_train_desc = desc_vectorizer.fit_transform(train)
bow_test_desc = desc_vectorizer.transform(test)

train = [str (item) for item in x_train['TitleId']]
test = [str (item) for item in x_test['TitleId']]
title_vectorizer = CountVectorizer(binary=True)
bow_train_title = title_vectorizer.fit_transform(train)
bow_test_title = title_vectorizer.transform(test)

train = [str (item) for item in x_train['AdId']]
test = [str (item) for item in x_test['AdId']]
add_vectorizer = CountVectorizer(binary=True)
bow_train_add = add_vectorizer.fit_transform(train)
bow_test_add = add_vectorizer.transform(test)

train = [str (item) for item in x_train['AdvId']]
test = [str (item) for item in x_test['AdvId']]
adv_vectorizer = CountVectorizer(binary=True)
bow_train_adv = adv_vectorizer.fit_transform(train)
```

```
bow_test_adv = adv_vectorizer.transform(test)

train = [str (item) for item in x_train['AdURL']]
test = [str (item) for item in x_test['AdURL']]
adurl_vectorizer = CountVectorizer(binary=True)
bow_train_adurl = adurl_vectorizer.fit_transform(train)
bow_test_adurl = adurl_vectorizer.transform(test)

end = datetime.now()
print("Time taken to run this cell: ",end-start)
```

```
Time taken to run this cell:  0:02:22.711414
```

In [30]:

```
x_train_sparse = csr_matrix(x_train.values)
train_data = hstack((x_train_sparse,bow_train_keyword,bow_train_query,bow_trai
n_desc,bow_train_title,bow_train_add,bow_train_adv,bow_train_adurl))

x_test_sparse = csr_matrix(x_test.values)
test_data = hstack((x_test_sparse,bow_test_keyword,bow_test_query,bow_test_des
c,bow_test_title,bow_test_add,bow_test_adv,bow_test_adurl))
```

In [31]:

```
test_data
```

Out[31]:

```
<368123x768756 sparse matrix of type '<class 'numpy.float64'>'
	with 26215324 stored elements in COOrdinate format>
```

In [32]:

```
train_normalized = normalize(train_data, axis=1)
test_normalized = normalize(test_data , axis=1)
```

In [68]:

```
# To plot confusion matrix.
def plot_confusion_matrix(y_test, pred):
    C = confusion_matrix(y_test, pred)

    labels = [1,2]
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(10,4))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, y
ticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

LOGISTIC REGRESSION MODEL

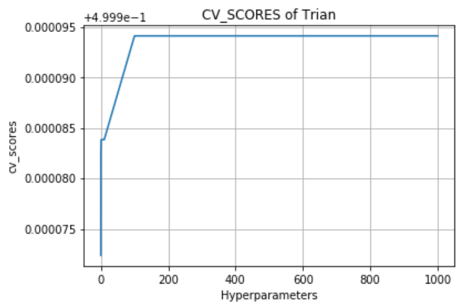
In [34]:
```
# To print scores
def print_metrics_measure(pred,y_test):

     # Plotting Confusion_Matrix
    plot_confusion_matrix(y_test, pred)

    #Score
    score = metrics.roc_auc_score(y_test, pred)*100
    print('\nThe score of the LR [Test data] : ',(score))

    print("----"*20)
```

