1.Importing packages

In [0]:

```
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [3]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

2. Loading Byte bigram features and removing top 200 features and converting to dataframe

In [0]:

```
from sklearn.preprocessing import normalize
import scipy
byte_bigram_vect = normalize(scipy.sparse.load_npz('/content/drive/My Drive/microsoft
malware/bytebigram.npz'), axis = 0)

# type(byte_bigram_vect)
```

In [0]:

```
byte_vocab =
"00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,2
22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,
4,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,
,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,8
89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,
b,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,
,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,e
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [0]:

```
byte_bigram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for j in range(0, len(byte_vocab.split(','))):
        byte_bigram_vocab.append(v + ' ' +byte_vocab.split(',')[j])
len(byte_bigram_vocab)
```

Out[0]:

```
66049
```

In [0]:

```
byte_bigram_vocab[:5]
```

Out[0]:

```
['00 00', '00 01', '00 02', '00 03', '00 04']
```

In [0]:

```
byte_bigram_vect.toarray()
```

Out[0]:

```
array([[0.07176836, 0.01576409, 0.01615476, ..., 0.        , 0.        ,
         0.        ],
        [0.00521783, 0.01131176, 0.00129077, ..., 0.        , 0.        ,
         0.        ],
        [0.0042138 , 0.00931371, 0.00316641, ..., 0.        , 0.        ,
         0.        ],
        ...,
        [0.00453893, 0.02443276, 0.02067244, ..., 0.        , 0.        ,
         0.        ],
        [0.00456758, 0.01843863, 0.0102858 , ..., 0.        , 0.        ,
         0.        ],
        [0.00411234, 0.00084956, 0.00046387, ..., 0.        , 0.        ,
         0.        ]])
```

In [0]:

```
byte_bigram_DataFrame = pd.SparseDataFrame(byte_bigram_vect.toarray(),columns = byte_bigram_vocab)

byte_bigram_DataFrame.shape
```

Out[0]:

```
(5000, 66049)
```

In [0]:

```
byte_bigram_DataFrame.head()
```

Out[0]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	00 0a	00 0b	00 0c

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	00 0a	00 0b	00 0c	
0	0.071768	0.015764	0.016155	0.028769	0.018437	0.022912	0.026551	0.030253	0.021756	0.038112	0.026084	0.031898	0.036654	(
1	0.005218	0.011312	0.001291	0.001057	0.003109	0.000273	0.000142	0.000302	0.000885	0.000309	0.000322	0.000255	0.000660	(
2	0.004214	0.009314	0.003166	0.003541	0.009952	0.016093	0.013004	0.004404	0.013231	0.005947	0.004155	0.004246	0.015057	(
3	0.002603	0.003209	0.001190	0.001697	0.002014	0.000927	0.000448	0.000633	0.001391	0.000541	0.000563	0.000481	0.001893	(
4	0.004019	0.000912	0.000403	0.002705	0.000156	0.000300	0.000071	0.000151	0.000202	0.000077	0.000000	0.000085	0.000201	(

5 rows × 66049 columns

In [0]:

```
final_bytes = pd.read_csv('/content/drive/My Drive/microsoft malware/final_bytes.csv')
```

In [0]:

```
final_bytes.shape
```

Out[0]:

```
(5000, 261)
```

In [0]:

```
result_y = final_bytes['Class']
```

In [0]:

```
### Function for getting top features
```

In [0]:

```
def imp_features(data, features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    sns.set()
    plt.figure(figsize = (10, 5))
    ax = sns.barplot(x = imp_feature_name, y = imp_value)
    ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
    sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
    plt.title('Important Features')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')
    return imp_feature_indx[:keep]
```

In [0]:

```
byte_bigram_index = imp_features(normalize(byte_bigram_vect, axis = 0), byte_bigram_vocab, 200)
```

In [0]:

```
best_byte_bigram = np.zeros((5000, 0))
for i in byte_bigram_index:
    sliced = byte_bigram_vect[:, i].todense()
    best_byte_bigram = np.hstack([best_byte_bigram, sliced])
```

In [0]:

```
 best_byte_bigram_dataframe = pd.SparseDataFrame(best_byte_bigram, columns = np.take(byte_bigram_vo
cab, byte_bigram_index))
```

In [0]:

In [0]:

```
best_byte_bigram_dataframe = best_byte_bigram_dataframe.to_dense()
```

In [0]:

```
best_byte_bigram_dataframe.head()
```

Out[0]:

	8f 9a	8e 9a	b5 bb	ca d3	8b ff	8a 06	0f 7f	8b 5d	4e 47	28 5d	f5 a9	20 69	93 2e	
0	0.002198	0.009609	0.008819	0.006042	0.000324	0.001846	0.000270	0.000364	0.000835	0.000238	0.017296	0.000163	0.00668	0.
1	0.000000	0.008408	0.011759	0.003021	0.009774	0.005075	0.000631	0.003637	0.001838	0.000714	0.000000	0.000722	0.00000	0.
2	0.006594	0.006005	0.004410	0.005035	0.030903	0.002768	0.000225	0.002983	0.000501	0.000714	0.006290	0.000140	0.00668	0.
3	0.000000	0.002402	0.000000	0.000000	0.013626	0.000461	0.000045	0.003128	0.002339	0.000000	0.000000	0.000116	0.00167	0.
4	0.002198	0.000000	0.002940	0.001007	0.001663	0.000692	0.000000	0.000073	0.000334	0.000238	0.000000	0.000000	0.00501	0.

5 rows × 200 columns

In [0]:

```
best_byte_bigram_dataframe.shape
```

Out[0]:

```
(5000, 200)
```

In [0]:

```
best_byte_bigram_dataframe = best_byte_bigram_dataframe.fillna(0)
```

In [0]:

```
best_byte_bigram_dataframe['ID'] = final_bytes.ID
```

In [0]:

```
best_byte_bigram_dataframe.head()
```

Out[0]:

	8f 9a	8e 9a	b5 bb	ca d3	8b ff	8a 06	0f 7f	8b 5d	4e 47	28 5d	f5 a9	20 69	93 2e	
0	0.002198	0.009609	0.008819	0.006042	0.000324	0.001846	0.000270	0.000364	0.000835	0.000238	0.017296	0.000163	0.00668	0.
1	0.000000	0.008408	0.011759	0.003021	0.009774	0.005075	0.000631	0.003637	0.001838	0.000714	0.000000	0.000722	0.00000	0.
2	0.006594	0.006005	0.004410	0.005035	0.030903	0.002768	0.000225	0.002983	0.000501	0.000714	0.006290	0.000140	0.00668	0.
3	0.000000	0.002402	0.000000	0.000000	0.013626	0.000461	0.000045	0.003128	0.002339	0.000000	0.000000	0.000116	0.00167	0.
4	0.002198	0.000000	0.002940	0.001007	0.001663	0.000692	0.000000	0.000073	0.000334	0.000238	0.000000	0.000000	0.00501	0.

5 rows × 201 columns

In [0]:

```
best_byte_bigram_dataframe.shape
```

Out[0]:

```
(5000, 201)
```

In [0]:

```
# best_byte_bigram_dataframe =
best_byte_bigram_dataframe.drop(best_byte_bigram_dataframe.index[451])
```

```
# best_byte_bigram_dataframe =
best_byte_bigram_Datarame.drop(best_byte_bigram_Datarame.index[512])
# best_byte_bigram_Datarame =
best_byte_bigram_Datarame.drop(best_byte_bigram_Datarame.index[1601])
```

In [0]:

```
best_byte_bigram_dataframe.shape
```

Out[0]:

```
(4997, 201)
```

In [0]:

```
best_byte_bigram_dataframe.to_csv('/content/drive/My Drive/microsoft
malware/best_byte_bigram_dataframe.csv')
```

In [0]:

```
best_byte_bigram_dataframe.to_csv('/content/drive/My Drive/microsoft
malware/best_byte_bigram_dataframe_5k.csv')
```

In [0]:

```
byte_bigram_df = byte_bigram_df.to_dense()
```

In [0]:

```
byte_bigram_df.head()
```

In [0]:

```
byte_bigram_df = byte_bigram_df.fillna(0)
```

In [0]:

```
byte_bigram_DataFrame['ID'] = final_bytes.ID
```

In [0]:

```
asm_opcode_bigram_df['ID'] = result_x.ID
# asm_opcode_bigram_df.head()
```

3. Loading all files

In [0]:

```
final_bytes = pd.read_csv('/content/drive/My Drive/microsoft malware/final_bytes.csv')
final_bytes_1 = pd.read_csv('/content/drive/My Drive/microsoft malware/final_bytes_1.csv')
final_asm = pd.read_csv('/content/drive/My Drive/microsoft malware/final_asm.csv')
asm_opcode_bigram_df = pd.read_csv('/content/drive/My Drive/microsoft
malware/asm_opcode_bigram_df.csv')
image_dataframe = joblib.load('/content/drive/My Drive/microsoft malware/image_dataframe')
image_dataframe['ID'] = final_bytes.ID
```

