Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

The term malware is a contraction of malicious software. Put simply, malware is any piece of software that was written with the intent of doing harm to data, devices or to people.
Source: https://www.avg.com/en/signal/what-is-malware

1.2. Problem Statement

In the past few years, the malware industry has grown very rapidly that, the syndicates invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust softwares to detect and terminate these attacks. The major part of protecting a computer system from a malware attack is to **identify whether a given piece of file/software is a malware.**

1.3 Source/Useful Links

Microsoft has been very active in building anti-malware products over the years and it runs it's anti-malware utilities over **150 million computers** around the world. This generates tens of millions of daily data points to be analyzed as potential malware. In order to be effective in analyzing and classifying such large amounts of data, we need to be able to group them into groups and identify their respective families.

This dataset provided by Microsoft contains about 9 classes of malware. ,

Source: https://www.kaggle.com/c/malware-classification

1.4. Real-world/Business objectives and constraints.

1. Minimize multi-class error.
2. Multi-class probability estimates.
3. Malware detection should not take hours and block the user's computer. It should fininsh in a few seconds or a minute.

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- Source : https://www.kaggle.com/c/malware-classification/data
- For every malware, we have two files
  1. .asm file (read more: https://www.reviversoft.com/file-extensions/asm)
  2. .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)
- Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:
- **Lots of Data for a single-box/computer.**
- There are total 10,868 .bytes files and 10,868 asm files total 21,736 files
- There are 9 types of malwares (9 classes) in our give data
- Types of Malware:
  1. Ramnit
  2. Lollipop
  3. Kelihos_ver3
  4. Vundo
  5. Simda
  6. Tracur
  7. Kelihos_ver1
  8. Obfuscator.ACY
  9. Gatak

2.1.2. Example Data Point

.asm file

```
.text:00401000                                          assume es:nothing, ss:nothing, ds:_data,    fs:not
hing, gs:nothing
.text:00401000 56                                       push    esi
.text:00401001 8D 44 24    08                                   lea     eax, [esp+8]
.text:00401005 50                                       push    eax
.text:00401006 8B F1                                       mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                                   call    ??0exception@std@@QAE@ABQBD@Z ;
std::exception::exception(char const * const &)
.text:0040100D C7 06 08    BB 42 00                               mov     dword ptr [esi],    offset off_42
BB08
.text:00401013 8B C6                                       mov     eax, esi
.text:00401015 5E                                       pop     esi
.text:00401016 C2 04 00                                       retn    4
.text:00401016                                  ; ---------------------------------------------------------
-------------------
.text:00401019 CC CC CC    CC CC CC CC                               align 10h
.text:00401020 C7 01 08    BB 42 00                               mov     dword ptr [ecx],    offset off_42
BB08
.text:00401026 E9 26 1C    00 00                                   jmp     sub_402C51
.text:00401026                                  ; ---------------------------------------------------------
-------------------
.text:0040102B CC CC CC    CC CC                                   align 10h
.text:00401030 56                                       push    esi
.text:00401031 8B F1                                       mov     esi, ecx
.text:00401033 C7 06 08    BB 42 00                               mov     dword ptr [esi],    offset off_42
BB08
.text:00401039 E8 13 1C    00 00                                   call    sub_402C51
.text:0040103E F6 44 24    08 01                                   test    byte ptr    [esp+8], 1
.text:00401043 74 09                                       jz      short loc_40104E
.text:00401045 56                                       push    esi
.text:00401046 E8 6C 1E    00 00                                   call    ??3@YAXPAX@Z    ; operator delet
e(void *)
.text:0040104B 83 C4 04                                       add     esp, 4
.text:0040104E
.text:0040104E                                  loc_40104E:                     ; CODE XREF: .text:00401043 j
.text:0040104E 8B C6                                       mov     eax, esi
```

```
.text:00401050 5E                                    pop     esi
.text:00401051 C2 04 00                                    retn    4
.text:00401051                              ; ---------------------------------------------------------
-------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 20 00 02 00 04
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
```

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

```
        There are nine different classes of malware that we need to classify a given a data point => Mu
lti class classification problem
```

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/malware-classification#evaluation (https://www.kaggle.com/c/malware-classification#evaluation)

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:
* Class probabilities are needed. * Penalize the errors in class probabilites => Metric is Log-loss. * Some Latency constraints.

2.3. Train and Test Dataset

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

In [ ]:

```

```

3. Exploratory Data Analysis

In [40]:

```
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

from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import normalize
from mlxtend.classifier import StackingClassifier
from lightgbm import LGBMClassifier
```

In [ ]:

```

```

In [2]:
```
#separating byte files and asm files

source = 'train'
destination = 'byteFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a folder with the same name
if not os.path.isdir(destination):
    os.makedirs(destination)

# if we have folder called 'train' (train folder contains both .asm files and .bytes files) we will rename it '
# for every file that we have in our 'asmFiles' directory we check if it is ending with .bytes, if yes we will 
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    os.rename(source,'asmFiles')
    source='asmFiles'
    data_files = os.listdir(source)
    for file in data_files:
        print(file)
        if (file.endswith("bytes")):
            shutil.move(source+ "/" +file,destination)
```

```
dKeYscvQ1KZX9EZAJ75M.bytes
cSMXzpQ2q4nCy5UfPBg9.asm
dexOVwSPEDv4AYR8f3bI.asm
DhzFERM3B61lSmNP2JTZ.bytes
gdpCryb5PsOv4TzWcLHQ.asm
a84udcisW2mPRvrSFk0w.asm
Do9QfaXw52dYzcFipUeq.bytes
gak4Zc3ztRCB7NDUIXh5.bytes
4jVLlkxAIGvb3MBzDYHc.asm
86QrjZewznD2W3VhpRbm.bytes
djy2nxpL3gDSzf4G01vw.bytes
i5u2KDJ9t0OyAdokafj7.bytes
ieTyx3pGN70aXrcqwFu4.asm
hQnAcOfHYisDkINaod7L.asm
2p9Dqri6aAzO5yVhQSLX.asm
caL7sn2qd4JwxlrpR0BP.bytes
jTgsFer9LQikYJ5aXBZR.bytes
K86VgF4pZnPzHeSkqhtG.bytes
4LNpxlPiRBTqZy0sEaMY.asm
5su2fIARtLgUzSdOgDB9.asm
```

3.1. Distribution of malware classes in whole data set

In [2]:
```
Y=pd.read_csv("trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

#put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
ax.yaxis.set_ticks(np.linspace(0, total, 11))

#adjust the ticklabel to the desired format, without changing the position of the ticks.
ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
plt.show()
```

3.2. Feature extraction

3.2.1 File size of byte files as a feature

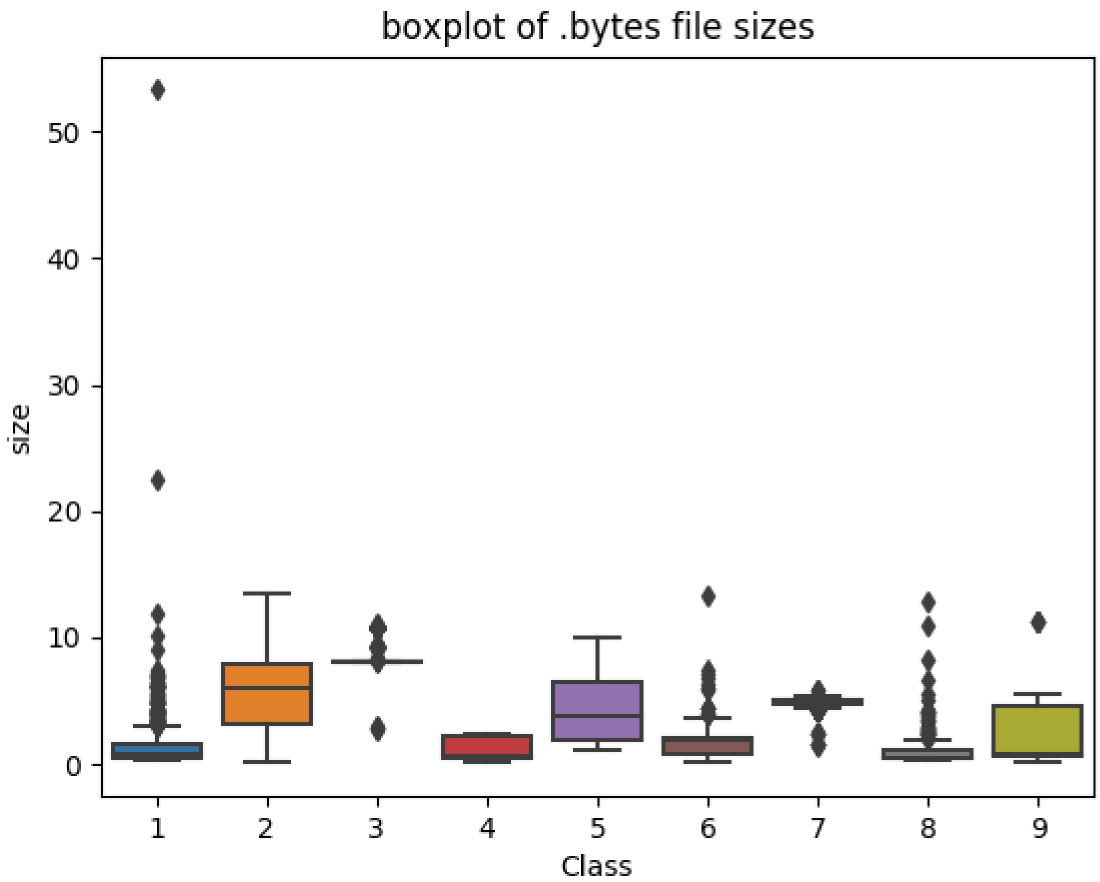
In [3]:
```
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

```
   Class                    ID      size
0      1  F6WTdXrgLfco1s2PUlyH  1.741699
1      2  FIVR08jS5sZo6avbmPHk  1.996582
2      8  9LWNpGBmUctnVraSis5j  0.438965
3      9  hcug1H8Rw6ZJ2sOITmnG  0.594727
4      7  dLKYot9Ix2Bib1DZerGg  5.055176
```

3.2.2 box plots of file size (.byte files) feature

In [24]:
```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

```
<IPython.core.display.Javascript object>
```



3.2.3 feature extraction from byte files

In [25]:
```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        file=file.split('.')[0]
        text_file = open('byteFiles/'+file+".txt", 'w+')
        with open('byteFiles/'+file+".bytes","r") as fp:
            lines=""
            for line in fp:
                a=line.rstrip().split(" ")[1:]
                b=' '.join(a)
                b=b+"\n"
                text_file.write(b)
            fp.close()
            os.remove('byteFiles/'+file+".bytes")
        text_file.close()

files = os.listdir('byteFiles')
filenames2=[]
feature_matrix = np.zeros((len(files),257),dtype=int)
k=0


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,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
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
```

```
                for lines in byte_flie:
                    line=lines.rstrip().split(" ")
                    for hex_code in line:
                        if hex_code=='??':
                            feature_matrix[k][256]+=1
                        else:
                            feature_matrix[k][int(hex_code,16)]+=1
            byte_flie.close()
        for i, row in enumerate(feature_matrix[k]):
            if i!=len(feature_matrix[k])-1:
                byte_feature_file.write(str(row)+",")
            else:
                byte_feature_file.write(str(row))
        byte_feature_file.write("\n")

        k += 1

byte_feature_file.close()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-25-de64919173c9> in <module>
     15         with open('byteFiles/'+file+".bytes","r") as fp:
     16             lines=""
---> 17             for line in fp:
     18                 a=line.rstrip().split(" ")[1:]
     19                 b=' '.join(a)

/usr/lib/python3.5/codecs.py in decode(self, input, final)
    316         raise NotImplementedError
    317
--> 318     def decode(self, input, final=False):
    319         # decode input (taking the buffer into account)
    320         data = self.buffer + input

KeyboardInterrupt:
```

In [ ]:

```
byte_features=pd.read_csv("result.csv")
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)
```

In [27]:
```
data_size_byte.head(2)
```

Out[27]:

	Class	ID	size
0	1	F6WTdXrgLfco1s2PUlyH	1.741699
1	2	FIVR08jS5sZo6avbmPHk	1.996582

In [ ]:
```
byte_features_with_size = byte_features.merge(data_size_byte, on='ID')
byte_features_with_size.to_csv("result_with_size.csv")
byte_features_with_size.head(2)
```

In [13]:
```
byte_features_with_size = pd.read_csv("result_with_size.csv")
```

In [152]:
```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
result = normalize(byte_features_with_size)
```

In [15]:
```
result.head(2)
```

Out[15]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9	fa	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.01356	0.013107	0.0136
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.00192	0.001147	0.0013

2 rows × 261 columns

In [16]:
```
data_y = result['Class']
result.head()
```

Out[16]:

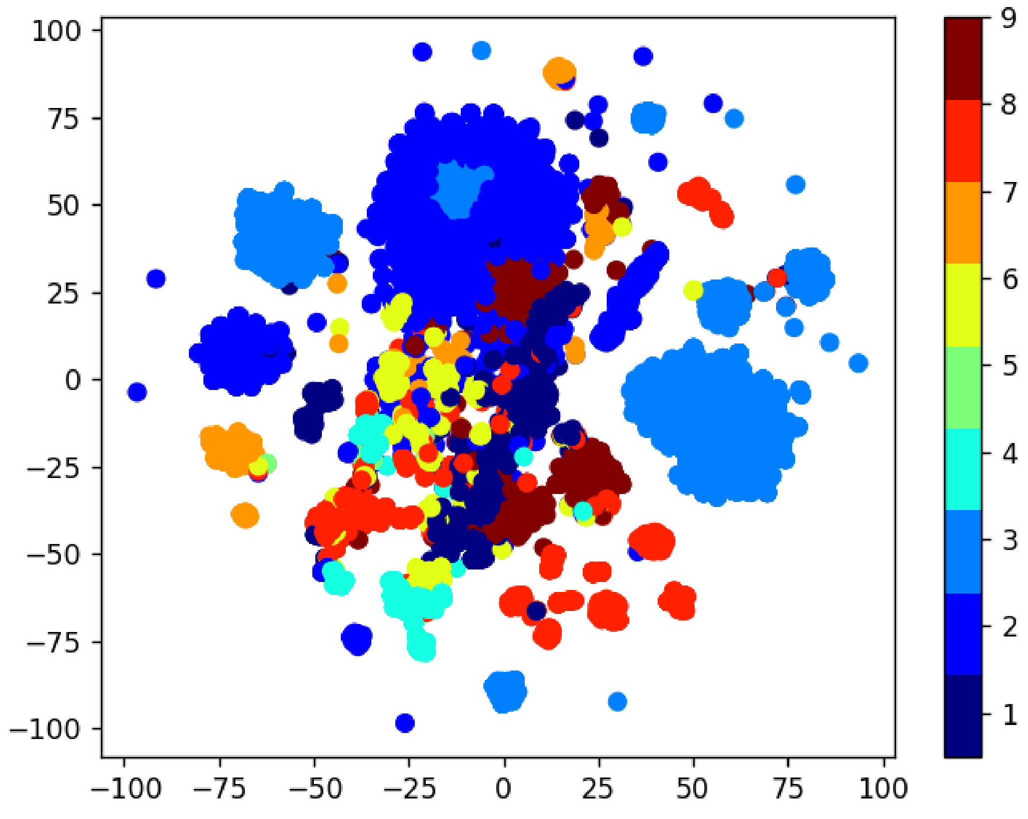
	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9	fa	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.013560	0.013107	0.01
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.001920	0.001147	0.00
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	...	0.009804	0.011777	0.01
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	...	0.002121	0.001886	0.00
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	...	0.001530	0.000853	0.00

5 rows × 261 columns

3.2.4 Multivariate Analysis

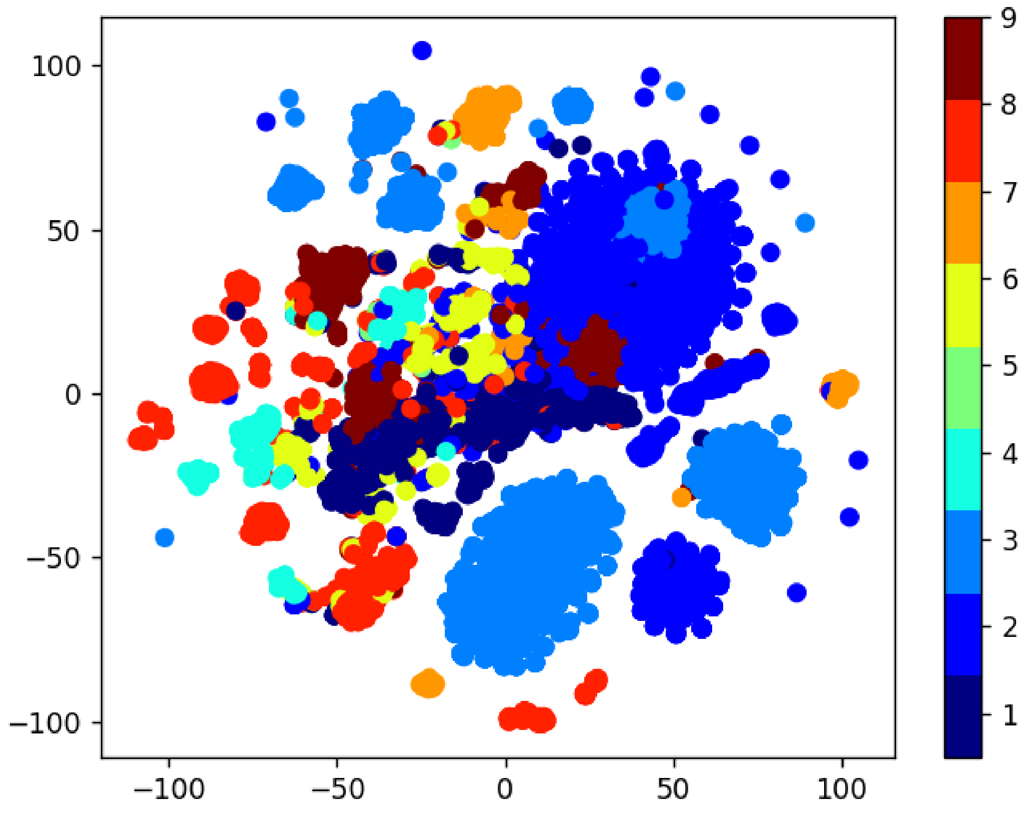
In [58]:
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