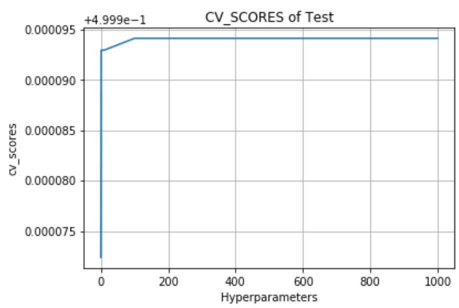
In [35]:
```
def gridsearch(train_std_data,test_std_data,y_1,y_test):

    start = datetime.now()
    cv_scores = []
    tuned_parameters = [{'C': [0.001,0.01,0.1,1,10,100,1000], "penalty":["l1",
"l2"]}]
    k = StratifiedKFold(n_splits=3)
    grid_lr = GridSearchCV(LogisticRegression(), tuned_parameters, scoring =
'roc_auc', cv=k)
    grid_lr.fit(train_std_data,y_1)
    print("Best Estimator: ")
    model = grid_lr.best_estimator_
    print(model)
    print("******"*20)

    C = [0.001,0.01,0.1,1,10,100,1000]

    if grid_lr.best_params_['penalty'] == 'l1':
        cv_scores = (grid_lr.cv_results_['mean_train_score'][::2])
    else:
        cv_scores = (grid_lr.cv_results_['mean_train_score'][1::2])

    plt.plot(C, cv_scores)

    for xy in zip(C, np.round(cv_scores,3)):
        plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

    plt.xlabel('Hyperparameters')
    plt.ylabel('cv_scores')
    plt.title("CV_SCORES of Trian")
    plt.grid()
    plt.show()

    #print("Best Hyperparameter is: ",grid_lr.best_params_['C'])


    if grid_lr.best_params_['penalty'] == 'l1':
        cv_scores = (grid_lr.cv_results_['mean_test_score'][::2])
    else:
        cv_scores = (grid_lr.cv_results_['mean_test_score'][1::2])

    plt.plot(C, cv_scores)

    for xy in zip(C, np.round(cv_scores,3)):
        plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

    plt.xlabel('Hyperparameters')
    plt.ylabel('cv_scores')
    plt.title("CV_SCORES of Test")
    plt.grid()
    plt.show()

    #print("Best Hyperparameter is: ",grid_lr.best_params_['C'])

    print("******"*20)
```

```
    pred = grid_lr.predict(test_std_data)

    print_metrics_measure(pred,y_test)

    end = datetime.now()
    print("Time taken to run this cell: ",end-start)
    return model
```

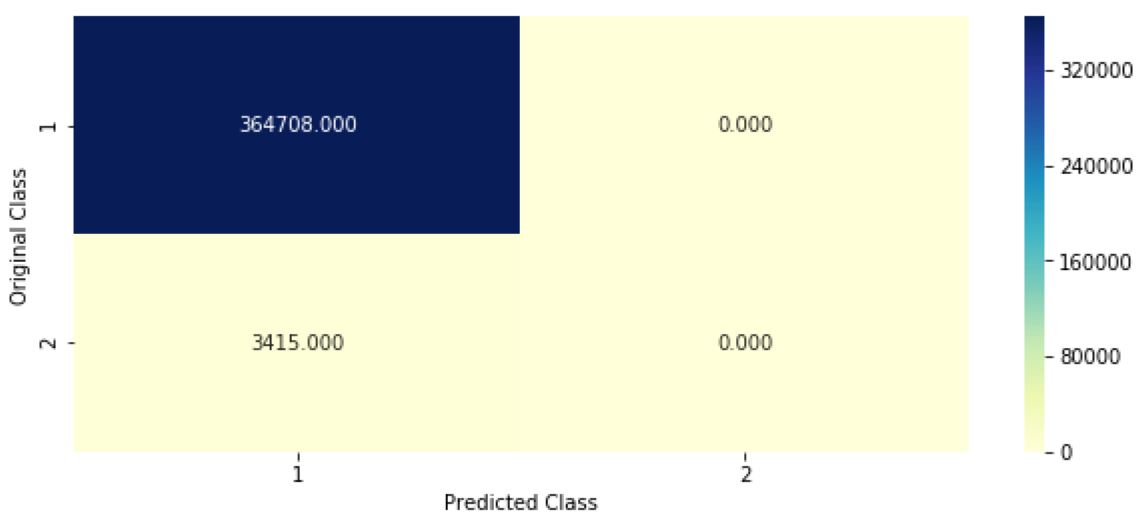
In [36]:
```
lr_grid_bow = gridsearch(train_normalized,test_normalized,y_train,y_test)
```

```
Best Estimator:
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
*****************************************************************************
*******************************************
```






```
*****************************************************************************
*******************************************
-------------------- Confusion matrix --------------------
```




```
The score of the LR [Test data] :  50.0
-----------------------------------------------------------------------------
---
Time taken to run this cell:  0:18:00.569395
```

Observations:

1. We used grid search inorder to tune the hyperparameters(C, penalty).

1. The scorimg metric we used here is auc which is equivalent to the probability that a random pair of a positive sample (clicked ad) and a negative one (unclicked ad) is ranked correctly.