In [0]:

```
best_byte_bigram_dataframe = pd.read_csv('/content/drive/My Drive/microsoft
malware/best_byte_bigram_dataframe_5k.csv')
```

In [0]:

```
# image_dataframe = image_dataframe.drop(image_dataframe.index[451])
# image_dataframe = image_dataframe.drop(image_dataframe.index[512])
# image_dataframe = image_dataframe.drop(image_dataframe.index[1601])
# image_dataframe['ID'] = final_asm.ID
# image_dataframe.shape
```

Out[0]:

```
(4997, 201)
```

In [0]:

```
result_x = pd.merge(final_bytes_1,final_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[0]:

	Unnamed: 0_x	ID	size_x	0	1	2	3	4	5	6	7
0	0	01azqd4InC7m9JpocGv5	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.00
1	1	01IsoiSMh5gxyDYTl4CB	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.01
2	2	01jsnpXSAlgw6aPeDxrU	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.01
3	3	01kcPWA9K2BOxQeS5Rju	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.00
4	4	01SuzwMJEIXsK7A8dQbl	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000430	0.00

5 rows × 310 columns

In [0]:

```
print('Shape of BYTE BIGRAMS :',best_byte_bigram_dataframe.shape)
print('Shape of ASM OPCODE BIGRAMS :',asm_opcode_bigram_df.shape)
print('Shape of Image Data:',image_dataframe.shape)
print('Shape of result_x:',result_x.shape)
```

```
Shape of BYTE BIGRAMS : (5000, 202)
Shape of ASM OPCODE BIGRAMS : (4997, 202)
Shape of Image Data: (5000, 201)
Shape of result_x: (4997, 310)
```

In [0]:

```
result_x.head()
```

Out[0]:

	Unnamed: 0_x	ID	size_x	0	1	2	3	4	5	6	7
0	0	01azqd4InC7m9JpocGv5	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.00
1	1	01IsoiSMh5gxyDYTl4CB	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.01
2	2	01jsnpXSAlgw6aPeDxrU	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.01
3	3	01kcPWA9K2BOxQeS5Rju	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.00
4	4	01SuzwMJEIXsK7A8dQbl	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000430	0.00

5 rows × 310 columns

In [0]:

```
# asm_opcode_bigram_df.head()
```

Out[0]:

jmp jmp	jmp mov	jmp	jmp pop	jmp xor	jmp sub	jmp dec	jmp add	jmp cmp	jmp	jmp lea	jmp	mov jmp	n

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp dec	jmp add	jmp cmp	jmp jz	jmp lea	jmp movzx	mov jmp	n n
0	0.046114	0.005294	0.000578	0.000000	0.003371	0.011986	0.001871	0.023819	0.000151	0.0	0.000561	0.000000	0.005109	0.004
1	0.000000	0.000882	0.000289	0.000556	0.000595	0.000000	0.000000	0.002802	0.000000	0.0	0.000281	0.000000	0.001053	0.002
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000
3	0.000000	0.000138	0.000096	0.000000	0.000397	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000093	0.000
4	0.000524	0.001572	0.000385	0.000556	0.000198	0.000000	0.000000	0.000000	0.000151	0.0	0.000000	0.001093	0.001579	0.002

5 rows × 201 columns

In [0]:

```
asm_opcode_bigram_df = asm_opcode_bigram_df.drop(['Unnamed: 0'],axis = 1)
result_x = result_x.drop(['Unnamed: 0_x'],axis = 1)
```

In [0]:

```
result_x.head()
```

Out[0]:

	ID	size_x	0	1	2	3	4	5	6	7	8	
0	01azqd4InC7m9JpocGv5	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.003511	0.003
1	01IsoiSMh5gxyDYTl4CB	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.011003	0.000
2	01jsnpXSAlgw6aPeDxrU	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.010785	0.002
3	01kcPWA9K2BOxQeS5Rju	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.001277	0.000
4	01SuzwMJEIXsK7A8dQbl	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000430	0.000500	0.000

5 rows × 309 columns

In [0]:

```
# asm_opcode_bigram_df.head()
```

Out[0]:

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp dec	jmp add	jmp cmp	jmp jz	jmp lea	jmp movzx	mov jmp	n n
0	0.046114	0.005294	0.000578	0.000000	0.003371	0.011986	0.001871	0.023819	0.000151	0.0	0.000561	0.000000	0.005109	0.004
1	0.000000	0.000882	0.000289	0.000556	0.000595	0.000000	0.000000	0.002802	0.000000	0.0	0.000281	0.000000	0.001053	0.002
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000
3	0.000000	0.000138	0.000096	0.000000	0.000397	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000093	0.000
4	0.000524	0.001572	0.000385	0.000556	0.000198	0.000000	0.000000	0.000000	0.000151	0.0	0.000000	0.001093	0.001579	0.002

5 rows × 201 columns

In [0]:

```
print('Shape of BYTE BIGRAMS :',best_byte_bigram_dataframe.shape)
print('Shape of ASM OPCODE BIGRAMS :',asm_opcode_bigram_df.shape)
print('Shape of Image Data:',image_dataframe.shape)
print('Shape of result_x:',result_x.shape)
```

```
Shape of BYTE BIGRAMS : (5000, 202)
Shape of ASM OPCODE BIGRAMS : (4997, 201)
Shape of Image Data: (5000, 201)
Shape of result_x: (4997, 309)
```

In [0]:

```
# res = pd.merge(df, df1, on='key')
```

In [0]:

```python
final_features = pd.merge(result_x, best_byte_bigram_dataframe, on = 'ID')
final_features = pd.merge(final_features, asm_opcode_bigram_df, on = 'ID')

final_features = pd.merge(final_features,image_dataframe ,on = 'ID')
```

In [0]:

```python
print(final_features.shape)
```

```
(4997, 910)
```

In [0]:

```python
final_features.head()
```

Out[0]:

	ID	size_x	0	1	2	3	4	5	6	7	8	
0	01azqd4InC7m9JpocGv5	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.003511	0.003
1	01IsoiSMh5gxyDYTl4CB	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.011003	0.000
2	01jsnpXSAlgw6aPeDxrU	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.010785	0.002
3	01kcPWA9K2BOxQeS5Rju	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.001277	0.000
4	01SuzwMJEIXsK7A8dQbl	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000430	0.000500	0.000

5 rows × 710 columns

In [0]:

```python
final_features.iloc[4996]
```

Out[0]:

```
ID            CbRnEdeAj2FNzmDfZMQI
size_x                  0.00532765
0                        0.0172442
1                       0.00150642
2                      0.000281969
                         ...
pixel 195                0.0670959
pixel 196                0.0670959
pixel 197                0.0670959
pixel 198                0.0670959
pixel 199                0.0670959
Name: 4996, Length: 710, dtype: object
```

In [0]:

```python
final_features = final_features.drop(['ID'], axis = 1)
```

In [0]:

```python
final_features.head()
```

Out[0]:

	size_x	0	1	2	3	4	5	6	7	8	9	0a	0b
0	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.003511	0.003531	0.006862	0.007215
1	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.011003	0.000394	0.000727	0.013528
2	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.010785	0.002707	0.005674	0.005431
3	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.001277	0.000521	0.001103	0.000905

	size_x	0	1	2	3	4	5	6	7	8	9	0a	0b
4	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000439	0.000509	0.000246	0.000506	0.000460

5 rows × 909 columns

In [0]:

```
result_y.shape
```

Out[0]:

```
(4997,)
```

4. Splitting data into Train,Test and CV

In [0]:

```
X_train, X_test, y_train, y_test = train_test_split(final_features, result_y,stratify=result_y,test
_size=0.20)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

Shape of data of result_x + byte_bigrams + asm_opcode_bigrams + image_features

In [0]:

```
print('Shape of Train Data:',X_train.shape)
print('Shape of Test Data:',X_test.shape)
print('Shape of Cv Data:',X_cv.shape)
```

```
Shape of Train Data: (3197, 909)
Shape of Test Data: (1000, 909)
Shape of Cv Data: (800, 909)
```

Shape of data of result_x + byte_bigrams + image_features

In [0]:

```
print('Shape of Train Data:',X_train.shape)
print('Shape of Test Data:',X_test.shape)
print('Shape of Cv Data:',X_cv.shape)
```

```
Shape of Train Data: (3197, 709)
Shape of Test Data: (1000, 709)
Shape of Cv Data: (800, 709)
```

5. Function for ploting Confusion matrix

In [0]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*50, "Precision matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of columns in precision matrix",B.sum(axis=0))