<IPython.core.display.Javascript object>



In [59]:
```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

```
<IPython.core.display.Javascript object>
```



Train Test split

In [60]:
```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_t
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), data_y,stratify=data_y
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_t
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

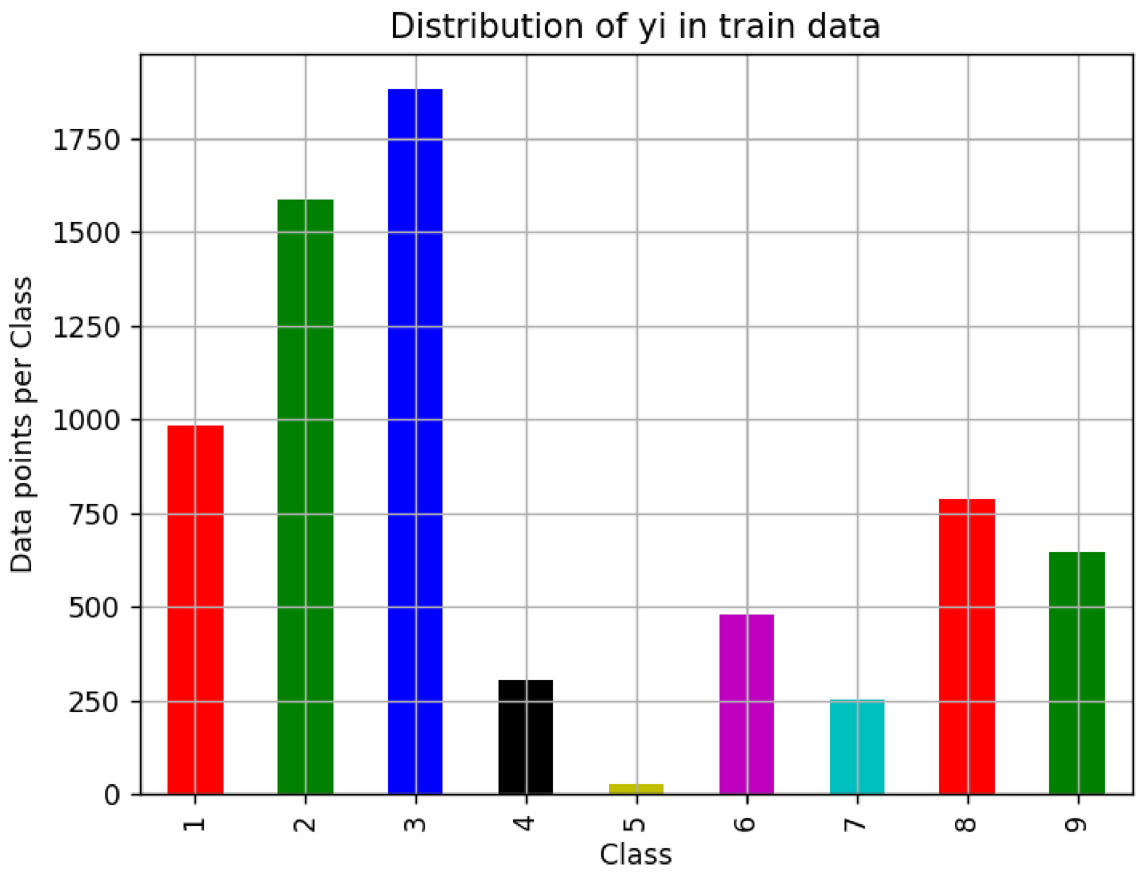
In [63]:
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

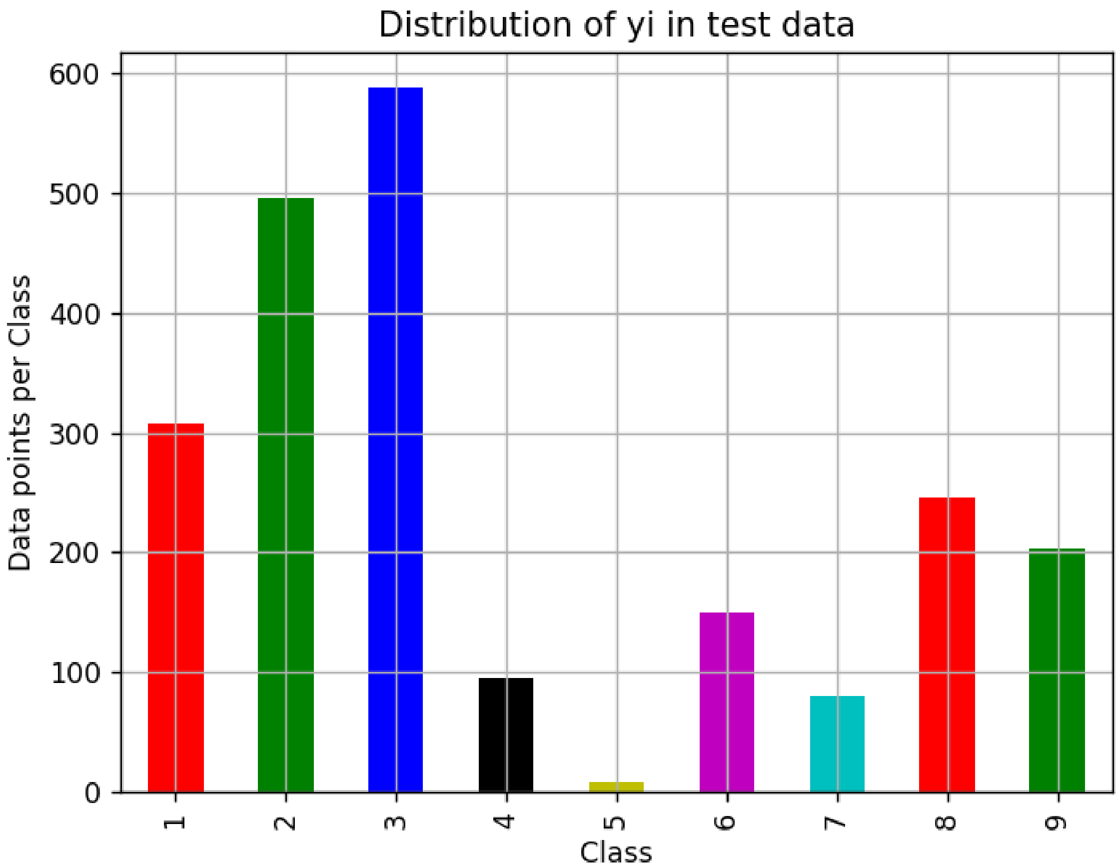
In [67]:
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar', color=my_colors)
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.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_c


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
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.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_cla

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
```

```
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_d
```

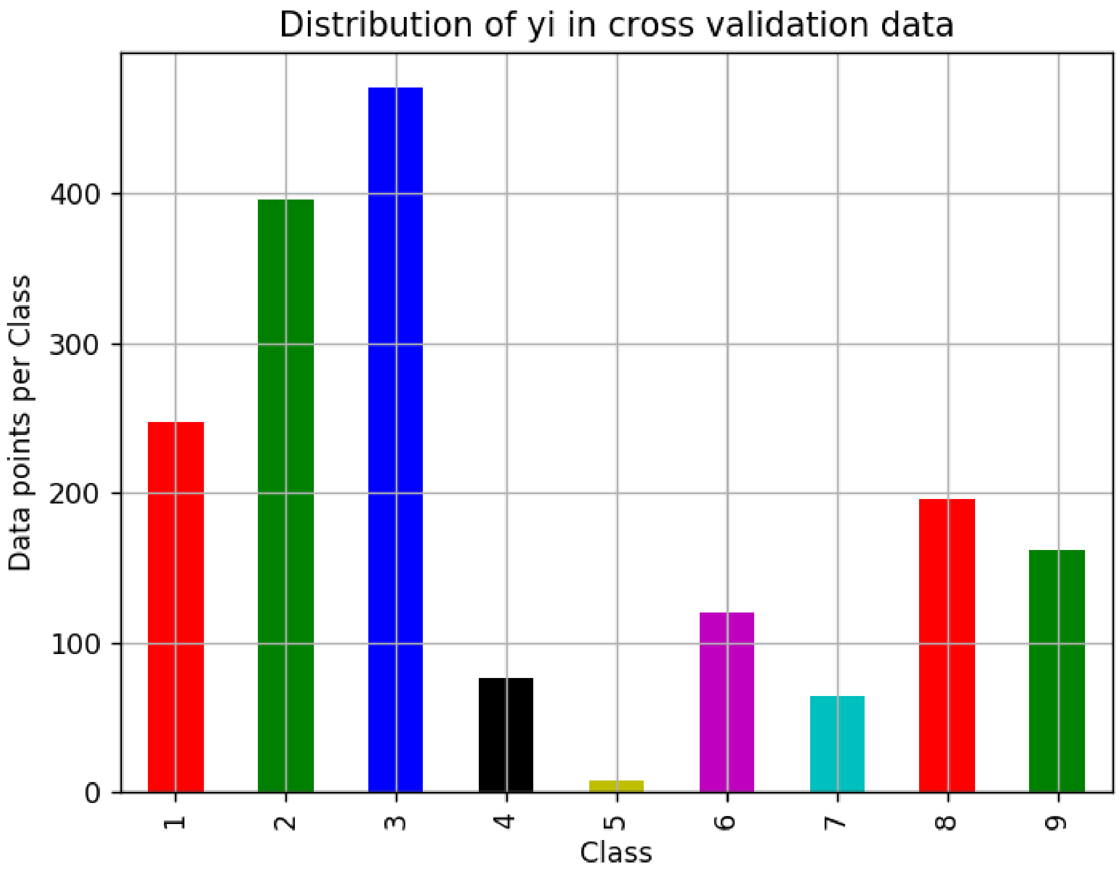
```
<IPython.core.display.Javascript object>
```



```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
--------------------------------------------------------------------------------

<IPython.core.display.Javascript object>
```




```
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
--------------------------------------------------------------------------------

<IPython.core.display.Javascript object>
```


Distribution of yi in cross validation data
Data points per Class
0
100
200
300
400
Class
1
2
3
4
5
6
7
8
9

```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

In [61]:
```
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))
```

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

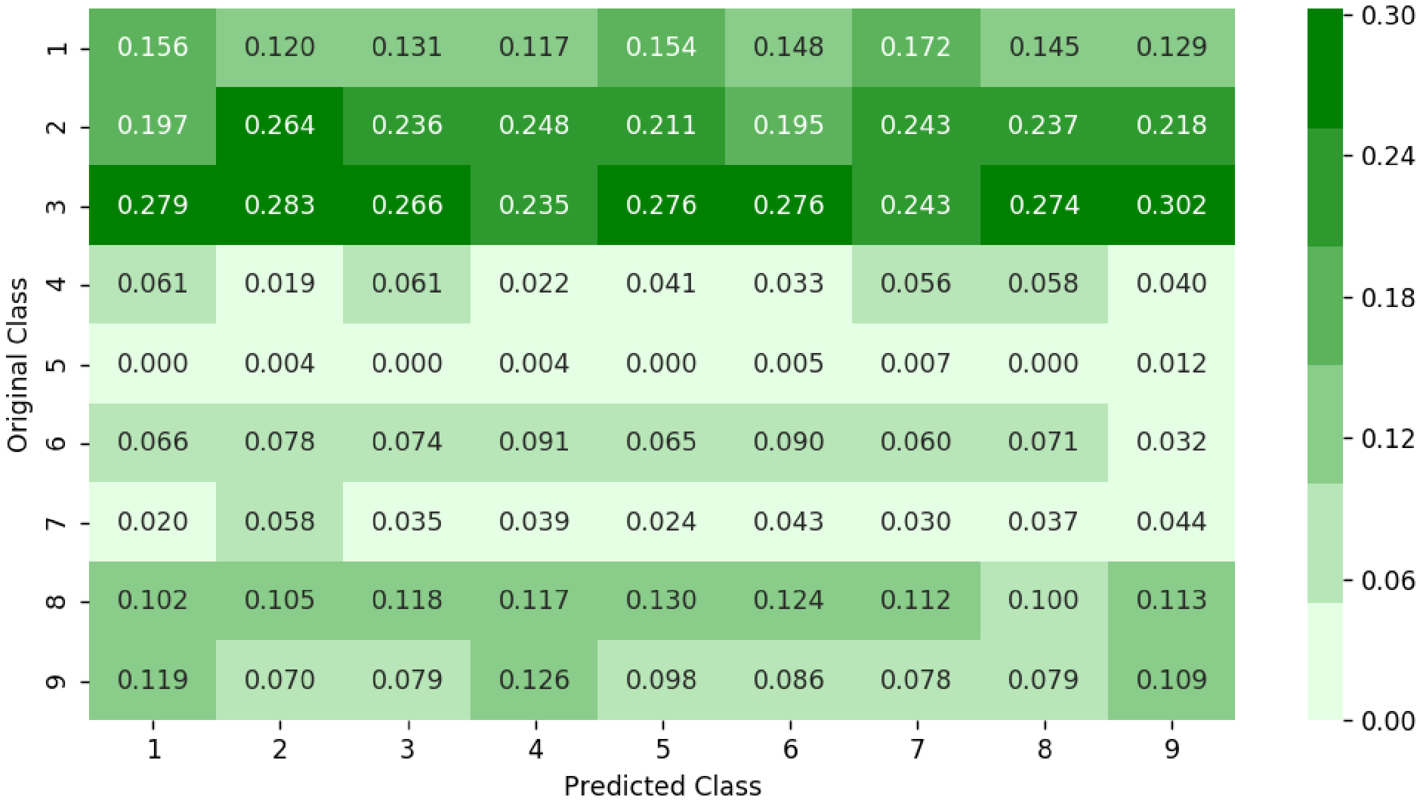
In [62]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.45615644965
Log loss on Test Data using Random Model 2.48503905509
Number of misclassified points  88.5004599816
-------------------------------------------------- Confusion matrix ---------------------------------------------
---------

<IPython.core.display.Javascript object>
```

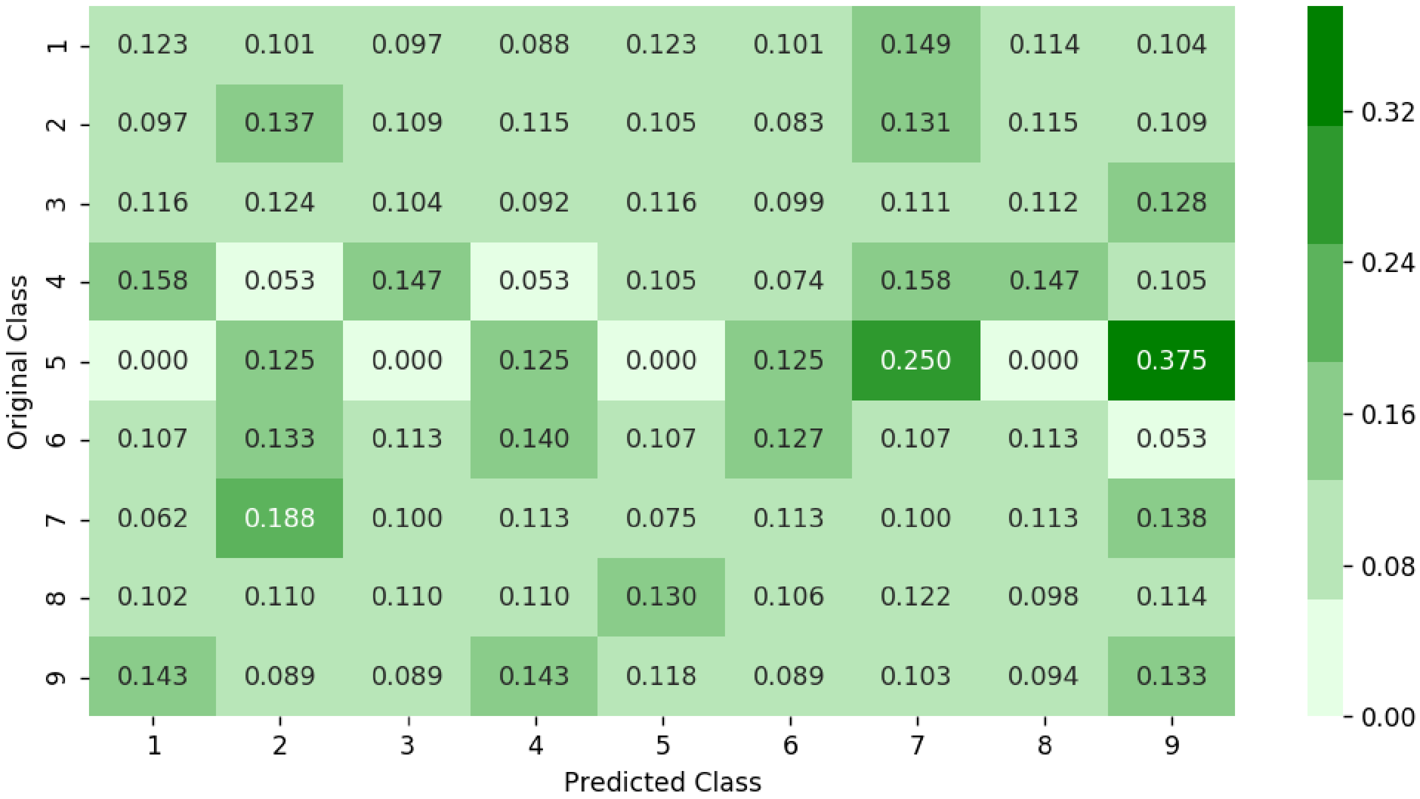