1. We tried to plot the train and test cv scores inorder to understand which value of C is good.

1. With better hyperparameter tuning, the Logistic Regression model scored 50%. From the confusion matrix we can understand that for any query point it is trying to output class 0 (if imbalanced). If we tries to balance the dataset using class_weight=balanced, it is trying to output class 1

The easiest way to successfully generalize a model is by using more data.

The problem is that out-of-the-box classifiers like logistic regression or random forest tend to generalize by discarding the rare class

XGBOOST MODEL

To model, I used only average (click-through rate and pseudo click-through rate) features (47 features). Only using these features gave me better results(74% test auc) than using all the features we engineered above(67% Test auc - tried in different notebook).

In [71]:
```
x_train.columns
```

Out[71]:
```
Index(['clicks', 'impressions', 'AdURL', 'AdId', 'AdvId', 'Depth', 'Pos',
       'QId', 'KeyId', 'TitleId', 'DescId', 'UId', 'CTR', 'RPosition', 'pCT
R',
       'Gender', 'Age', 'QCount', 'TCount', 'DCount', 'KCount', 'mAdURL',
       'mAdId', 'mAdvId', 'mDepth', 'mPos', 'mQId', 'mKeyId', 'mTitleId',
       'mDescId', 'mUId', 'mRPosition', 'mGender', 'mAge', 'pAdURL', 'pAdId',
       'pAdvId', 'pDepth', 'pPos', 'pQId', 'pKeyId', 'pTitleId', 'pDescId',
       'pUId', 'pRPosition', 'pGender', 'pAge'],
      dtype='object')
```

In [69]:
```
x_train.head()
```

Out[69]:

	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	QId	K
0	0.0	1.0	14340390157469405184	4803006	23777	1	1	22997071	1
1	0.0	2.0	14340390157469405184	4803006	23777	2	1	2293	1
2	0.0	1.0	14340390157469405184	4803006	23777	3	1	2293	1
3	0.0	1.0	14340390157469405184	4803006	23777	2	1	2293	1
4	0.0	1.0	14340390157469405184	4803006	23777	2	1	2293	1

5 rows × 47 columns

In [70]:
```
x_test.head()
```

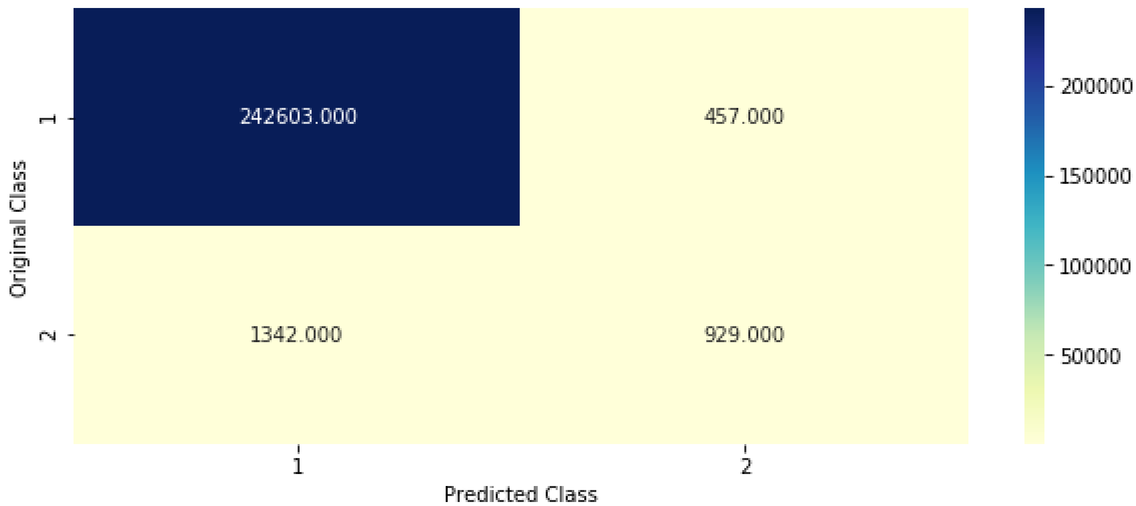
Out[70]:

	clicks	impressions	AdURL	AdId	AdvId	Depth	Pos	QId	Key
0	0.0	1.0	15349556856043354112	22098439	36855	3	3	12875	892
1	0.0	1.0	15349556856043354112	22098439	36855	3	2	12875	892
2	0.0	1.0	15349556856043354112	22098439	36855	2	2	52877	892
3	0.0	1.0	15349556856043354112	22098457	36855	2	1	33318	978
4	0.0	1.0	15349556856043354112	22098457	36855	1	1	33318	978