    # representing B in heatmap format
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

6. Modelling using XGboost

6.1. Model 1 : Modeling using all features

In [0]:

```
x_cfl=XGBClassifier()

prams={
     'n_estimators':[1000,3000,5000],
     'max_depth':[2,3,5],
    'min_child_weight':[3,5]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  6.0min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 33.2min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 77.4min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 102.4min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 147.4min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 223.8min finished
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [2, 3, 5],
                                        'min_child_weight': [3, 5],
                                        'n_estimators': [1000, 3000, 5000]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [0]:

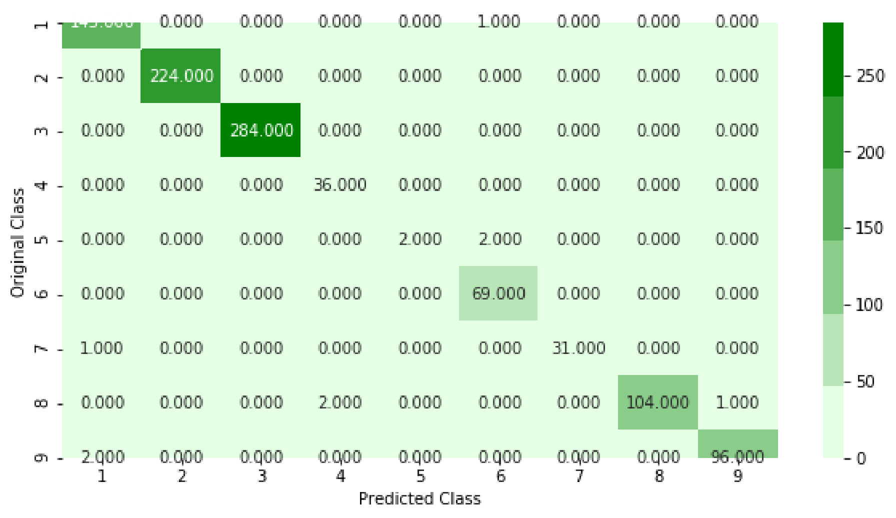
```
print (random_cfl.best_params_)
```

```
{'n_estimators': 1000, 'min_child_weight': 3, 'max_depth': 3}
```

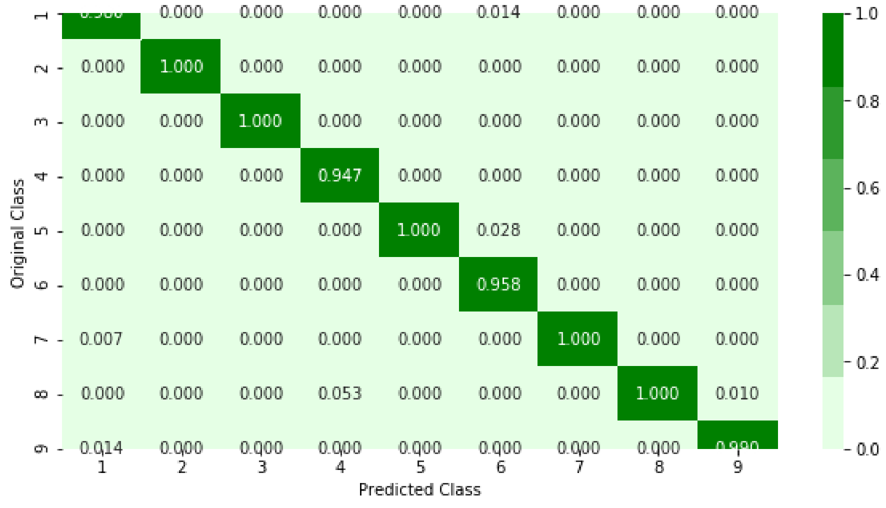
In [0]:

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=1000,max_depth=3,min_child_weight=3,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

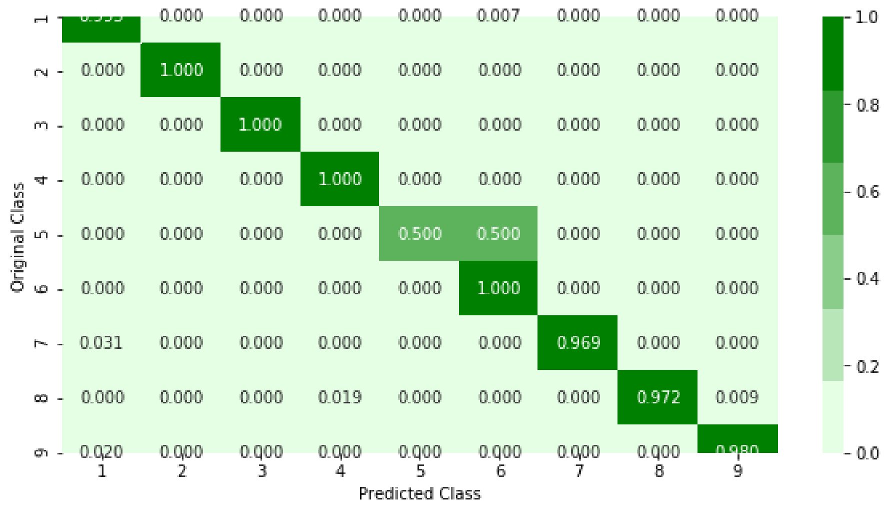
```
The train log loss is: 0.02287424609629777
The cross validation log loss is: 0.04725959883058853
The test log loss is: 0.05374319905601612
-------------------------------------------------- Confusion matrix --------------------------------
----------------
```



```
-------------------------------------------------- Precision matrix --------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ---------------------------------
-------------
```



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

6.2. Model 2 : Modeling using byte bigram, image and asm bigram features

In [0]:

```
x_cfl=XGBClassifier()

prams={
     'n_estimators':[1000,3000,5000],
     'max_depth':[2,3,5],
    'min_child_weight':[3,5]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 14.1min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 24.6min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 56.2min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 96.0min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 150.8min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 229.3min finished
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [2, 3, 5],
                                        'min_child_weight': [3, 5],
                                        'n_estimators': [1000, 3000, 5000]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [0]:

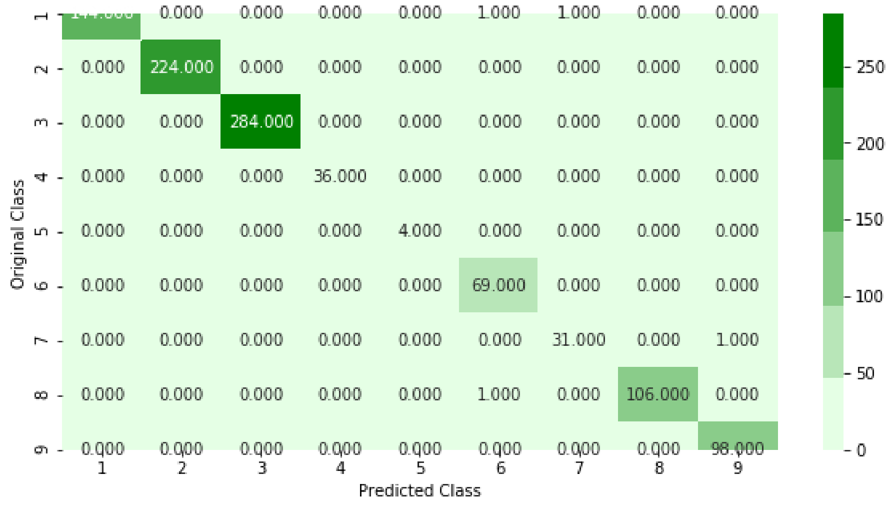
```
print (random_cfl.best_params_)
```

```
{'n_estimators': 3000, 'min_child_weight': 3, 'max_depth': 2}
```

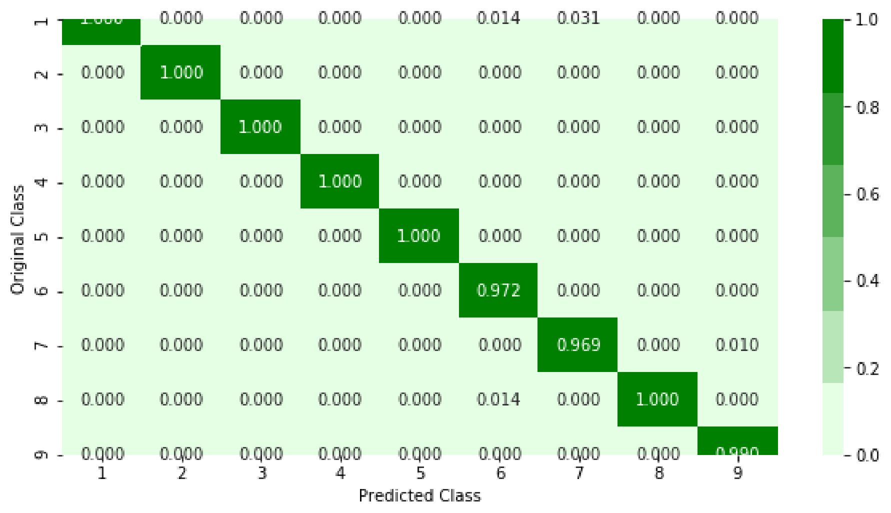
In [0]:

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=2,min_child_weight=3,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

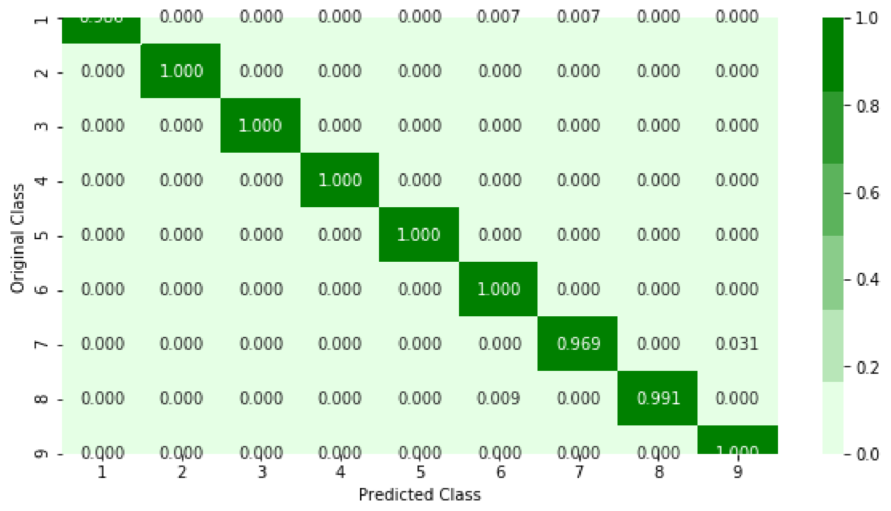
```
The train log loss is: 0.020916052468548132
The cross validation log loss is: 0.053713864339213394
The test log loss is: 0.0330693413108238
Number of misclassified points  0.4
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```



```
-------------------------------------------------- Precision matrix -------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ----------------------------------
-------------
```



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

6.3. Model 3 : Modeling using result_x, byte bigram, image and asm opcodes bigram features

In [0]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.1,0.15,0.2],
     'n_estimators':[1000,3000,5000],
     'max_depth':[2,3,5],
     'colsample_bytree':[0.1,0.3,0.5,1],
     'subsample':[0.1,0.3,0.5,1],
    'min_child_weight':[3,5]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
```

```
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  5.2min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 10.5min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 27.9min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 48.4min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 66.8min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 101.6min finished
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.1, 0.15, 0.2],
                                        'max_depth': [2, 3, 5],
                                        'min_child_weight': [3, 5],
                                        'n_estimators': [1000, 3000, 5000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [0]:

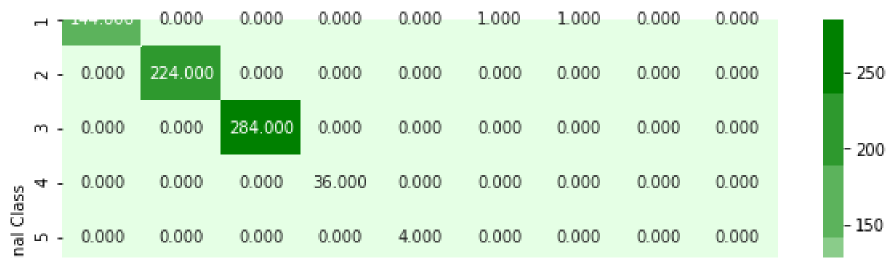
```
print (random_cfl.best_params_)
```

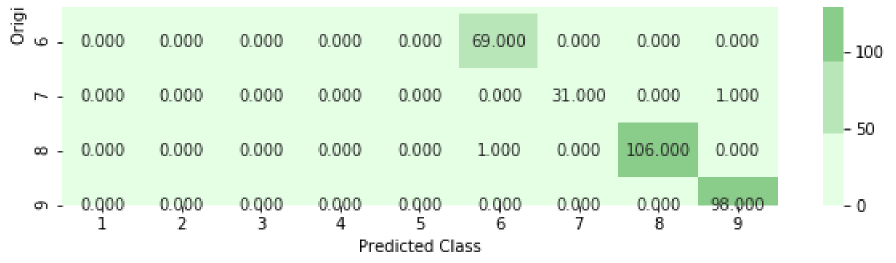
```
{'subsample': 1, 'n_estimators': 5000, 'min_child_weight': 3, 'max_depth': 3, 'learning_rate':
0.1, 'colsample_bytree': 0.5}
```

In [0]:

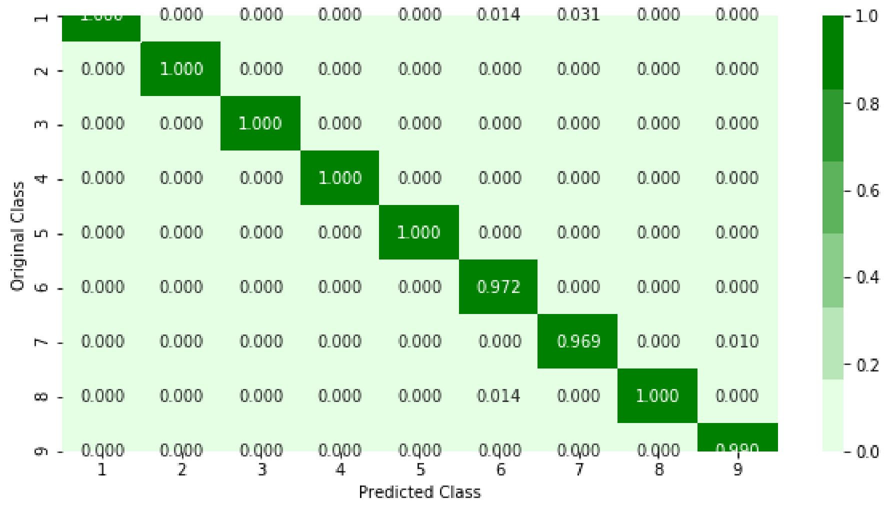
```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=5000,max_depth=3,min_child_weight=3,learning_rate = 0.1,colsample_
bytree = 0.5,subsample = 1,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

```
The train log loss is: 0.0210863552249353
The cross validation log loss is: 0.05758418365121257
The test log loss is: 0.0342343958898736
Number of misclassified points  0.4
-------------------------------------------------- Confusion matrix ---------------------------------
----------------
```

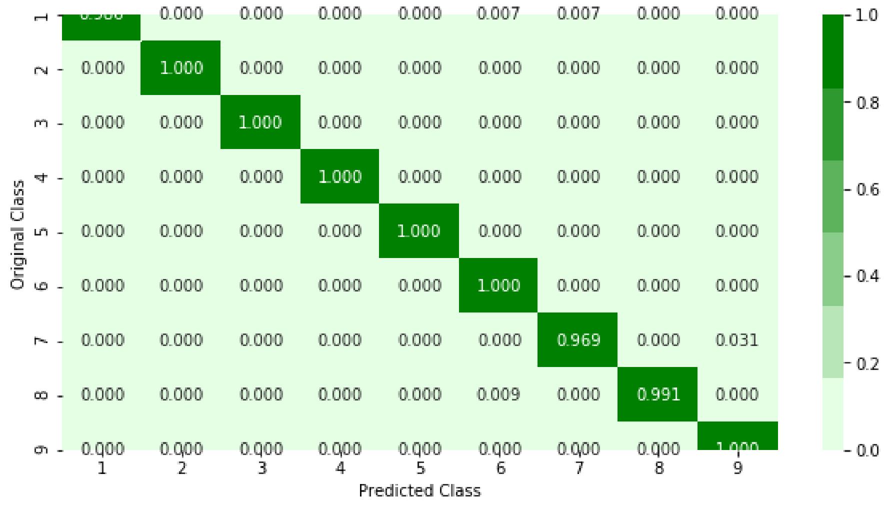




```
---------------------------------------------------- Precision matrix ----------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
---------------------------------------------------- Recall matrix ----------------------------------
-------------
```



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [0]:

6.4. Model 4 : Modeling using byte bigram and image features

In [0]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.1,0.15,0.2],
     'n_estimators':[1000,3000,5000],
     'max_depth':[2,3,5],
     'colsample_bytree':[0.1,0.3,0.5,1],
     'subsample':[0.1,0.3,0.5,1],
    'min_child_weight':[3,5]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 11.5min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 30.1min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 44.1min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 65.4min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 113.4min finished
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.1, 0.15, 0.2],
                                        'max_depth': [2, 3, 5],
                                        'min_child_weight': [3, 5],
                                        'n_estimators': [1000, 3000, 5000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [0]:

```
print (random_cfl.best_params_)
```