```
-------------------------------------------------- Precision matrix --------------------------------------------------
```

```
<IPython.core.display.Javascript object>
```




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

```
-------------------------------------------------- Recall matrix ---------------------------------------------------
------

<IPython.core.display.Javascript object>
```



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

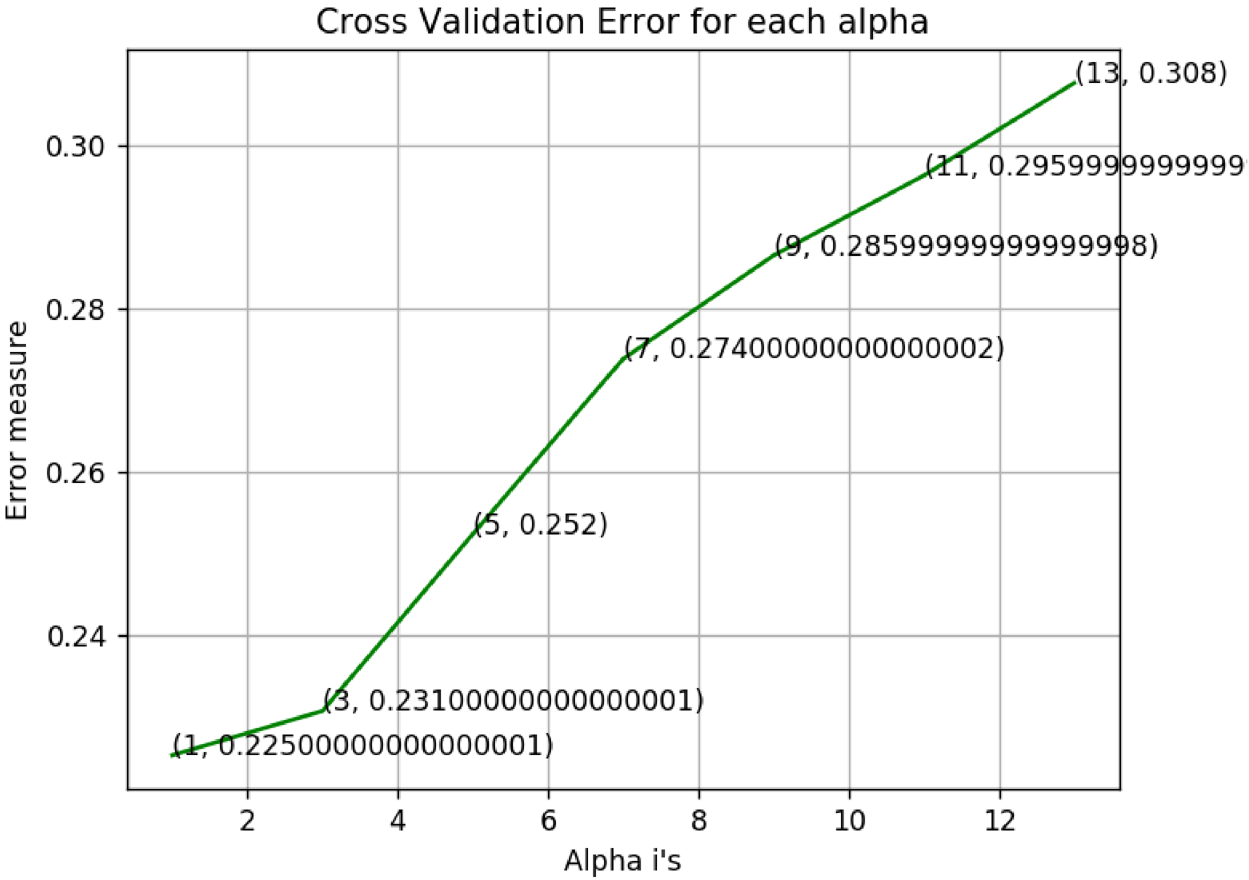
4.1.2. K Nearest Neighbour Classification

In [68]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbor
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geome
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calib
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)  Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])
```

```
best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

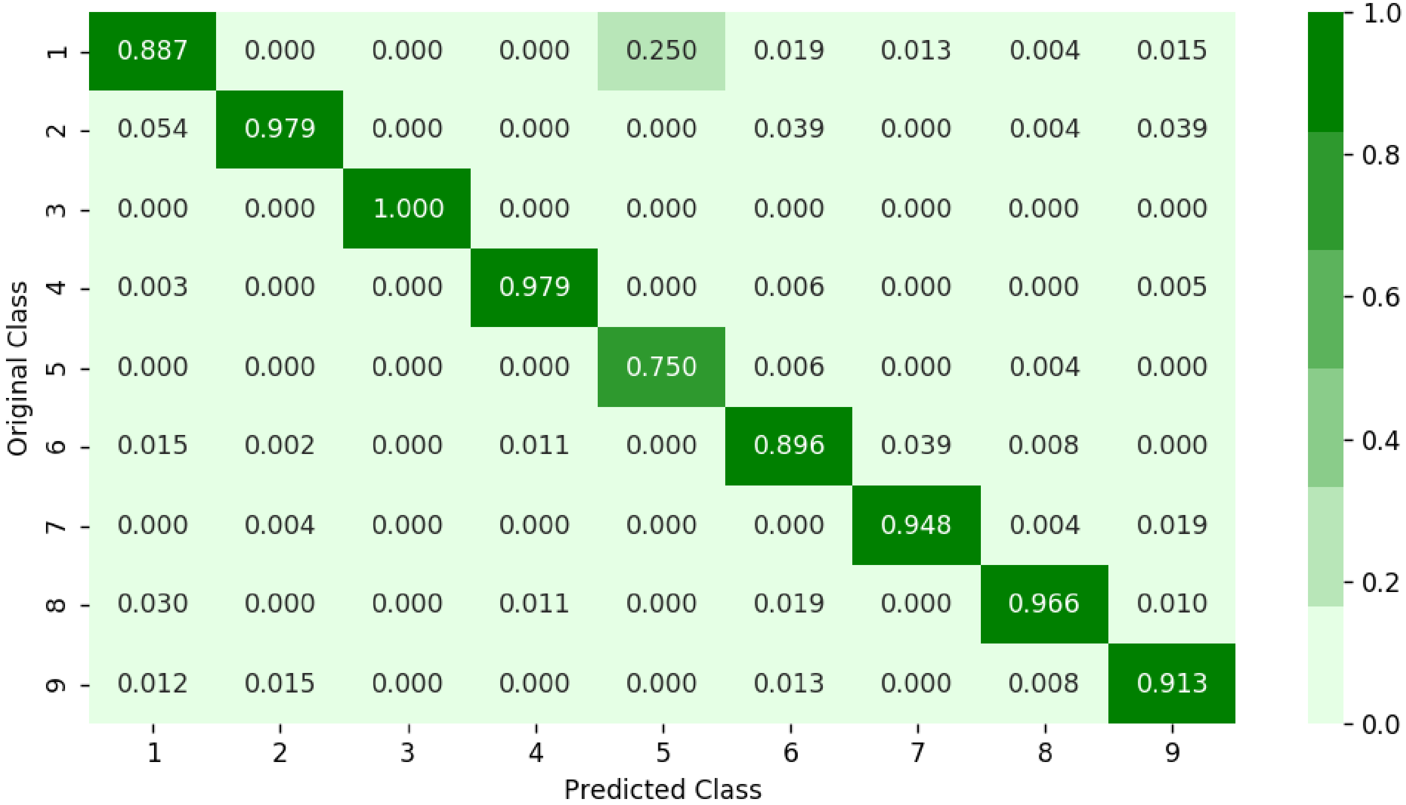
```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154
```

```
<IPython.core.display.Javascript object>
```

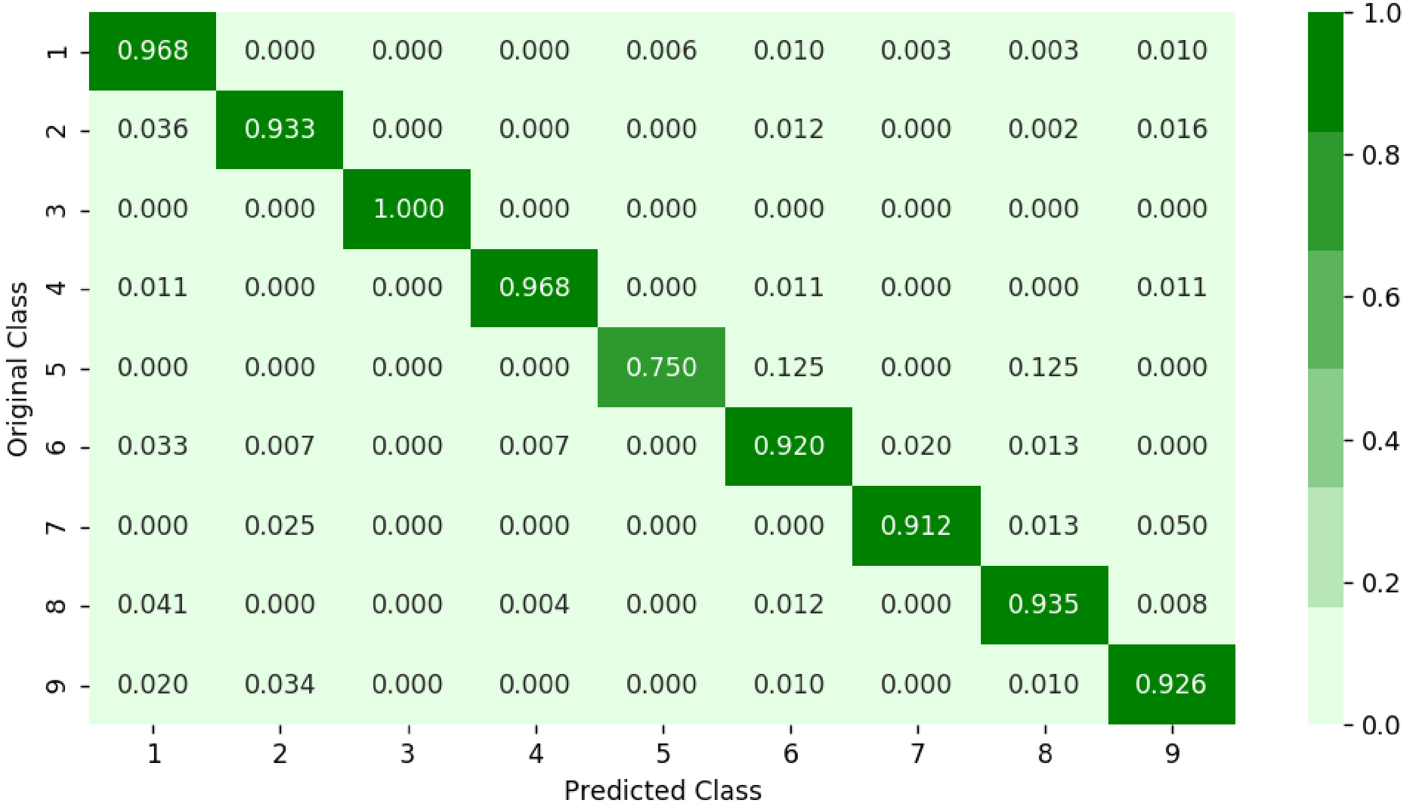



```
For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.225386237304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
-------------------------------------------------- Confusion matrix -----------------------------------------
---------

<IPython.core.display.Javascript object>
```


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

<IPython.core.display.Javascript object>
```


Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
0.887 0.000 0.000 0.000 0.250 0.019 0.013 0.004 0.015
0.054 0.979 0.000 0.000 0.000 0.039 0.000 0.004 0.039
0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000
0.003 0.000 0.000 0.979 0.000 0.006 0.000 0.000 0.005
0.000 0.000 0.000 0.000 0.750 0.006 0.000 0.004 0.000
0.015 0.002 0.000 0.011 0.000 0.896 0.039 0.008 0.000
0.000 0.004 0.000 0.000 0.000 0.000 0.948 0.004 0.019
0.030 0.000 0.000 0.011 0.000 0.019 0.000 0.966 0.010
0.012 0.015 0.000 0.000 0.000 0.013 0.000 0.008 0.913
1.0 0.8 0.6 0.4 0.2 0.0

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

<IPython.core.display.Javascript object>
```




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

4.1.3. Logistic Regression

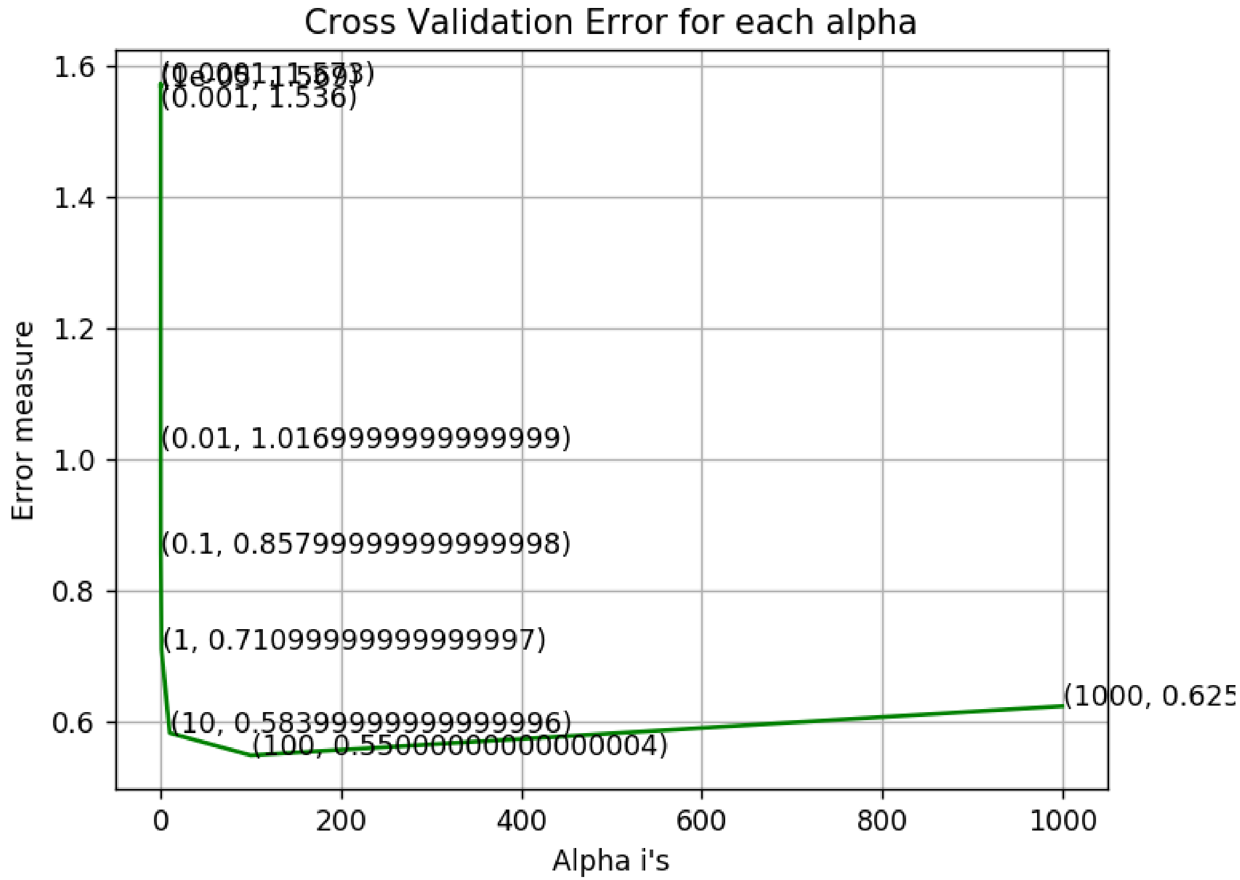
In [71]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDC
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------

alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
```

```python
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =  100 is 0.549929846589
log_loss for c =  1000 is 0.624746769121
```