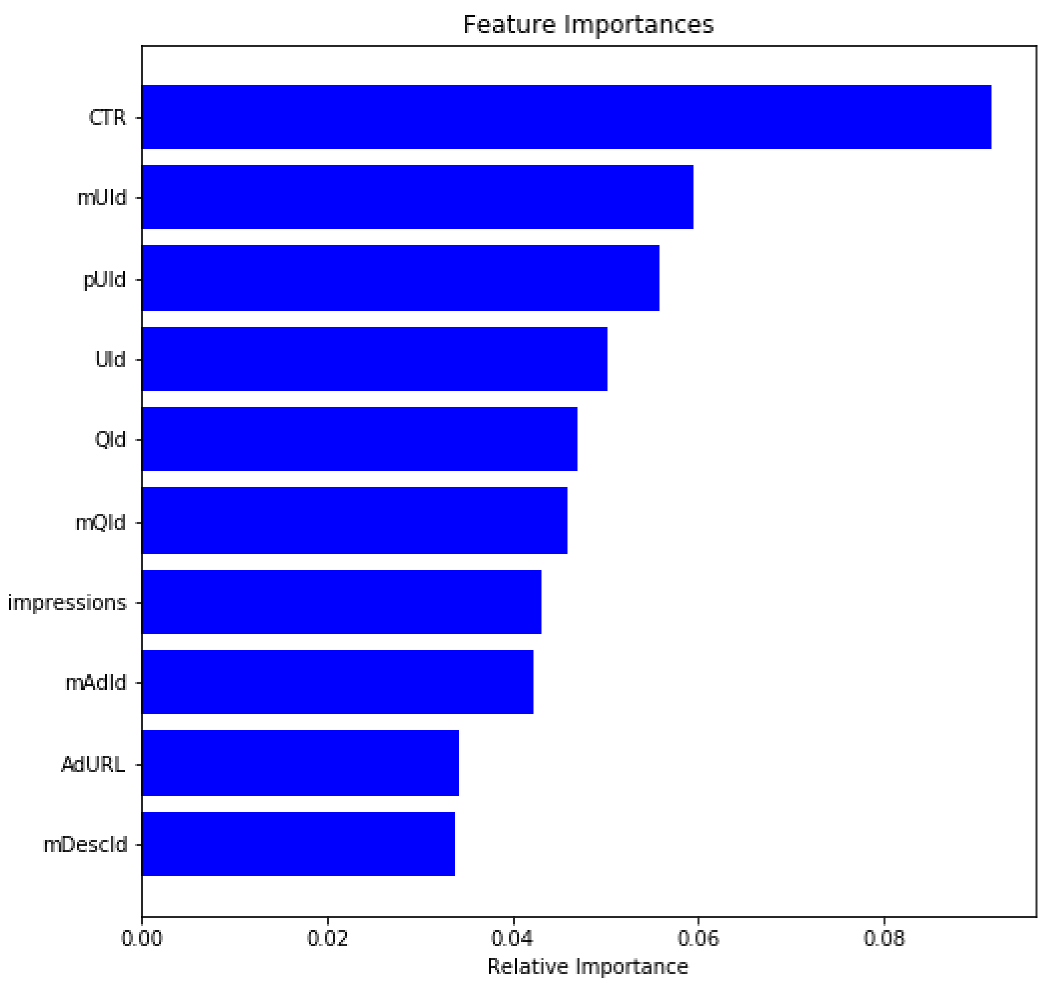
5 rows × 47 columns

In [33]:
```
start = datetime.now()

x_cfl=XGBClassifier(n_estimators=440,max_depth=3,learning_rate=0.132,colsample
_bytree=0.65,subsample=1)
x_cfl.fit(x_train,y_train,verbose=True)

predict_y = x_cfl.predict(x_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, predict_y, pos_label=1)
print ("The train score is:",metrics.auc(fpr, tpr))

predict_y = x_cfl.predict(x_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, predict_y, pos_label=1)
print("The test score is:",metrics.auc(fpr, tpr))

plot_confusion_matrix(y_test, predict_y)

print("-"*20, "Feature Importance", "-"*20)

features = x_train.columns
importances = x_cfl.feature_importances_
indices = (np.argsort(importances))[-10:]
plt.figure(figsize=(8,8))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()

print("Time taken to run this cell :", datetime.now() - start)
```

```
The train score is: 0.75357415066411
The test score is: 0.7035953498443067
-------------------- Confusion matrix --------------------
```




```
-------------------- Feature Importance --------------------
```




```
Time taken to run this cell : 0:12:11.856915