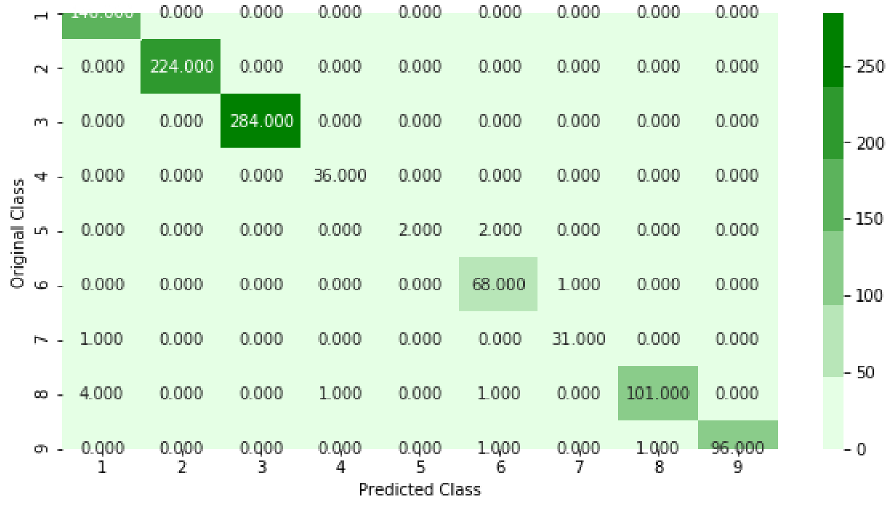
```
{'subsample': 1, 'n_estimators': 3000, 'min_child_weight': 3, 'max_depth': 3, 'learning_rate':
0.1, 'colsample_bytree': 0.3}
```

In [0]:

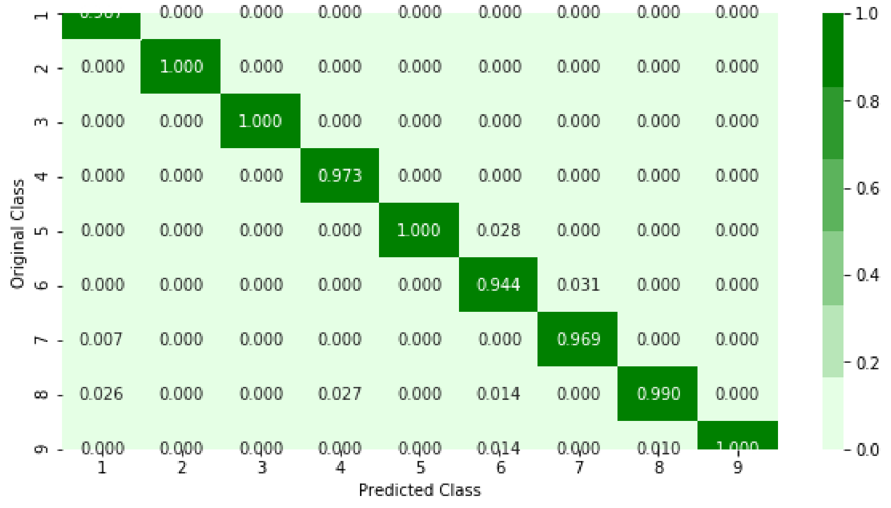
```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=3,min_child_weight=3,learning_rate = 0.1,colsample_
bytree = 0.3,subsample = 1,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

```
The train log loss is: 0.018968541151271664
The cross validation log loss is: 0.05977644208932924
The test log loss is: 0.05382138491527529
```

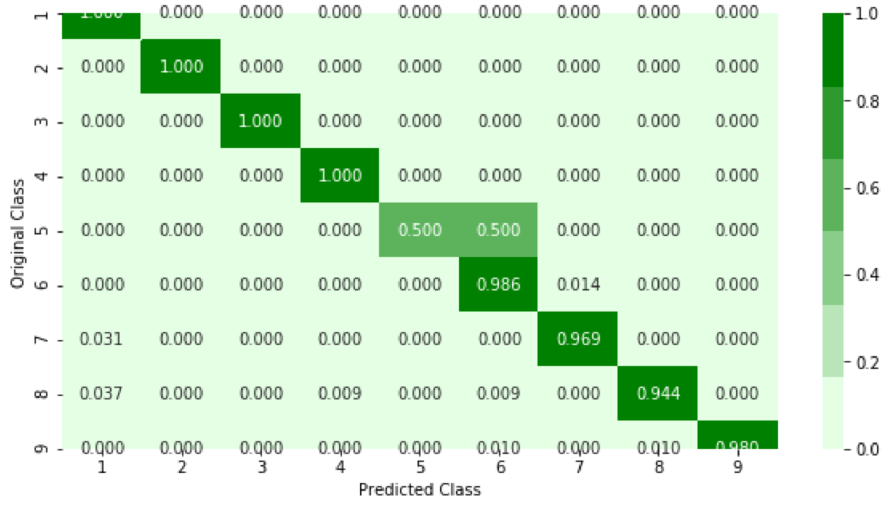
```
Number of misclassified points  1.2
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```



```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ---------------------------------
-------------
```



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

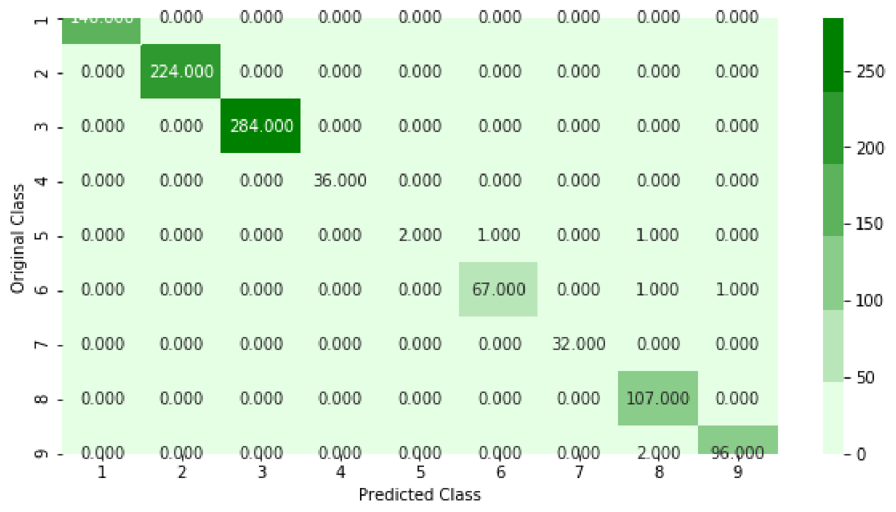
```
In [0]:
```

6.5. Model 5 : Modeling using result_x, byte bigram and image features

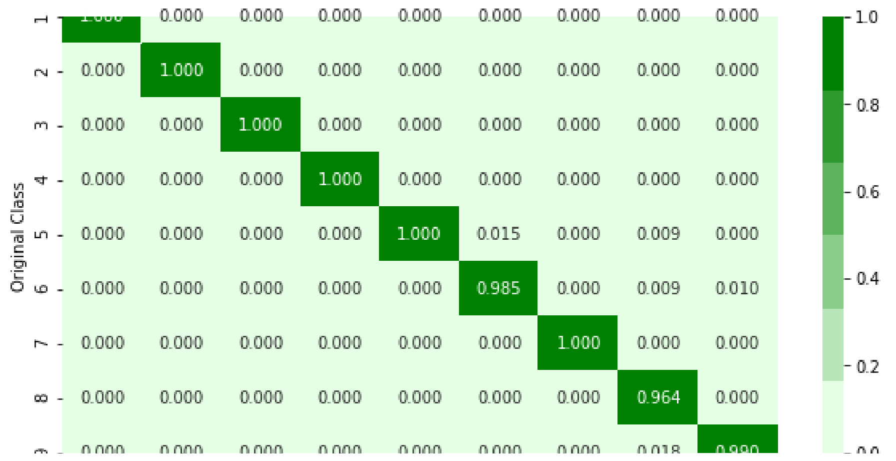
```
In [0]:
```

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.3,subsampl
e=1,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