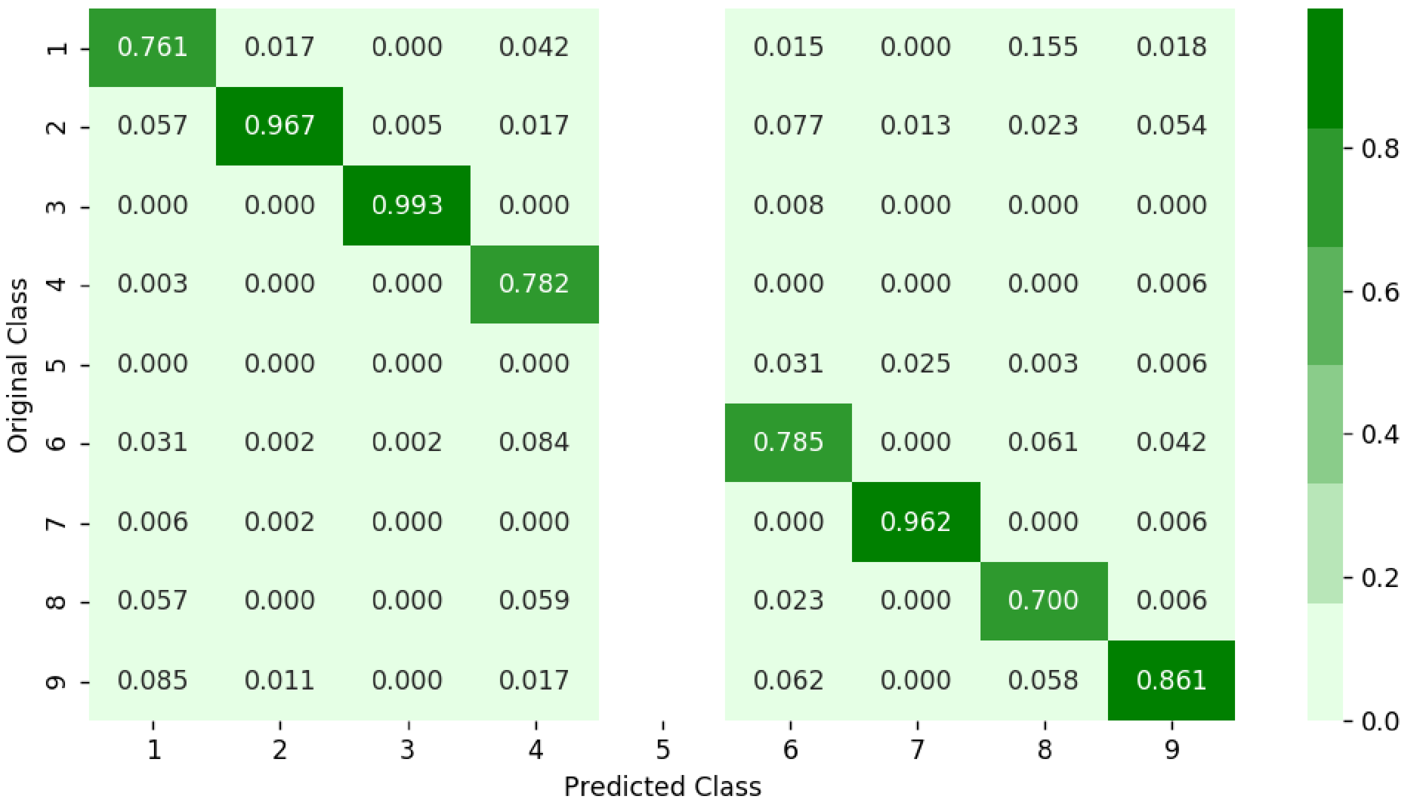
```
<IPython.core.display.Javascript object>
```



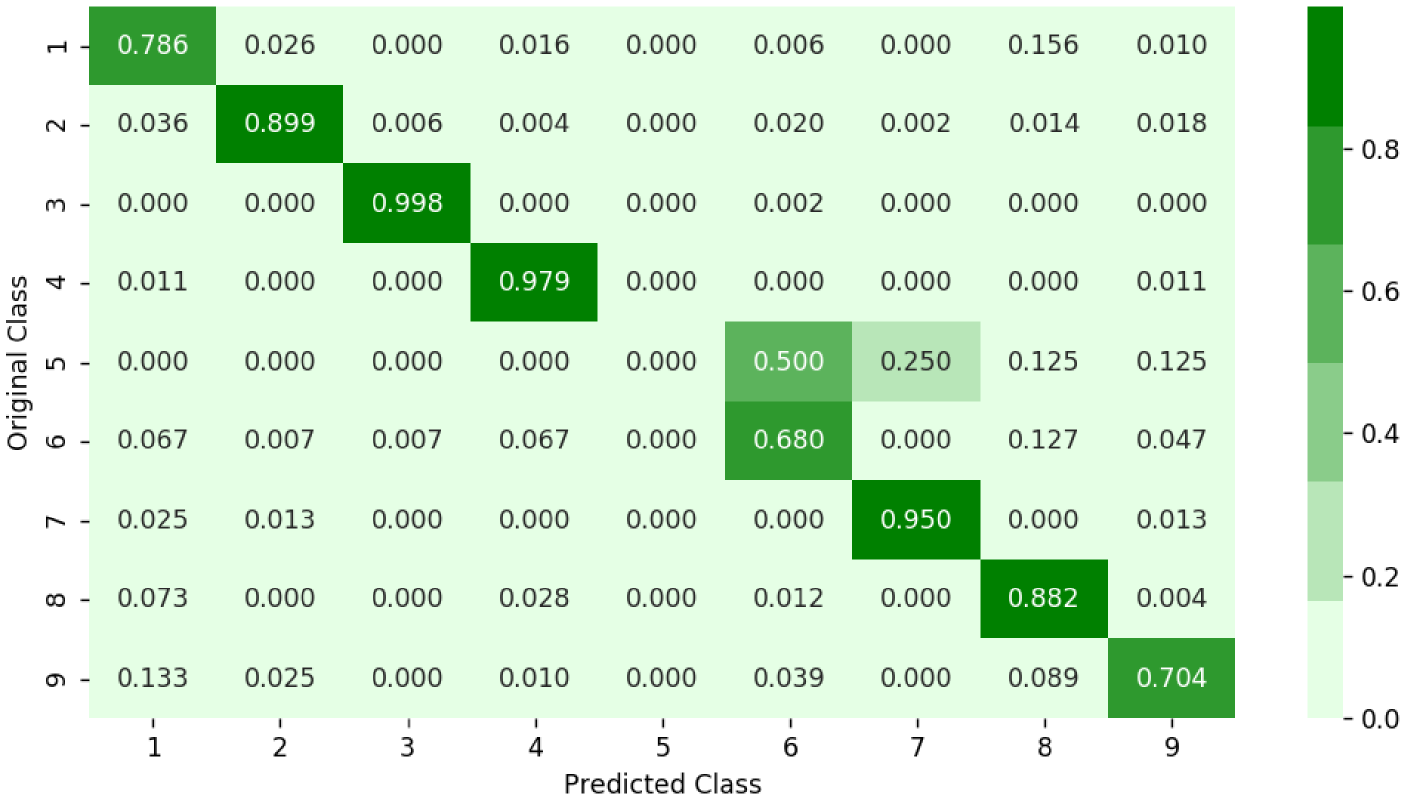

```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
-------------------------------------------------- Confusion matrix --------------------------------------------
---------

<IPython.core.display.Javascript object>
```

Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
1: 242.000 8.000 0.000 5.000 0.000 2.000 0.000 48.000 3.000
2: 18.000 446.000 3.000 2.000 0.000 10.000 1.000 7.000 9.000
3: 0.000 0.000 587.000 0.000 0.000 1.000 0.000 0.000 0.000
4: 1.000 0.000 0.000 93.000 0.000 0.000 0.000 0.000 1.000
5: 0.000 0.000 0.000 0.000 0.000 4.000 2.000 1.000 1.000
6: 10.000 1.000 1.000 10.000 0.000 102.000 0.000 19.000 7.000
7: 2.000 1.000 0.000 0.000 0.000 0.000 76.000 0.000 1.000
8: 18.000 0.000 0.000 7.000 0.000 3.000 0.000 217.000 1.000
9: 27.000 5.000 0.000 2.000 0.000 8.000 0.000 18.000 143.000
0
100
200
300
400
500

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

<IPython.core.display.Javascript object>
```




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

<IPython.core.display.Javascript object>
```




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

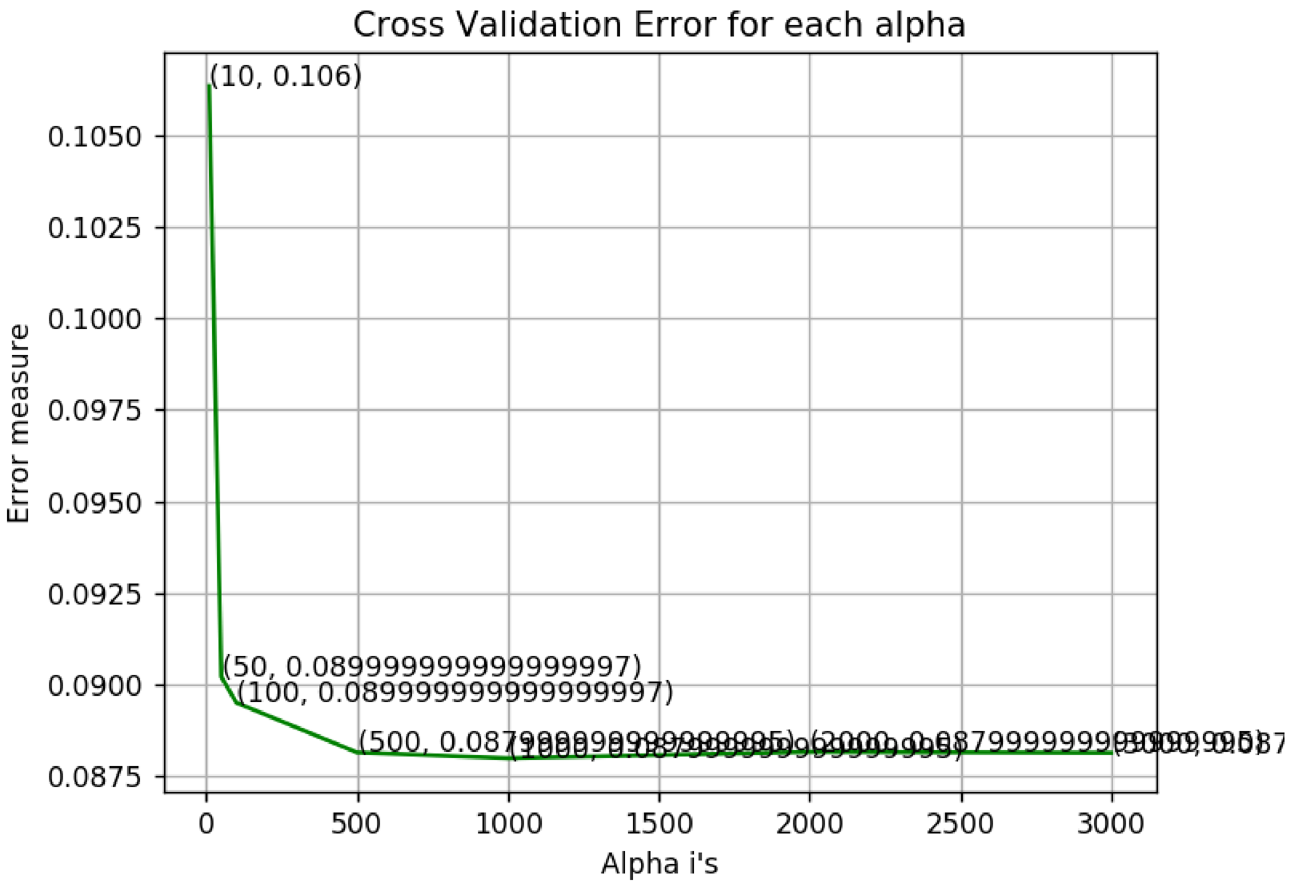
4.1.4. Random Forest Classifier

In [72]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decr
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-c
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
```

```
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

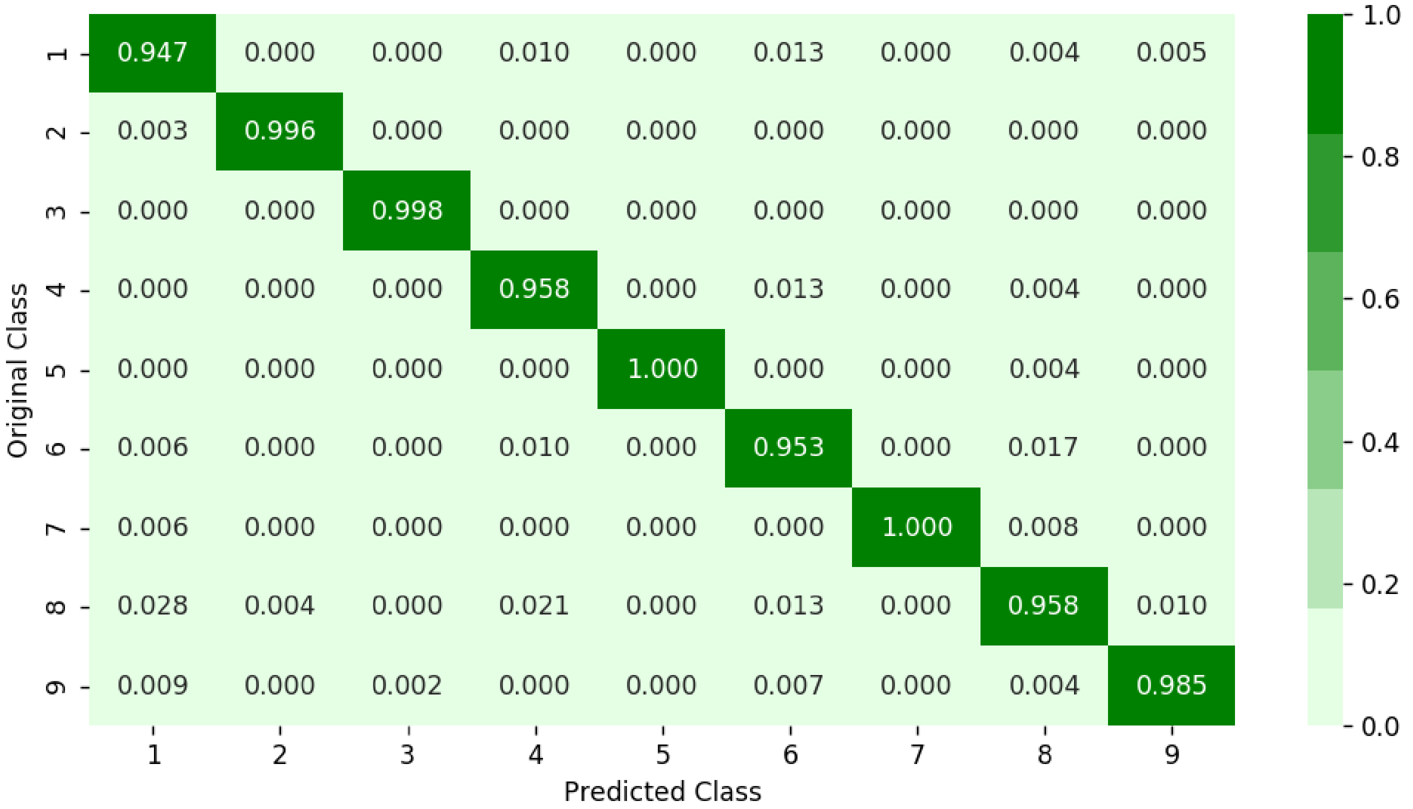
```
log_loss for c =  10 is 0.106357709164
log_loss for c =  50 is 0.0902124124145
log_loss for c =  100 is 0.0895043339776
log_loss for c =  500 is 0.0881420869288
log_loss for c =  1000 is 0.0879849524621
log_loss for c =  2000 is 0.0881566647295
log_loss for c =  3000 is 0.0881318948443
```

```
<IPython.core.display.Javascript object>
```