```

Observations:

1. XGBoost is already a good starting point if the classes are not skewed too much, because it internally takes care that the bags it trains on are not imbalanced. But then again, the data is resampled, it is just happening secretly.

1. We tried the hyperparameter tuning(n_estimators,max_depth,learning_rate,colsample_bytree,subsample) in differant notebook, used the values here, tried to tweak them and could see better results than Logistic regression.

1. The scorimg metric we used here is auc which is equivalent to the probability that a random pair of a positive sample (clicked ad) and a negative one (unclicked ad) is ranked correctly.

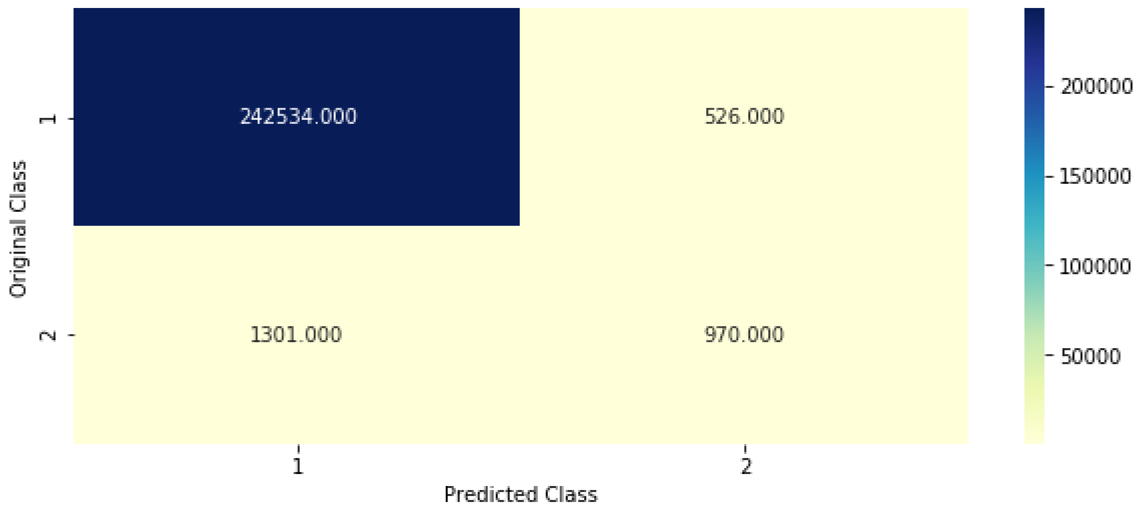
1. We tried to plot the confusion matrix inorder to better understand the results.

1. We tried to get feature importance to understand which features are important. Here we printed top 10 important features. We could see from the above feature importance plot that even raw ID features are very important.
2. We tried using all the features engineered above, but using averageor mean CTR features only gave us better results.