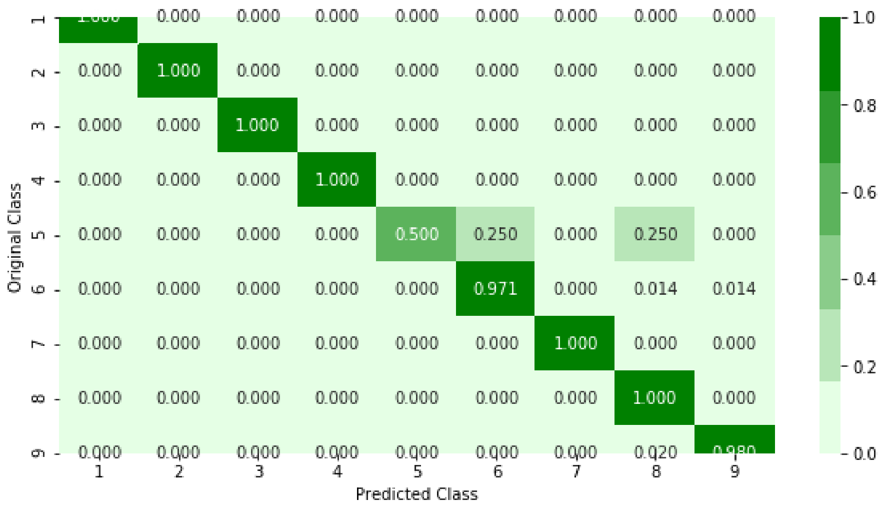
```
The train log loss is: 0.021005210904526974
The cross validation log loss is: 0.04930694995332006
The test log loss is: 0.03886770972767574
Number of misclassified points  0.6
-------------------------------------------------- Confusion matrix ---------------------------------
----------------
```



```
-------------------------------------------------- Precision matrix ---------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ----------------------------------
-------------
```



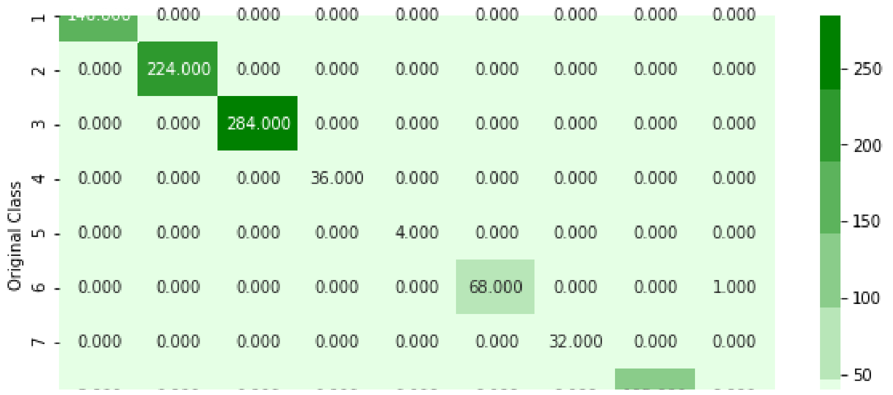
```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

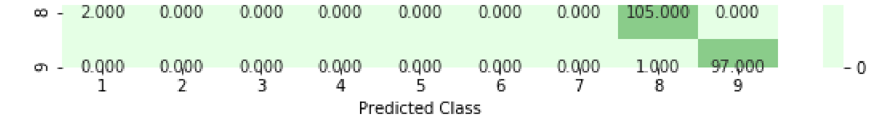
6.6. Model 6 : Modeling using result_x, byte bigram, image and asm opcodes bigram features

In [0]:

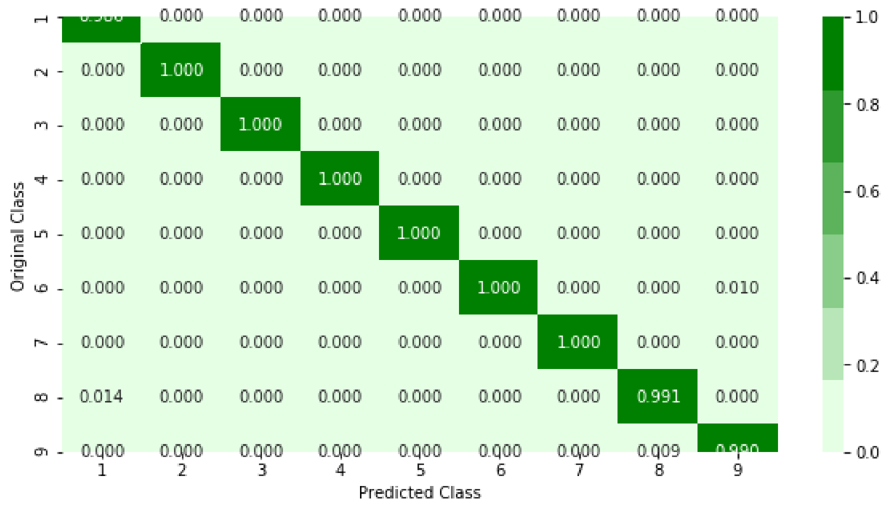
```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=15,learning_rate=0.15,colsample_bytree=0.3,subsampl
e=1,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

```
The train log loss is: 0.019683406716102143
The cross validation log loss is: 0.03843929098695146
The test log loss is: 0.03738138327587288
Number of misclassified points  0.4
-------------------------------------------------- Confusion matrix -------------------------------
----------------
```

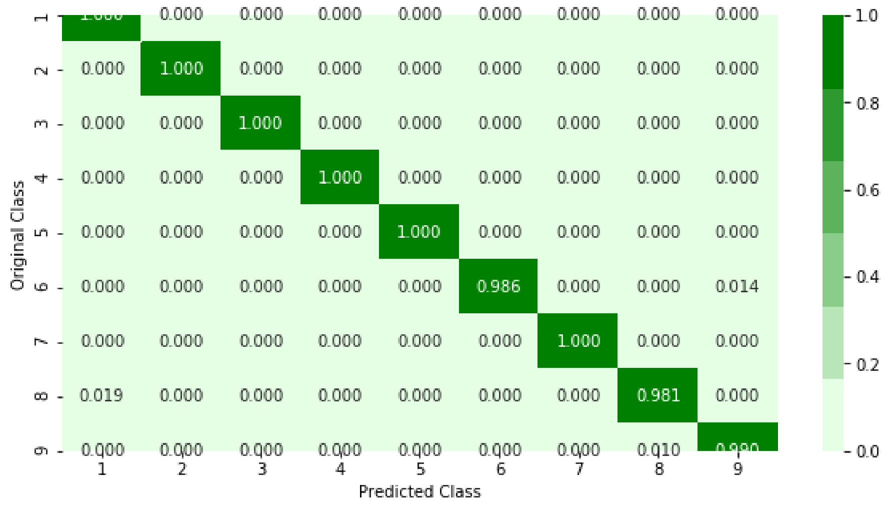




```
---------------------------------------------------- Precision matrix ------------------------------------------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
---------------------------------------------------- Recall matrix --------------------------------------------------
```



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

6.7. Model 7 : Modeling using result_x,byte bigram, image and asm opcodes bigram features

In [0]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.15,0.2,0.25],
     'n_estimators':[2000,3000,5000,6000],
     'max_depth':[10,15,25,30],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  9.3min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 18.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 35.3min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 69.7min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 94.7min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 135.4min finished
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.15, 0.2, 0.25],
                                        'max_depth': [10, 15, 25, 30],
                                        'n_estimators': [2000, 3000, 5000,
                                                         6000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [0]:

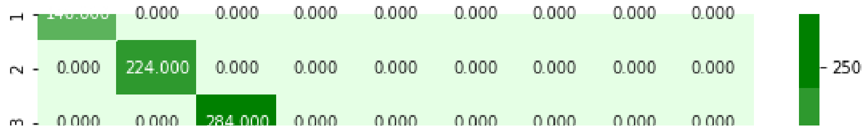
```
print (random_cfl.best_params_)
```

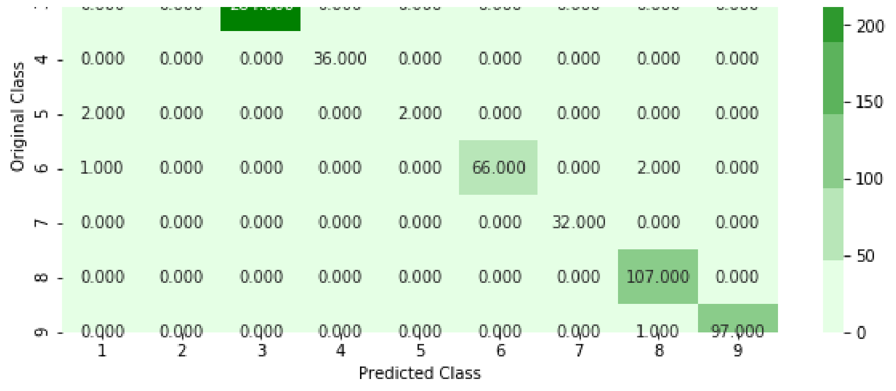
```
{'subsample': 0.3, 'n_estimators': 3000, 'max_depth': 15, 'learning_rate': 0.2,
'colsample_bytree': 1}
```

In [0]:

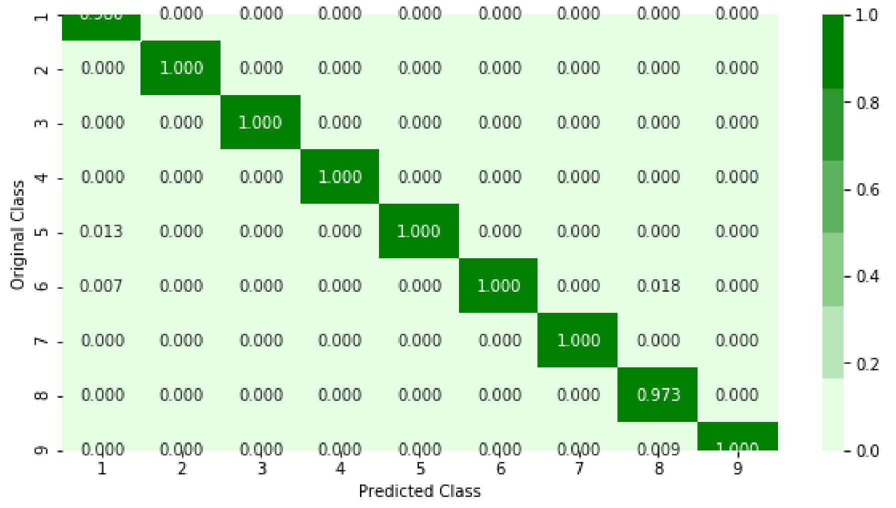
```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=15,learning_rate=0.2,colsample_bytree=1,subsample=
0.3,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

```
The train log loss is: 0.018566937098238583
The cross validation log loss is: 0.028775192362980602
The test log loss is: 0.031122629793037 99
Number of misclassified points  0.6
---------------------------------------------------- Confusion matrix --------------------------------
----------------
```

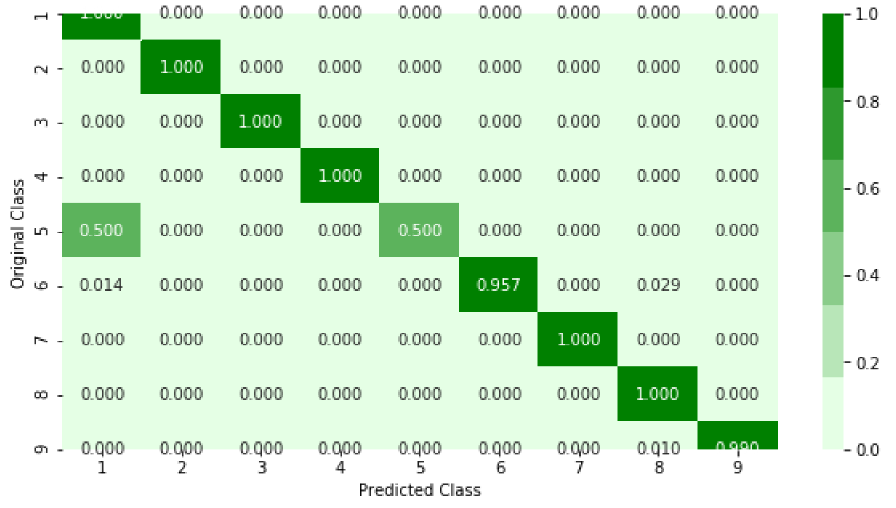




```
---------------------------------------------------- Precision matrix ----------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
---------------------------------------------------- Recall matrix -----------------------------------
-------------
```



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

6.8. Model 8 : Modeling using result_x,byte bigram, image and asm opcodes bigram features

In [0]:

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=15,learning_rate=0.25,colsample_bytree=1,subsample
=0.3,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

```
The train log loss is: 0.018601718840463 81
The cross validation log loss is: 0.030428671311987506
The test log loss is: 0.03389272106316636
```

6.9. Model 9 : Modeling using byte bigram, image and asm bytesbigram features

In [0]:

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=15,learning_rate=0.2,colsample_bytree=1,subsample=
0.3,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

```
The train log loss is: 0.018566937098238583
The cross validation log loss is: 0.028775192362980602
The test log loss is: 0.03112262979303799
```

6.10. Model 10 : Modeling using byte bigram, image and asm bytesbigram features

In [0]:

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=5000,max_depth=15,learning_rate=0.2,colsample_bytree=1,subsample=
0.3,nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

```
The train log loss is: 0.018445269768818565
The cross validation log loss is: 0.028564416548064697
The test log loss is: 0.03106145089227896
```

7. Merging byte bigrams,asm nigrams and image features and modelling on this data

In [0]:

```
final_bytes = pd.read_csv('/content/drive/My Drive/microsoft malware/final_bytes.csv')
final_bytes_1 = pd.read_csv('/content/drive/My Drive/microsoft malware/final_bytes_1.csv')
final_asm = pd.read_csv('/content/drive/My Drive/microsoft malware/final_asm.csv')
image_dataframe = joblib.load('/content/drive/My Drive/microsoft malware/image_dataframe')
image_dataframe['ID'] = final_bytes.ID
```

In [5]:

```
final_asm.shape
```

Out[5]:

```
(4997, 55)
```

In [6]:

```
final_asm.head()
```

Out[6]:

	Unnamed: 0	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	0	01azqd4InC7m9JpocGv5	0.403912	0.204545	0.032928	0.0	0.006937	0.543192	0.000000	0.000467	0.0	0.00000
1	1	01IsoiSMh5gxyDYTl4CB	0.100466	0.000000	0.161392	0.0	0.003690	0.009764	0.000000	0.006877	0.0	0.00000
2	2	01jsnpXSAlgw6aPeDxrU	0.061006	0.204545	0.101121	0.0	0.001821	0.000263	0.000000	0.000285	0.0	0.00000
3	3	01kcPWA9K2BOxQeS5Rju	0.000436	0.215909	0.001092	0.0	0.000761	0.000023	0.000000	0.000084	0.0	0.00007
4	4	01SuzwMJEIXsK7A8dQbl	0.007036	0.204545	0.015220	0.0	0.001234	0.001826	0.012842	0.000000	0.0	0.00007

In [0]:

```
best_byte_bigram_dataframe = pd.read_csv('/content/drive/My Drive/microsoft
malware/best_byte_bigram_dataframe_5k.csv')
```

In [0]:

```
final_features1 = pd.merge(final_asm , best_byte_bigram_dataframe,on = 'ID' )
```

In [0]:

```
final_features1 = pd.merge(final_features1,image_dataframe ,on = 'ID')
```

In [13]:

```
print(final_features1.shape)
```

```
(4997, 456)
```

In [14]:

```
final_features1.head()
```

Out[14]:

	Unnamed: 0_x	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	0	01azqd4InC7m9JpocGv5	0.403912	0.204545	0.032928	0.0	0.006937	0.543192	0.000000	0.000467	0.0	0.00000
1	1	01IsoiSMh5gxyDYTl4CB	0.100466	0.000000	0.161392	0.0	0.003690	0.009764	0.000000	0.006877	0.0	0.00000
2	2	01jsnpXSAlgw6aPeDxrU	0.061006	0.204545	0.101121	0.0	0.001821	0.000263	0.000000	0.000285	0.0	0.00000
3	3	01kcPWA9K2BOxQeS5Rju	0.000436	0.215909	0.001092	0.0	0.000761	0.000023	0.000000	0.000084	0.0	0.00007

	Unnamed: 0_x	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
4	4	01SuzwMJEIXsK7A8dQbl	0.007036	0.204545	0.015220	0.0	0.001234	0.001826	0.012842	0.000000	0.0	0.000072

5 rows × 456 columns

In [0]:

```
final_features1 = final_features1.drop(['Unnamed: 0_x'],axis = 1)
```

In [0]:

```
final_features1 = final_features1.drop(['ID'], axis = 1)
```

In [18]:

```
final_features1.head()
```

Out[18]:

	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	.reloc:	.BSS:	.CODE	j
0	0.403912	0.204545	0.032928	0.0	0.006937	0.543192	0.000000	0.000467	0.0	0.000000	0.0	0.00000	NaN	NaN	0.0367
1	0.100466	0.000000	0.161392	0.0	0.003690	0.009764	0.000000	0.006877	0.0	0.000000	0.0	0.00000	NaN	NaN	0.0023
2	0.061006	0.204545	0.101121	0.0	0.001821	0.000263	0.000000	0.000285	0.0	0.000000	0.0	0.00000	NaN	NaN	0.000(
3	0.000436	0.215909	0.001092	0.0	0.000761	0.000023	0.000000	0.000084	0.0	0.000072	0.0	0.00101	NaN	NaN	0.0006
4	0.007036	0.204545	0.015220	0.0	0.001234	0.001826	0.012842	0.000000	0.0	0.000072	0.0	0.00000	NaN	NaN	0.0036

5 rows × 454 columns

In [0]:

```
result_y = final_asm.Class
```

In [9]:

```
result_y.shape
```

Out[9]:

```
(4997,)
```

In [0]:

```
X_train, X_test, y_train, y_test = train_test_split(final_features1, result_y,stratify=result_y,tes
t_size=0.20)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

In [20]:

```
print('Shape of Train Data:',X_train.shape)
print('Shape of Test Data:',X_test.shape)
print('Shape of Cv Data:',X_cv.shape)
```

```
Shape of Train Data: (3197, 454)
Shape of Test Data: (1000, 454)
Shape of Cv Data: (800, 454)
```

In [25]:

```
%matplotlib inline
x_cfl=XGBClassifier(n_estimators=3000,max_depth=15,learning_rate=0.2,colsample_bytree=1,subsample=
0.3,objective="multi:softmax",nthread=-1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)
```

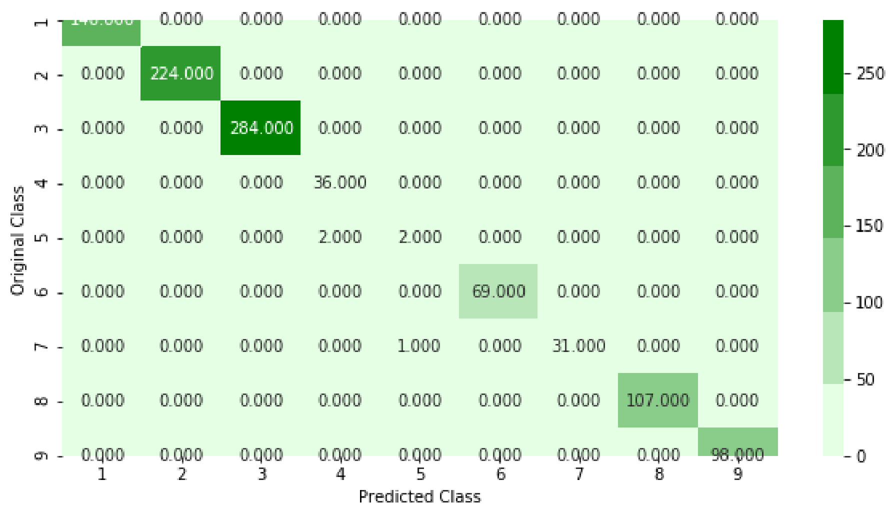
```
predict_y = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

```
The train log loss is: 0.010560251581961532
The cross validation log loss is: 0.015139740153308321
The test log loss is: 0.018675097082944743
```

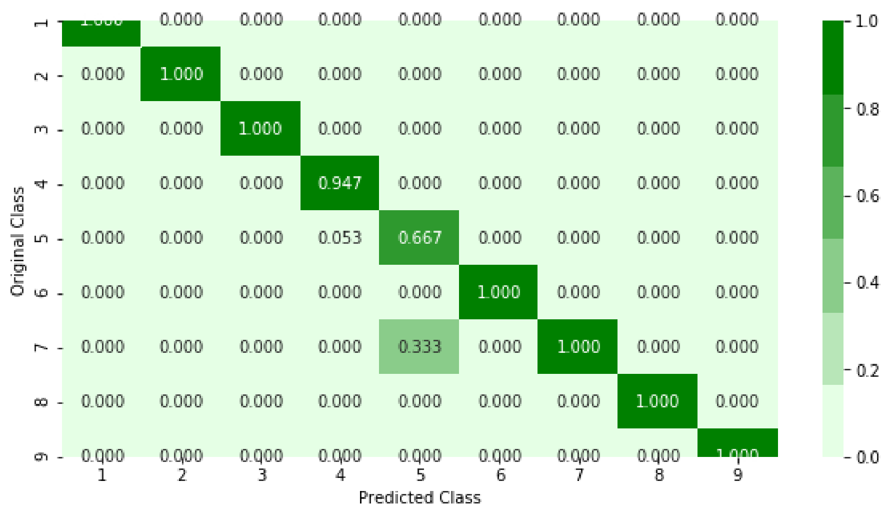
In [24]:

```
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

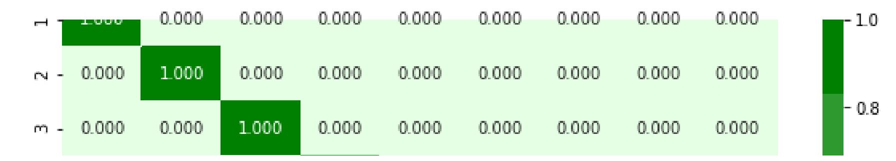
```
Number of misclassified points  0.3
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

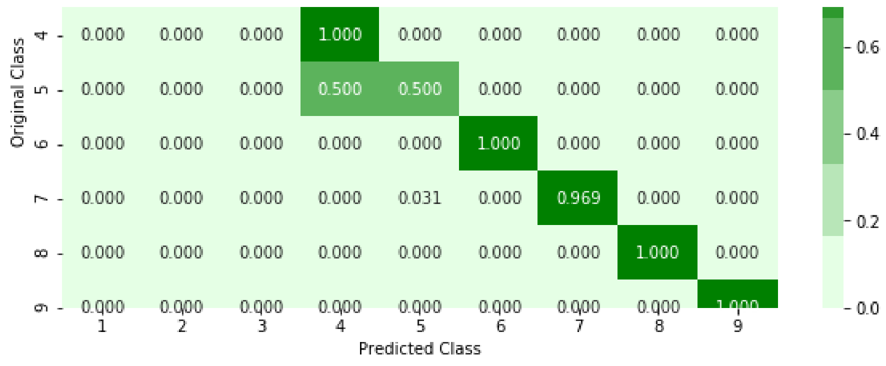


```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```



```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ---------------------------------
-------------
```





```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

8. Conclusion

In [1]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Train log-loss","CV log-loss","Test log-loss"]

x.add_row(["Model 1", 0.0228, 0.0472,0.0537])
x.add_row(["Model 2", 0.0209, 0.0537,0.0330])
x.add_row(["Model 3", 0.0210, 0.0575,0.0342])
x.add_row(["Model 4", 0.0189, 0.0597,0.0538])
x.add_row(["Model 5", 0.0210, 0.0493,0.0388])
x.add_row(["Model 6", 0.0196, 0.0384,0.0373])
x.add_row(["Model 7", 0.0185, 0.0287,0.0311])
x.add_row(["Model 8", 0.0186, 0.0304,0.0338])
x.add_row(["Model 9", 0.0185, 0.0287,0.0311])
x.add_row(["Model 10", 0.0184, 0.0285,0.0310])
```

In [2]:

```
print(x)
```

```
+----------+----------------+-------------+---------------+
|  Model   | Train log-loss | CV log-loss | Test log-loss |
+----------+----------------+-------------+---------------+
| Model 1  |     0.0228     |    0.0472   |     0.0537    |
| Model 2  |     0.0209     |    0.0537   |     0.033     |
| Model 3  |     0.021      |    0.0575   |     0.0342    |
| Model 4  |     0.0189     |    0.0597   |     0.0538    |
| Model 5  |     0.021      |    0.0493   |     0.0388    |
| Model 6  |     0.0196     |    0.0384   |     0.0373    |
| Model 7  |     0.0185     |    0.0287   |     0.0311    |
| Model 8  |     0.0186     |    0.0304   |     0.0338    |
| Model 9  |     0.0185     |    0.0287   |     0.0311    |
| Model 10 |     0.0184     |    0.0285   |     0.031     |
+----------+----------------+-------------+---------------+
```

In [21]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","n_estimators","max_depth", "Train log-loss","CV log-loss","Test log-loss"
]

x.add_row(["Model with CV \nlogloss <= 0.01", 3000,15,0.0105, 0.0151,0.0186])
```

In [22]:

```
print(x)
```

```
+------------------+--------------+-----------+----------------+-------------+---------------+
|      Model       | n_estimators | max_depth | Train log-loss | CV log-loss | Test log-loss |
+------------------+--------------+-----------+----------------+-------------+---------------+
|  Model with CV   |     3000     |     15    |     0.0105     |    0.0151   |     0.0186    |
| logloss <= 0.01  |              |           |                |             |               |
+------------------+--------------+-----------+----------------+-------------+---------------+
```

Step by Step procedure

- First downloaded the zip file which has total 10,868 .bytes files and 10,868 asm files total 21,736 files
- Here in this assignment i have used 5K points of .bytes and .asm each
- Then separated .bytes and .asm into different folders.
- Extracted features(size of file,byte unigrams and asm unigrams) from both the files.
- Then did modelling on .byte and .asm features seperately using KNN,Logistic Regression,Random Forest,Xgboost.
- Then combined both features and did modelling.
- Extracted more features to reduce the logloss.
- Extra features extracted are byte bigrams,asm opcodes bigram and image features from byte files.
- byte bigrams and image features worked very well after training Xgboost on it and helped in reducing the logloss to less than equal to 0.01.

In [ ]:

```

```

In [ ]:

```

```