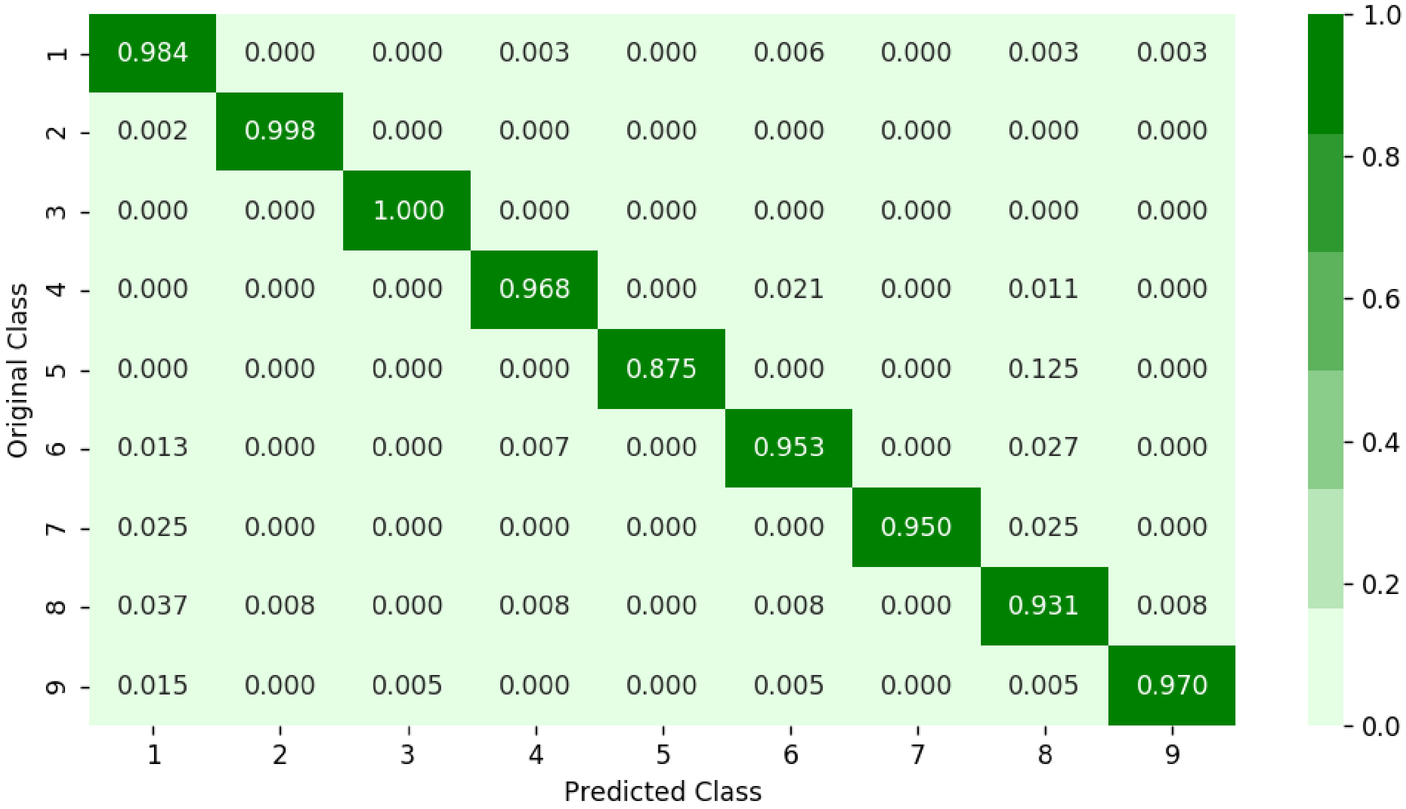



```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.0879849524621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
-------------------------------------------------- Confusion matrix ------------------------------------------------------

<IPython.core.display.Javascript object>
```


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

<IPython.core.display.Javascript object>
```




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

<IPython.core.display.Javascript object>
```



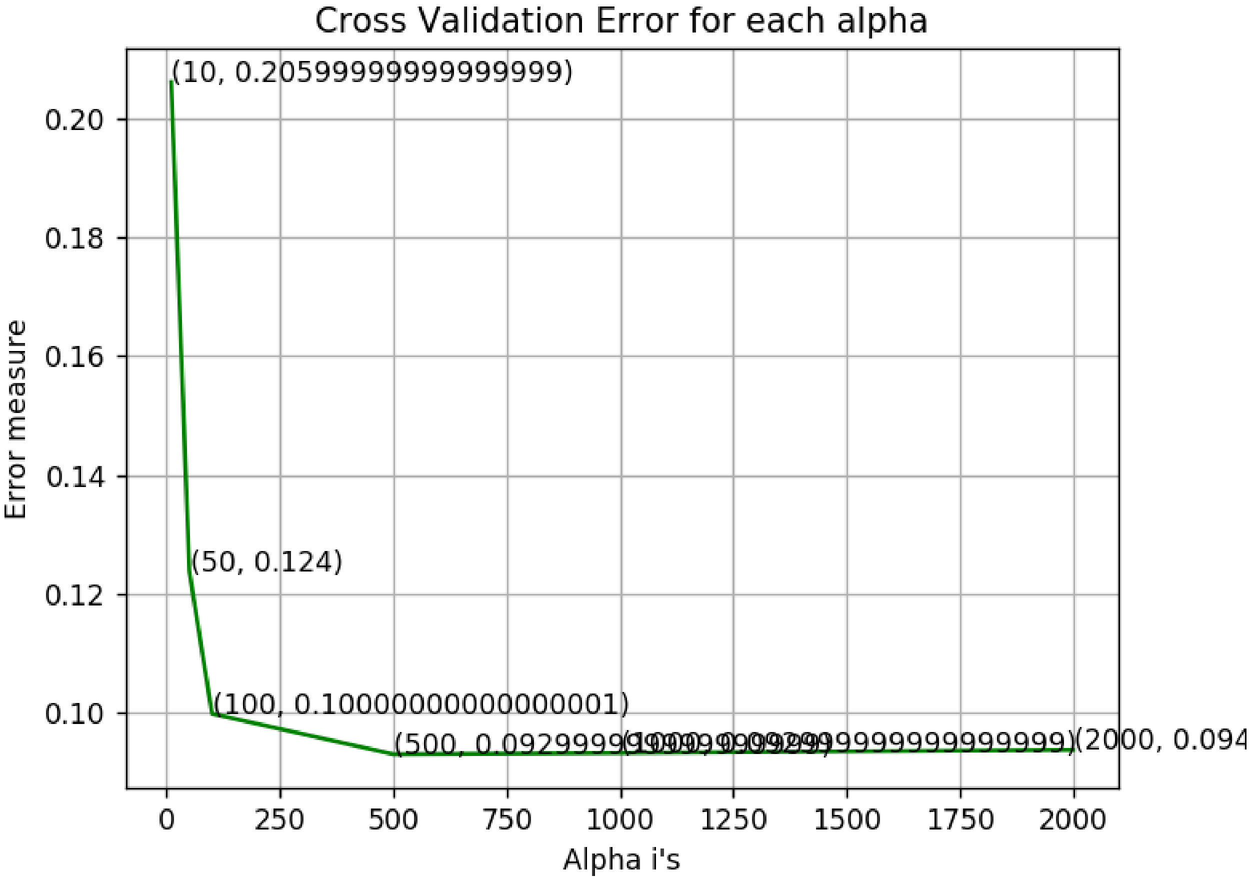

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.5. XgBoost Classification

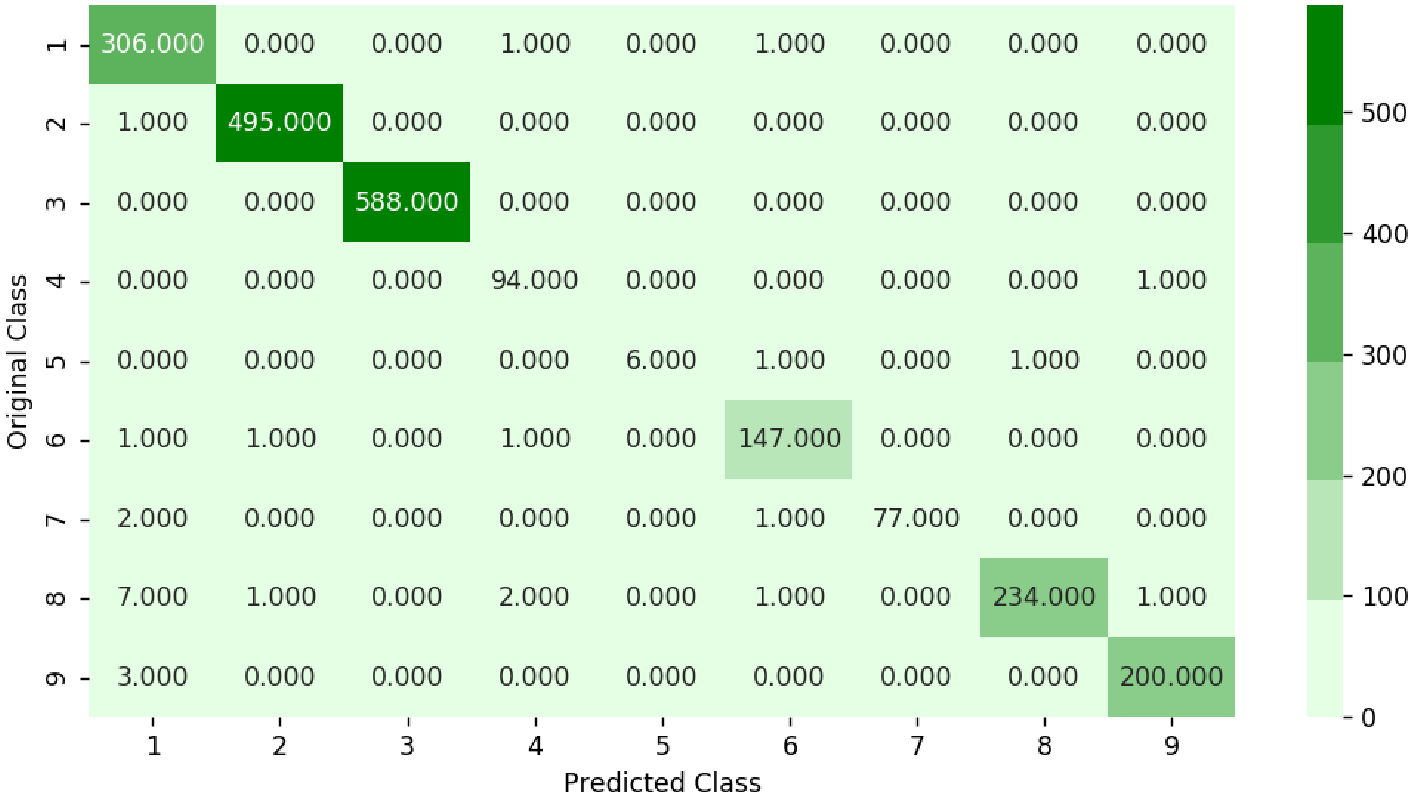
In [74]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#x
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decisio
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
```

```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.20615980494
log_loss for c =  50 is 0.123888382365
log_loss for c =  100 is 0.099919437112
log_loss for c =  500 is 0.0931035681289
log_loss for c =  1000 is 0.0933084876012
log_loss for c =  2000 is 0.0938395690309


<IPython.core.display.Javascript object>
```

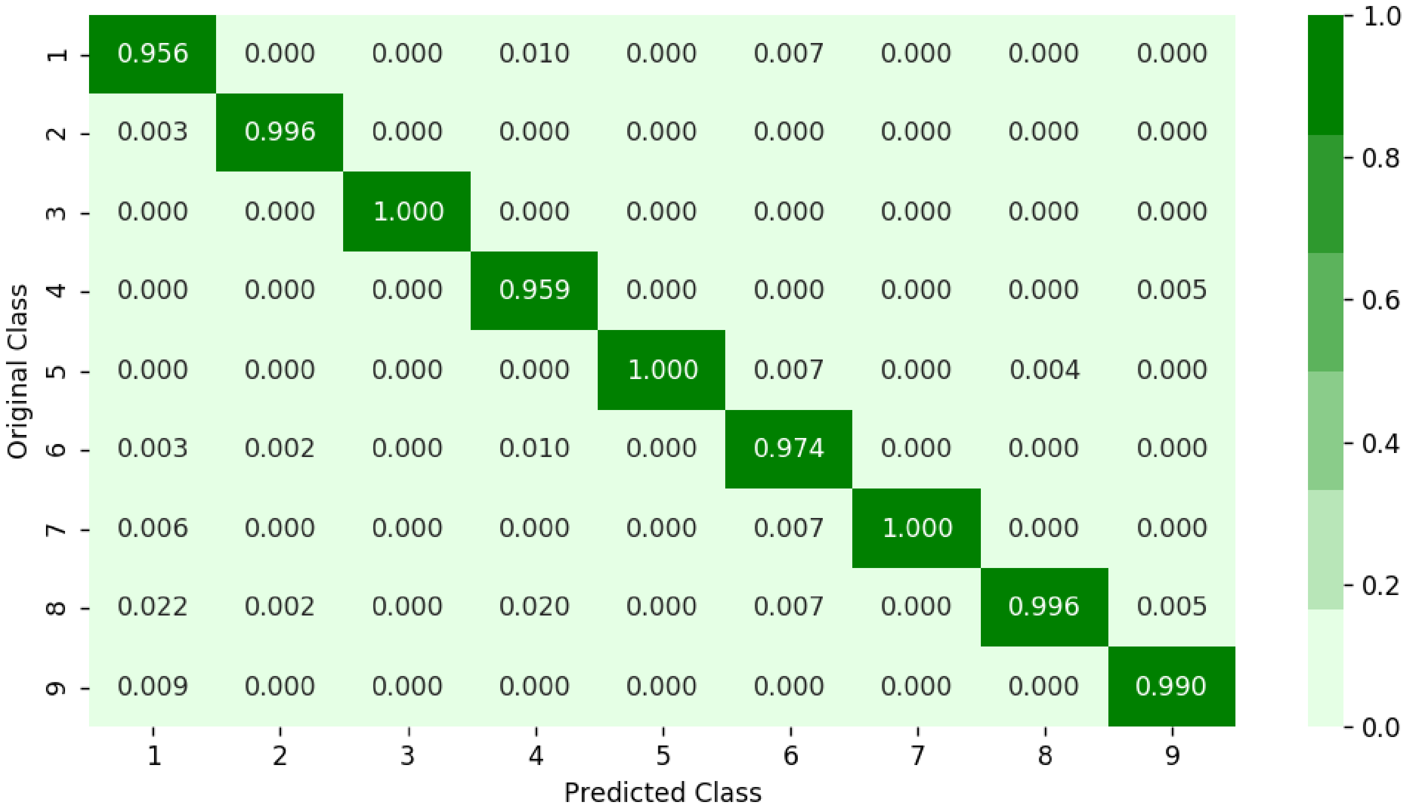



```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.0931035681289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
-------------------------------------------------- Confusion matrix -----------------------------------------
---------

<IPython.core.display.Javascript object>
```




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

```
<IPython.core.display.Javascript object>
```



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

<IPython.core.display.Javascript object>
```

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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [75]:

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl1.fit(X_train,y_train)
```

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  9.3min remaining:  5.4min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 10.1min remaining:  3.1min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 14.2min finished
```

Out[75]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=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, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 20
0, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3,
0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [76]:

```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 0.5}
```

In [80]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#x
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, max_depth=3)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.022540976086
cv loss 0.0928710624158
test loss 0.0782688587098
```

4.2 Modeling with .asm files

```
There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In parallel we can use all the cores
that are present in our computer.


Here we extracted 52 features from all the asm files which are important.

We read the top solutions and handpicked the features from those papers/videos/blogs.
 Refer:https://www.kaggle.com/c/malware-classification/discussion
```

4.2.1 Feature extraction from asm files

- To extract the unigram features from the .asm files we need to process ~150GB of data
- **Note: Below two cells will take lot of time (over 48 hours to complete)**
- We will provide you the output file of these two cells, which you can directly use it

In [ ]:
```
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='first'
folder_2 ='second'
folder_3 ='third'
folder_4 ='fourth'
folder_5 ='fifth'
folder_6 = 'output'
for i in [folder_1,folder_2,folder_3,folder_4,folder_5,folder_6]:
    if not os.path.isdir(i):
        os.makedirs(i)

source='train/'
files = os.listdir('train')
ID=df['Id'].tolist()
data=range(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'first')
    elif i%5==1:
        shutil.move(source+files[data[i]],'second')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'thrid')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fifth')
```

In [ ]:

```
#http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html

def firstprocess():
    #The prefixes tells about the segments that are present in the asm files
    #There are 450 segments(approx) present in all asm files.
    #this prefixes are best segments that gives us best values.
    #https://en.wikipedia.org/wiki/Data_segment

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.re
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'x
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
        with codecs.open('first/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                # https://www.tutorialspoint.com/python3/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
```

```
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


#same as above
def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.re
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'x
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
```

```
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.re
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'x
    keywords = ['.dll','std::',':dword']
```

```
registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
file1=open("output\largeasmfile.txt","w+")
files = os.listdir('thrid')
for f in files:
    prefixescount=np.zeros(len(prefixes),dtype=int)
    opcodescount=np.zeros(len(opcodes),dtype=int)
    keywordcount=np.zeros(len(keywords),dtype=int)
    registerscount=np.zeros(len(registers),dtype=int)
    features=[]
    f2=f.split('.')[0]
    file1.write(f2+",")
    opcodefile.write(f2+" ")
    with codecs.open('thrid/'+f,encoding='cp1252',errors ='replace') as fli:
        for lines in fli:
            line=lines.rstrip().split()
            l=line[0]
            for i in range(len(prefixes)):
                if prefixes[i] in line[0]:
                    prefixescount[i]+=1
            line=line[1:]
            for i in range(len(opcodes)):
                if any(opcodes[i]==li for li in line):
                    features.append(opcodes[i])
                    opcodescount[i]+=1
            for i in range(len(registers)):
                for li in line:
                    if registers[i] in li and ('text' in l or 'CODE' in l):
                        registerscount[i]+=1
            for i in range(len(keywords)):
                for li in line:
                    if keywords[i] in li:
                        keywordcount[i]+=1
    for prefix in prefixescount:
        file1.write(str(prefix)+",")
    for opcode in opcodescount:
        file1.write(str(opcode)+",")
    for register in registerscount:
        file1.write(str(register)+",")
    for key in keywordcount:
        file1.write(str(key)+",")
    file1.write("\n")
file1.close()
```

```
def fourthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.re
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'x
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
```

```
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fifthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.re
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'x
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
    files = os.listdir('fifth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fifth/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
```