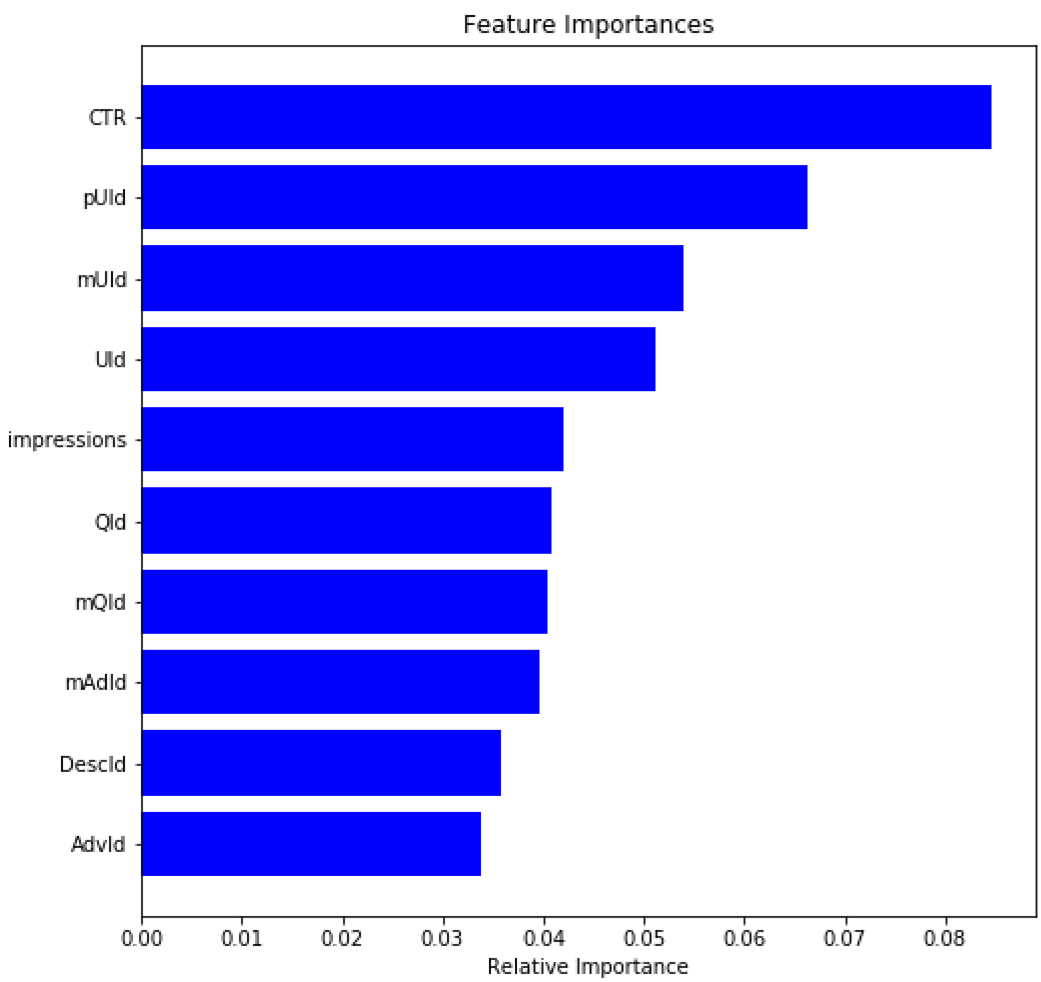
1. Here we got the result train score = 0.75 and test score = 0.70

In [35]:
```
start = datetime.now()

x_cfl=XGBClassifier(n_estimators=455,max_depth=3,learning_rate=0.137,colsample
_bytree=0.65,subsample=1)
x_cfl.fit(x_train,y_train,verbose=True)

predict_y = x_cfl.predict(x_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, predict_y, pos_label=1)
print ("The train score is:",metrics.auc(fpr, tpr))

predict_y = x_cfl.predict(x_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, predict_y, pos_label=1)
print("The test score is:",metrics.auc(fpr, tpr))

plot_confusion_matrix(y_test, predict_y)

print("-"*20, "Feature Importance", "-"*20)

features = x_train.columns
importances = x_cfl.feature_importances_
indices = (np.argsort(importances))[-10:]
plt.figure(figsize=(8,8))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()

print("Time taken to run this cell :", datetime.now() - start)
```

```
The train score is: 0.754796176153984
The test score is: 0.7124802699965576
-------------------- Confusion matrix --------------------
```




```
-------------------- Feature Importance --------------------
```




```
Time taken to run this cell : 0:11:53.470662
```

Observations:

1. XGBoost is already a good starting point if the classes are not skewed too much, because it internally takes care that the bags it trains on are not imbalanced. But then again, the data is resampled, it is just happening secretly.

1. We tried the hyperparameter tuning(n_estimators,max_depth,learning_rate,colsample_bytree,subsample) in differant notebook, used the values here, tried to tweak them and could see better results than Logistic regression.

1. The scorimg metric we used here is auc which is equivalent to the probability that a random pair of a positive sample (clicked ad) and a negative one (unclicked ad) is ranked correctly.

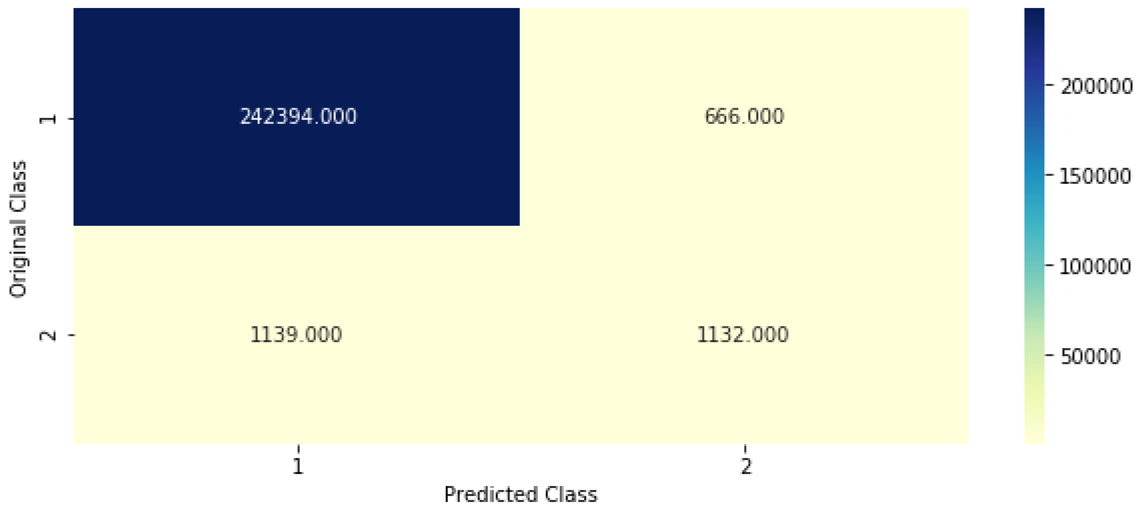
1. We tried to plot the confusion matrix inorder to better understand the results.

1. We tried to get feature importance to understand which features are important. Here we printed top 10 important features. We could see from the above feature importance plot that even raw ID features are very important.
2. We tried using all the features engineered above, but using averageor mean CTR features only gave us better results.

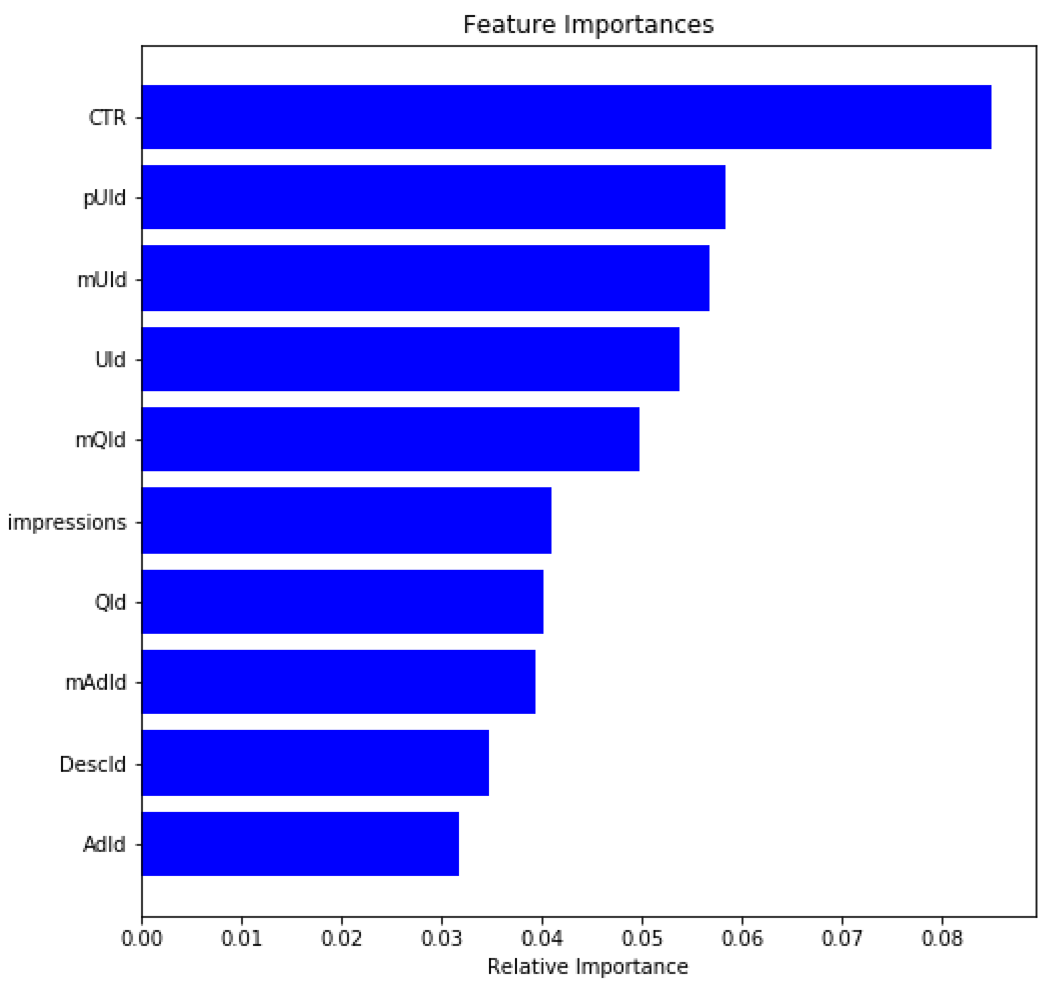
1. Here we got the result train score = 0.75 and test score = 0.71 (improved just by tweaking the parameters)