```
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present System
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
    p3=Process(target=thirdprocess)
    p4=Process(target=fourthprocess)
    p5=Process(target=fifthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

if __name__=="__main__":
    main()
```

In [17]:
```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the extracted features from .as
# this file will be uploaded in the drive, you can directly use this
dfasm=pd.read_csv("asmoutputfile.csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='left')
result_asm.head()
```

Out[17]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip	Cl
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83	0	17	48	29	
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12	0	14	0	20	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14	0	11	0	9	
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	0	8	0	6	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	0	11	0	11	

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

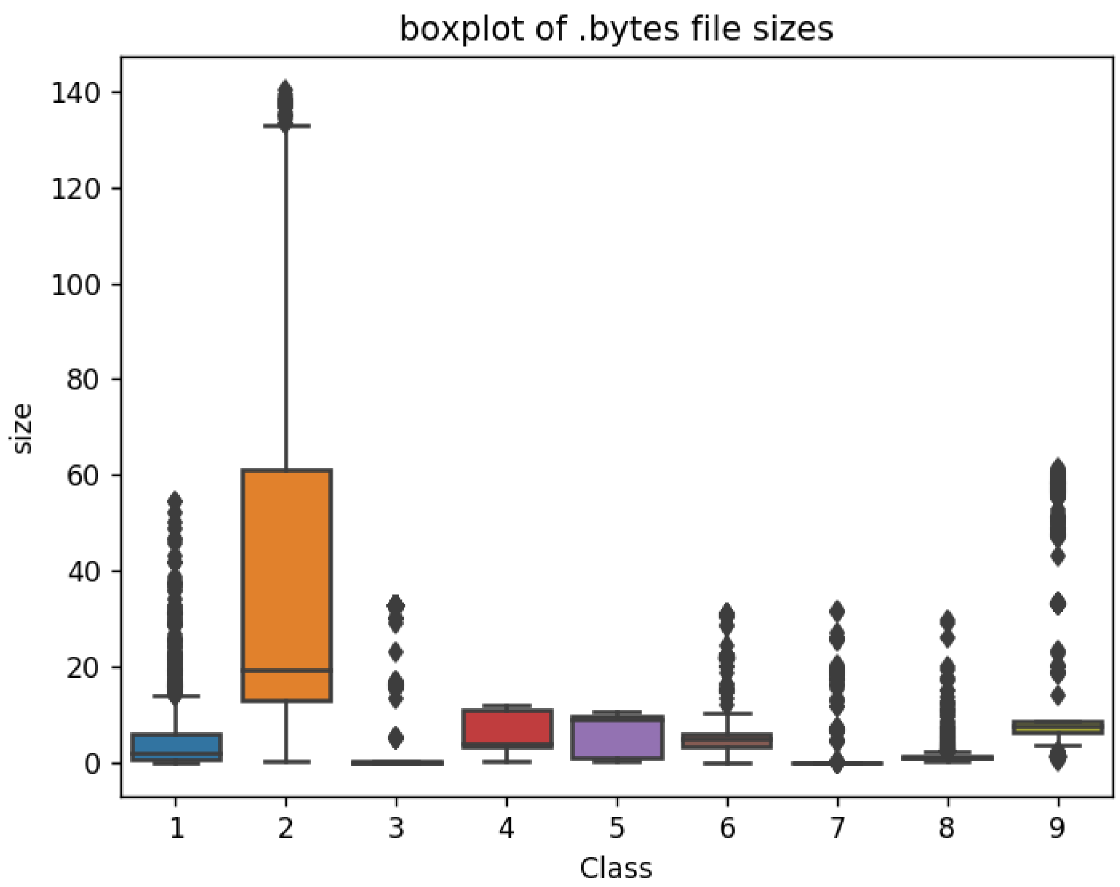
In [18]:
```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in tqdm(files):
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

```
100%|██████████| 10868/10868 [00:04<00:00, 2399.65it/s]

   Class                    ID      size
0      1  jsTnFQZN0zuGqAgcOfaS  0.991247
1      5  bGPHZFpAL3N957064wzj  0.539613
2      1  9iQ3G1aDjec46ULCHI8h  0.350420
3      3  cqHlrY9oAVpyWMKJ8mOF  0.122837
4      1  CM53GutBya9do7piSRe0  1.348861
```

4.2.1.2 Distribution of .asm file sizes

In [139]:
```
#boxplot of asm files
ax = sns.boxplot(x="Class", y="size", data=asm_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

```
<IPython.core.display.Javascript object>
```



In [19]:
```
# add the file size feature to previous extracted features
print(result_asm.shape)
print(asm_size_byte.shape)
result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='ID', how='left')
result_asm.head()
```

```
(10868, 53)
(10868, 3)
```

Out[19]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	48	29	1	(
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	0	20	1	(
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	0	9	1	(
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	0	6	1	(
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	0	11	1	(

5 rows × 54 columns

In [20]:
```
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

Out[20]:

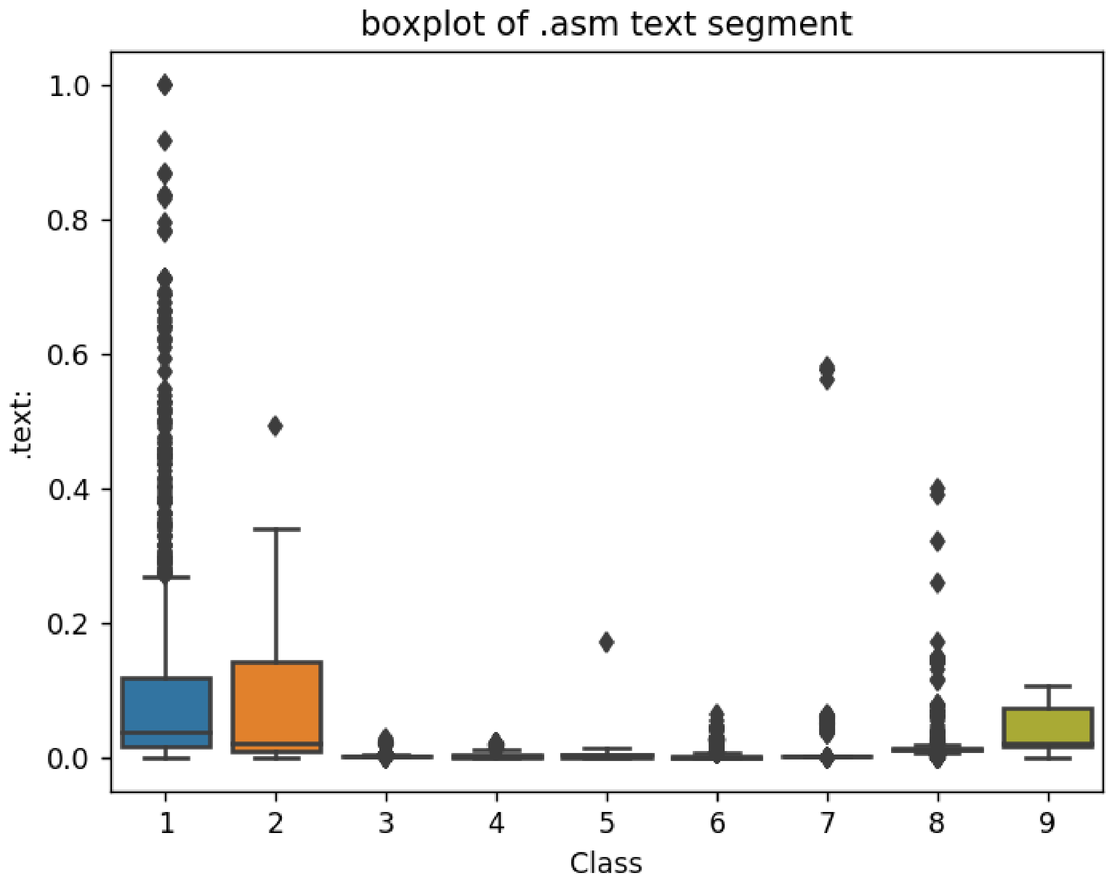
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.00
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.00
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.00
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.00
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.00

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [146]:
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```

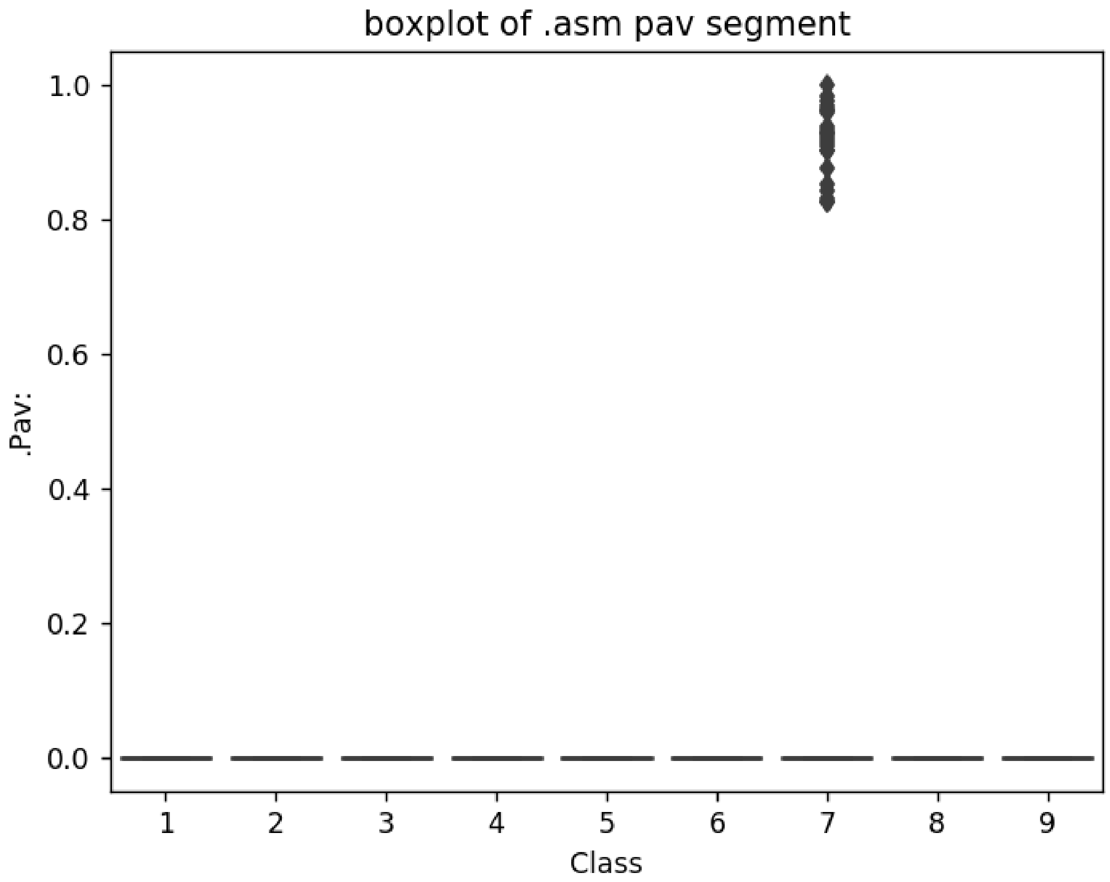
```
<IPython.core.display.Javascript object>
```




```
The plot is between Text and class
Class 1,2 and 9 can be easly separated
```

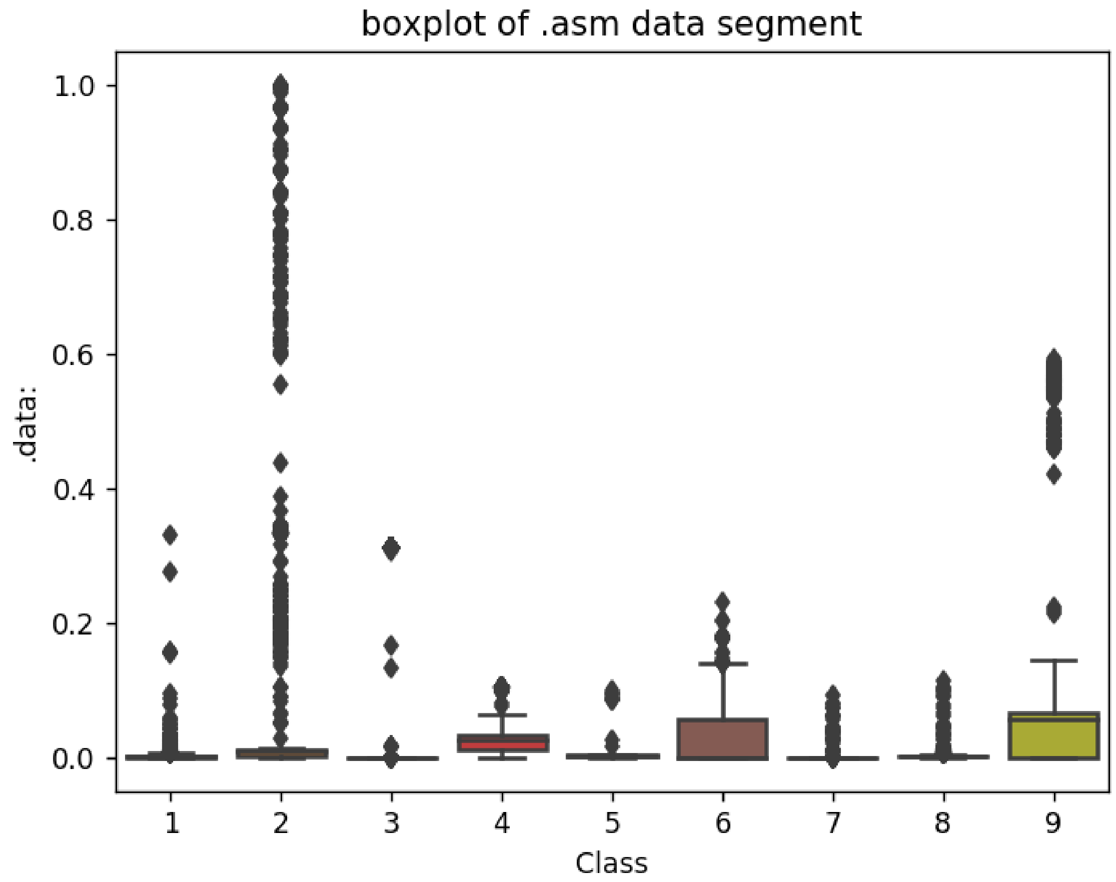
In [115]:
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```

<IPython.core.display.Javascript object>



In [116]:
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

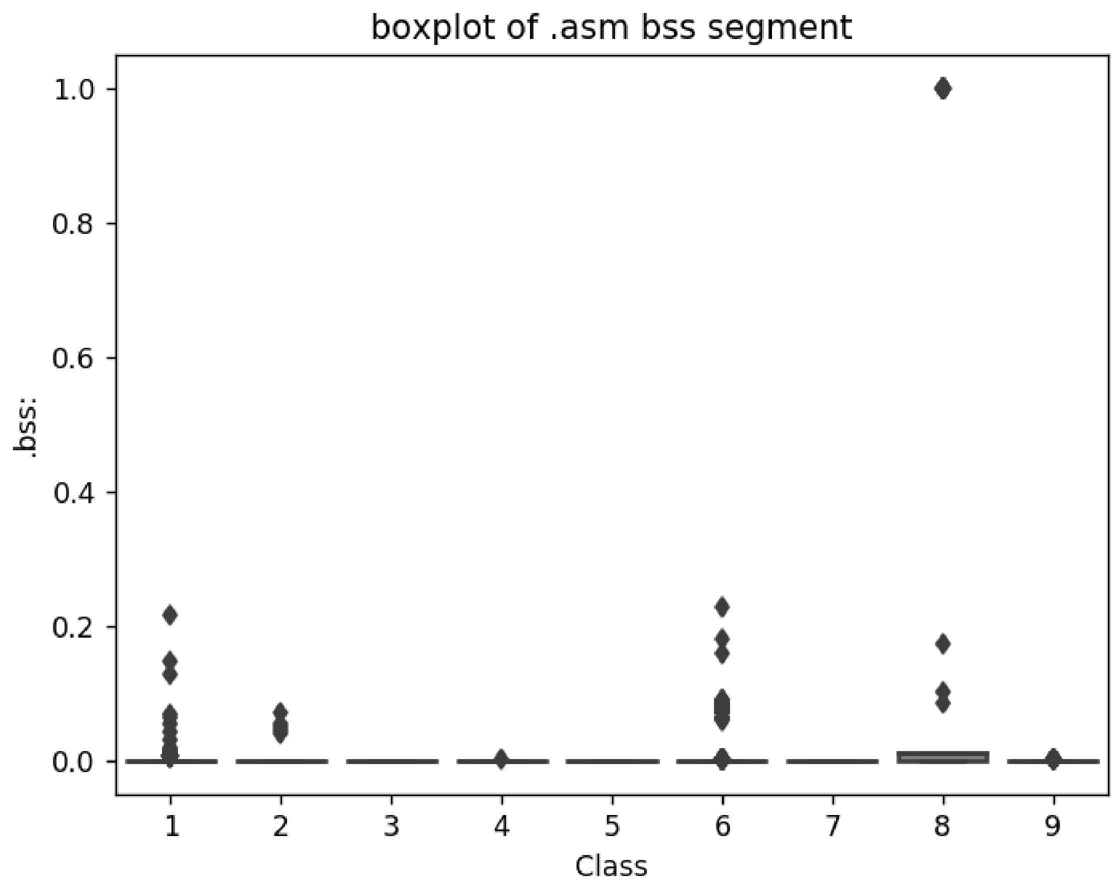



```
The plot is between data segment and class label
```