In [37]:
```
start = datetime.now()

x_cfl=XGBClassifier(n_estimators=455,max_depth=3,learning_rate=0.1373,colsampl
e_bytree=0.65,subsample=1)
x_cfl.fit(x_train,y_train,verbose=True)

predict_y = x_cfl.predict(x_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, predict_y, pos_label=1)
print ("The train score is:",metrics.auc(fpr, tpr))

predict_y = x_cfl.predict(x_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, predict_y, pos_label=1)
print("The test score is:",metrics.auc(fpr, tpr))

plot_confusion_matrix(y_test, predict_y)

print("-"*20, "Feature Importance", "-"*20)

features = x_train.columns
importances = x_cfl.feature_importances_
indices = (np.argsort(importances))[-10:]
plt.figure(figsize=(8,8))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()

print("Time taken to run this cell :", datetime.now() - start)
```

```
The train score is: 0.7567446759931792
The test score is: 0.747859382264068
-------------------- Confusion matrix --------------------
```




```
-------------------- Feature Importance --------------------
```




```
Time taken to run this cell : 0:12:10.394359
```

Observations:

1. XGBoost is already a good starting point if the classes are not skewed too much, because it internally takes care that the bags it trains on are not imbalanced. But then again, the data is resampled, it is just happening secretly.

1. We tried the hyperparameter tuning(n_estimators,max_depth,learning_rate,colsample_bytree,subsample) in differant notebook, used the values here, tried to tweak them and could see better results than Logistic regression.

1. The scorimg metric we used here is auc which is equivalent to the probability that a random pair of a positive sample (clicked ad) and a negative one (unclicked ad) is ranked correctly.

1. We tried to plot the confusion matrix inorder to better understand the results.

1. We tried to get feature importance to understand which features are important. Here we printed top 10 important features. We could see from the above feature importance plot that even raw ID features are very important.
2. We tried using all the features engineered above, but using averageor mean CTR features only gave us better results.

1. Here we got the result train score = 0.75 and test score = 0.74 (improved just by tweaking the parameters)

Conclusion:

In [72]:
```
x = PrettyTable()

x.field_names = ["Model","Test Score %"]
x.add_row([ "Logistic Regression",50])
x.add_row([ "XGBOOST",74.78])

print(x)
```

```
+---------------------+--------------+
|        Model        | Test Score % |
+---------------------+--------------+
| Logistic Regression |      50      |
|       XGBOOST       |    74.78     |
+---------------------+--------------+
```

From the above table we could observe that XGBOOST model beats Logistic Regression model as out of the box classifiers like Logistic Regression discards the rare classes.