class 6 and class 9 can be easily separated from given points

In [117]:
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```

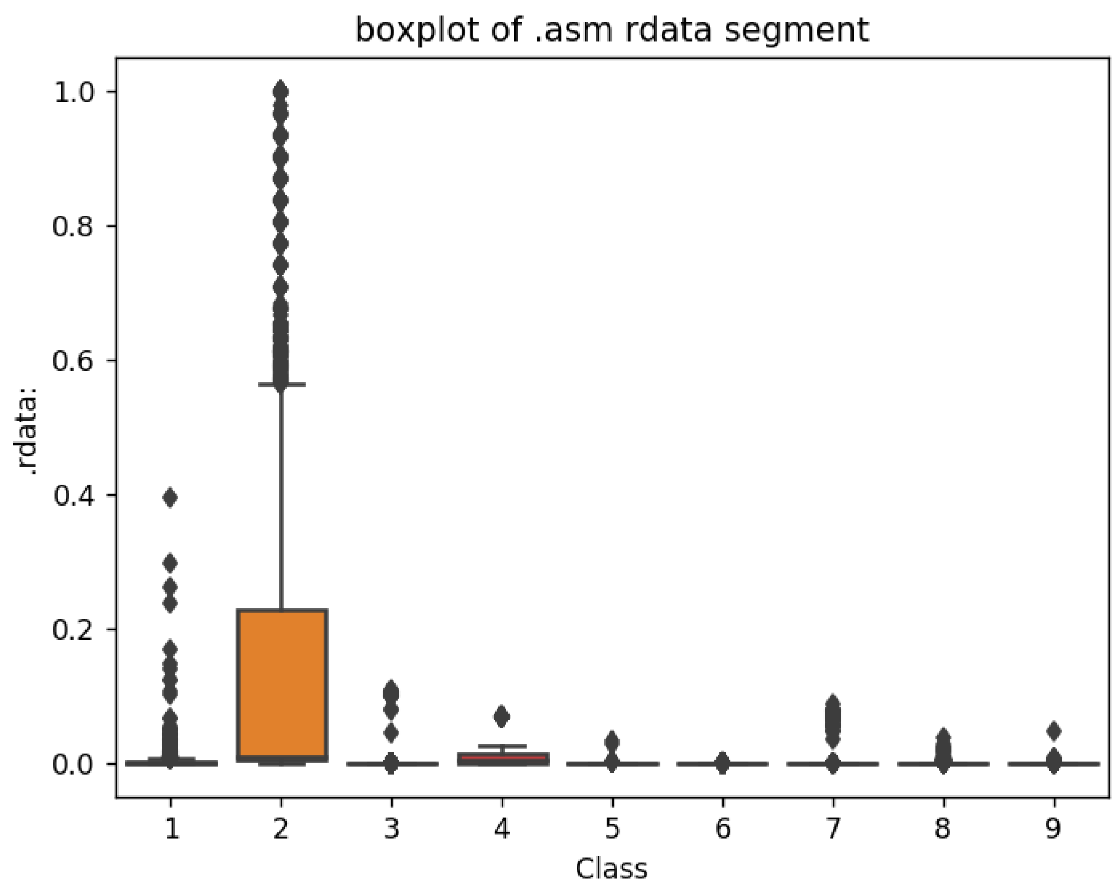
<IPython.core.display.Javascript object>



plot between bss segment and class label
very less number of files are having bss segment

In [118]:
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```

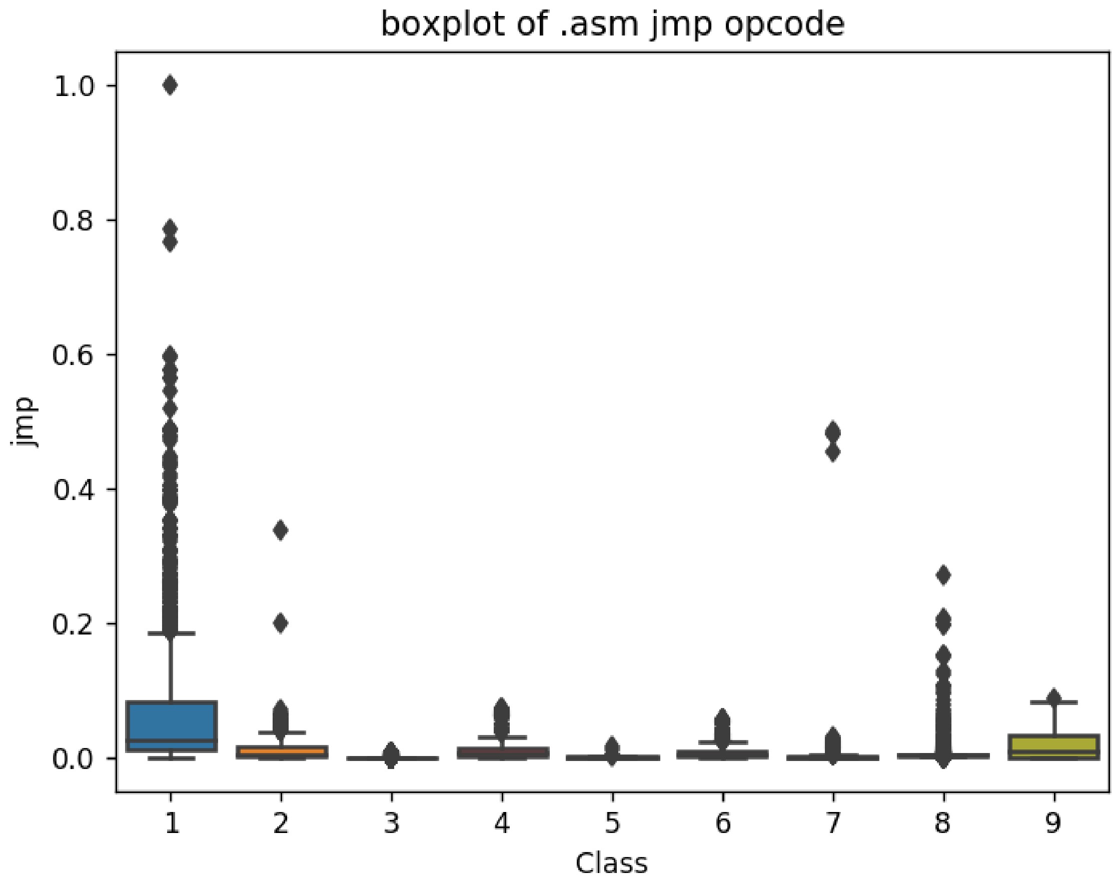
<IPython.core.display.Javascript object>




```
Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines
```

In [119]:
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```

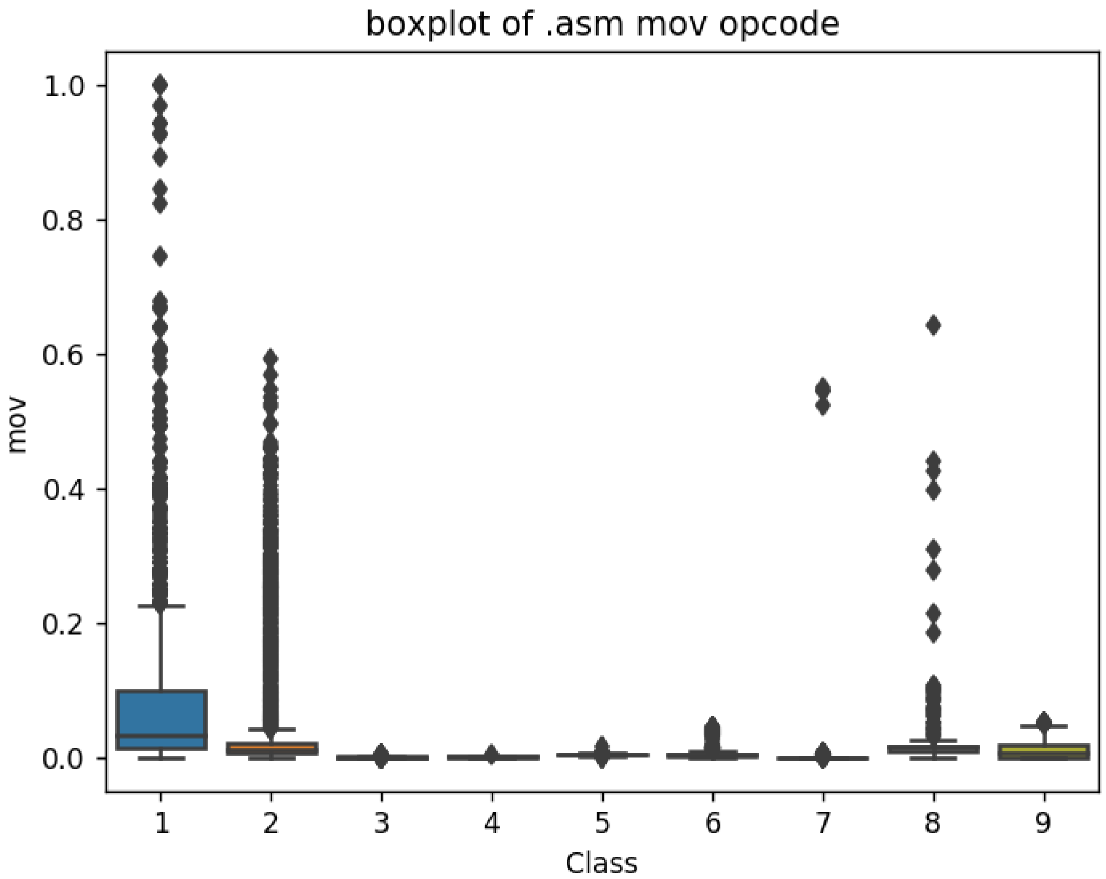
```
<IPython.core.display.Javascript object>
```




```
plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files
```

In [120]:
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```

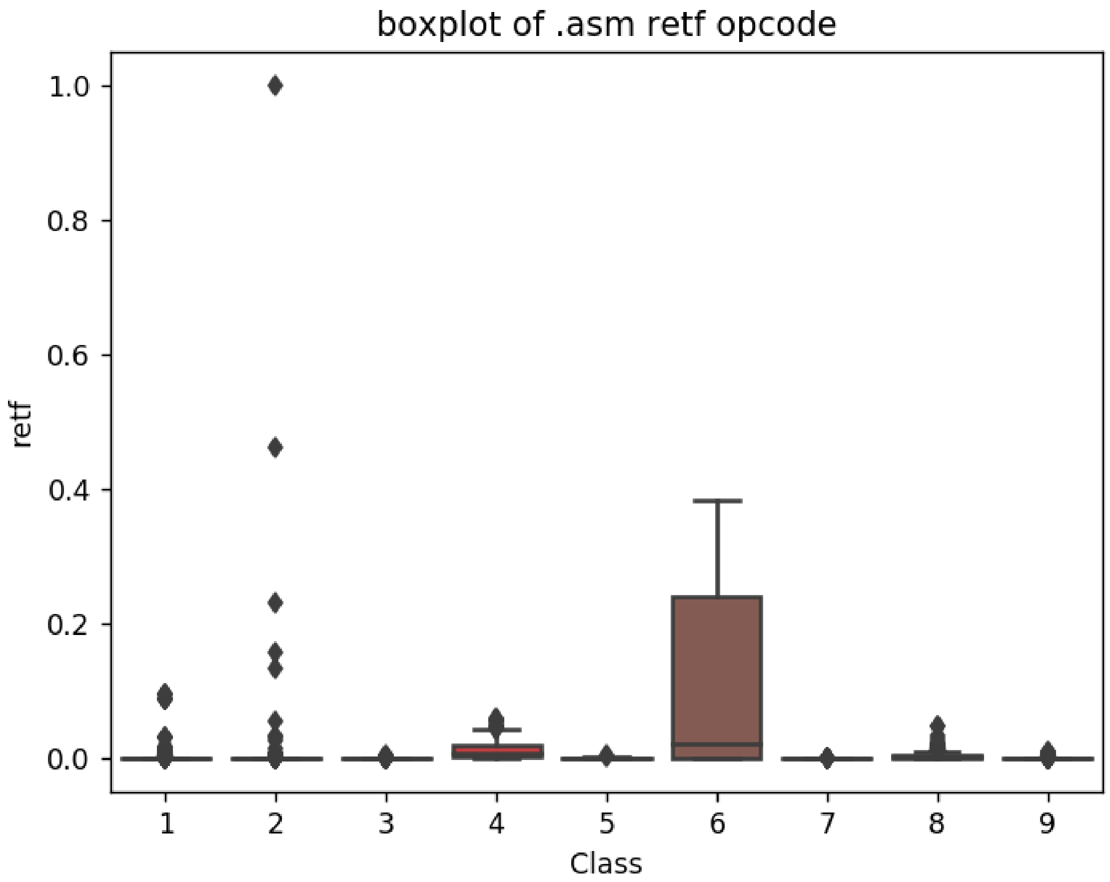
<IPython.core.display.Javascript object>



plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [121]:
```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```

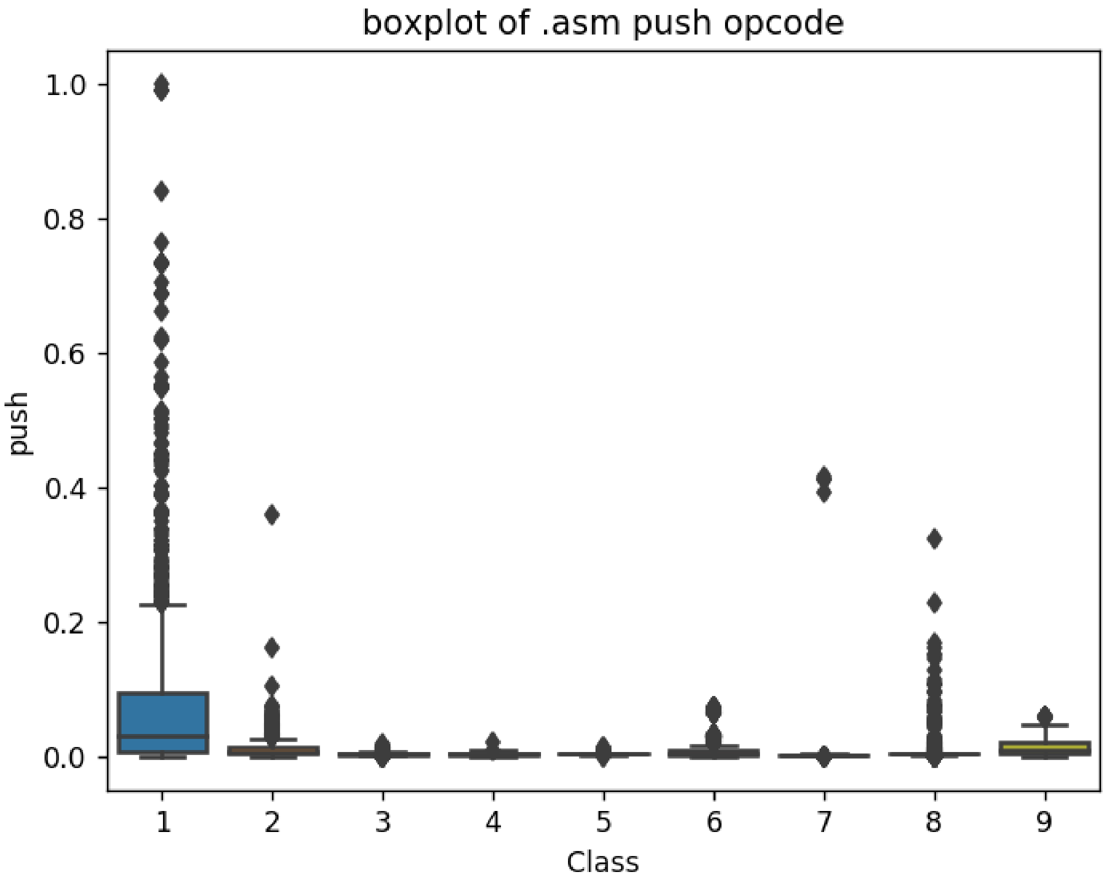
<IPython.core.display.Javascript object>




```
plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.
```

In [122]:
```
ax = sns.boxplot(x="Class", y="push", data=result_asm)
plt.title("boxplot of .asm push opcode")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

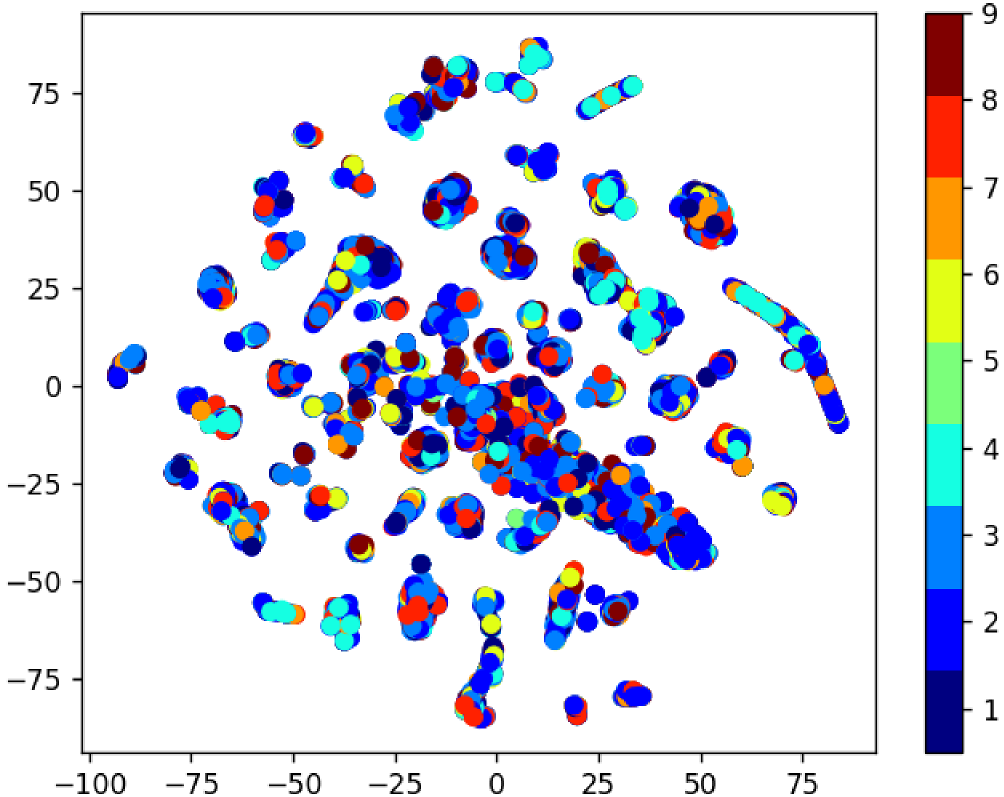



```
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000
```

4.2.2 Multivariate Analysis on .asm file features

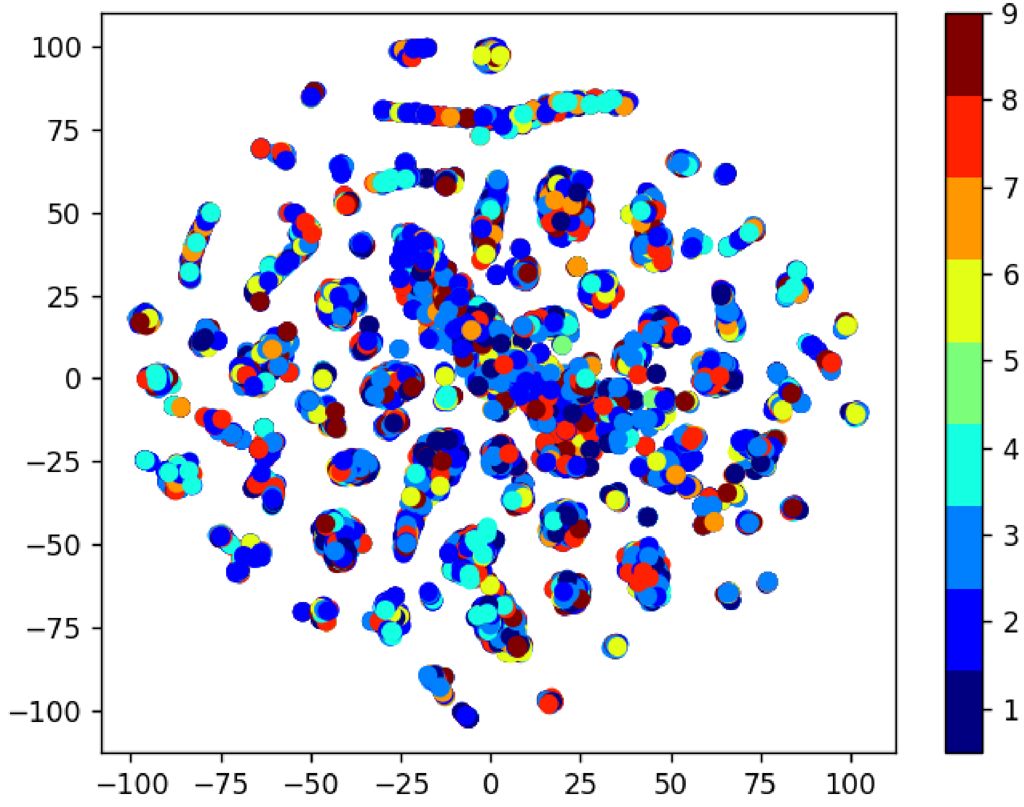
In [129]:
```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic-neighbourhoo

#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_asm.drop(['ID','Class'], axis=1).fillna(0))
vis_x = results[:, 0]
vis_y = results[:, 1   ]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

```
<IPython.core.display.Javascript object>
```


75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75
9
8
7
6
5
4
3
2
1

In [147]:
```
# by univariate analysis on the .asm file features we are getting very negligible information from
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing those features
# the plot looks very messy

xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.CODE','size'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

```
<IPython.core.display.Javascript object>
```




```
TSNE for asm data with perplexity 50
```

4.2.3 Conclusion on EDA

- We have taken only 52 features from asm files (after reading through many blogs and research papers)
- The univariate analysis was done only on few important features.
- Take-aways
  - 1. Class 3 can be easily separated because of the frequency of segments,opcodes and keywords being less

- 2. Each feature has its unique importance in separating the Class labels.

4.3 Train and test split

In [48]:

```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

In [150]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y_train_asm,t
```

In [153]:
```
print( X_cv_asm.isnull().all())
```

```
HEADER:    False
.text:     False
.Pav:      False
.idata:    False
.data:     False
.bss:      False
.rdata:    False
.edata:    False
.rsrc:     False
.tls:      False
.reloc:    False
jmp        False
mov        False
retf       False
push       False
pop        False
xor        False
retn       False
nop        False
sub        False
inc        False
dec        False
add        False
imul       False
xchg       False
or         False
shr        False
cmp        False
call       False
shl        False
ror        False
rol        False
jnb        False
jz         False
lea        False
movzx      False
.dll       False
std::      False
:dword     False
```

```
edx         False
esi         False
eax         False
ebx         False
ecx         False
edi         False
ebp         False
esp         False
eip         False
size        False
dtype: bool
```

4.4. Machine Learning models on features of .asm files

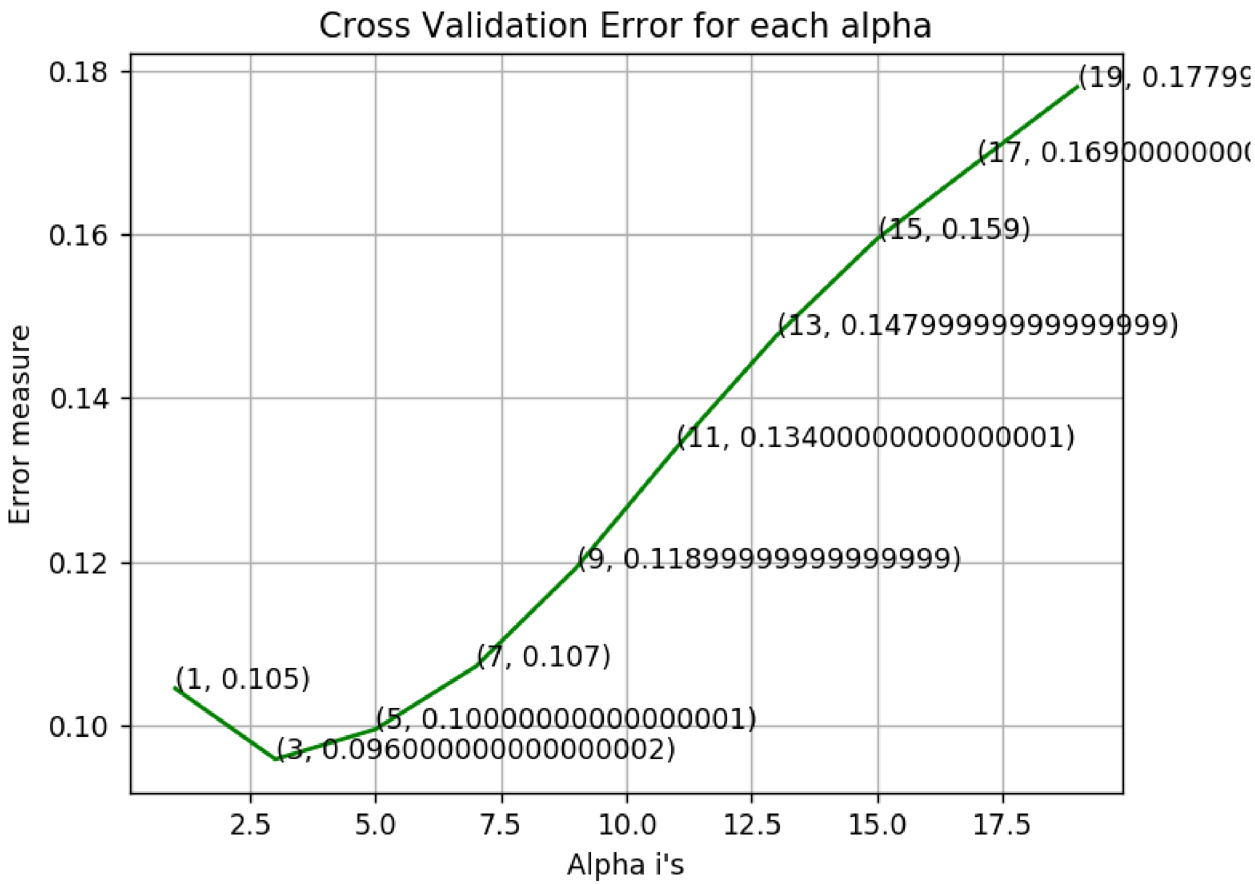
4.4.1 K-Nearest Neigbors

In [159]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbor
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geome
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calib
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)  Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])
```

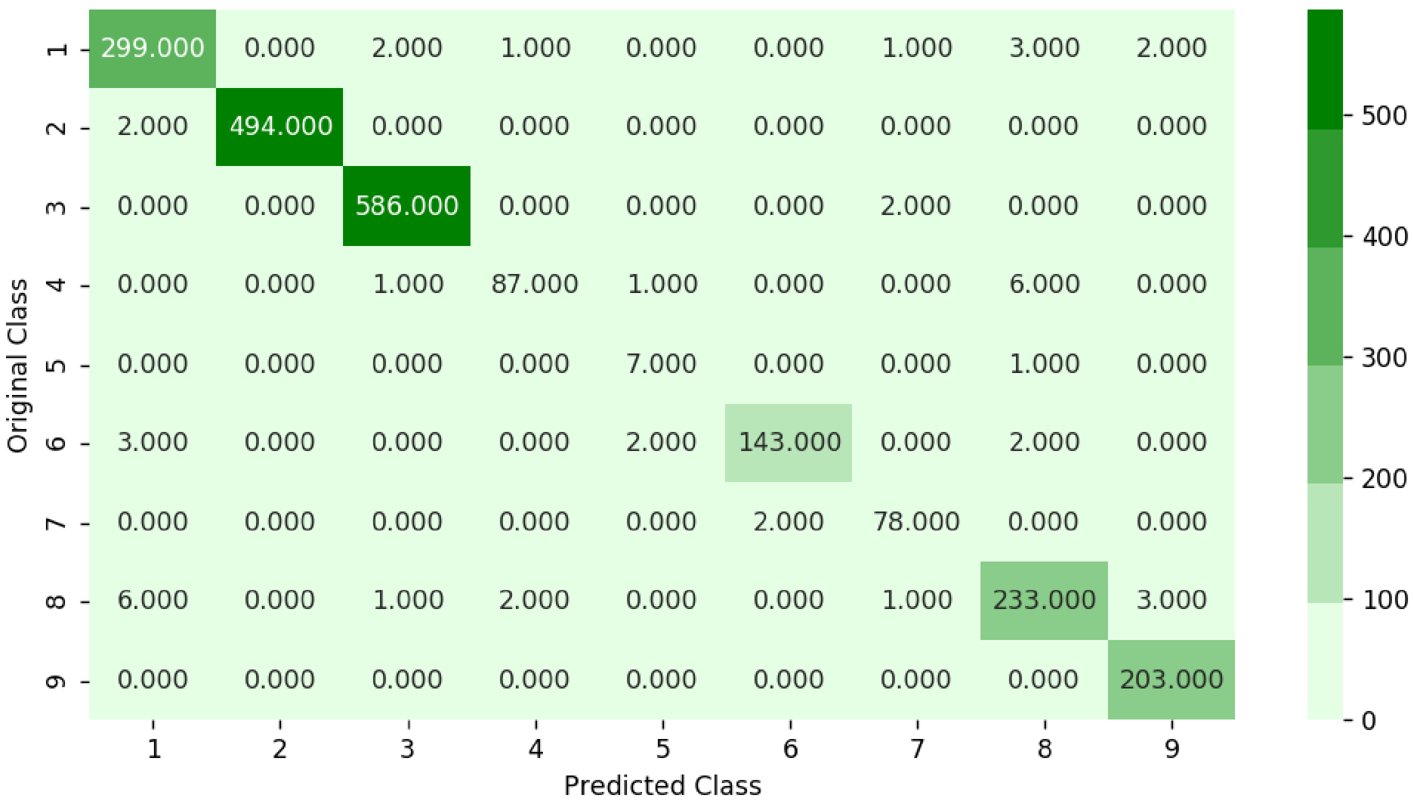
```
best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',log_loss(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',log_loss(y_cv_asm, predict_y))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',log_loss(y_test_asm, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for k =  1 is 0.104531321344
log_loss for k =  3 is 0.0958800580948
log_loss for k =  5 is 0.0995466557335
log_loss for k =  7 is 0.107227274345
log_loss for k =  9 is 0.119239543547
log_loss for k =  11 is 0.133926642781
log_loss for k =  13 is 0.147643793967
log_loss for k =  15 is 0.159439699615
log_loss for k =  17 is 0.16878376444
log_loss for k =  19 is 0.178020728839
```

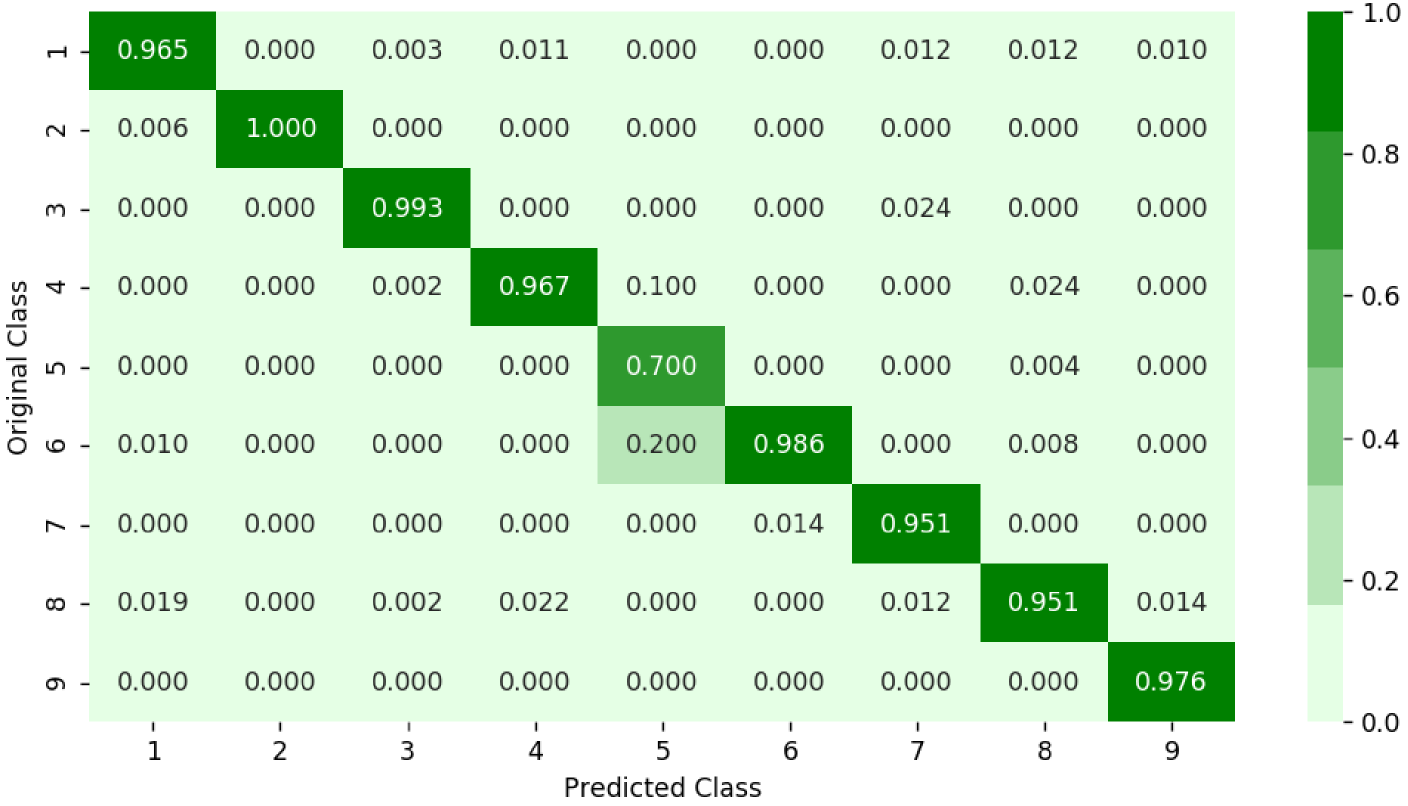
```
<IPython.core.display.Javascript object>
```


Cross Validation Error for each alpha
Error measure
Alpha i's
0.18
0.16
0.14
0.12
0.10
2.5
5.0
7.5
10.0
12.5
15.0
17.5
(1, 0.105)
(3, 0.09600000000000002)
(5, 0.10000000000000001)
(7, 0.107)
(9, 0.11899999999999999)
(11, 0.13400000000000001)
(13, 0.14799999999999999)
(15, 0.159)
(19, 0.1779

```
log loss for train data 0.0476773462198
log loss for cv data 0.0958800580948
log loss for test data 0.0894810720832
Number of misclassified points  2.02391904324
-------------------------------------------------- Confusion matrix --------------------------------------------
---------

<IPython.core.display.Javascript object>
```



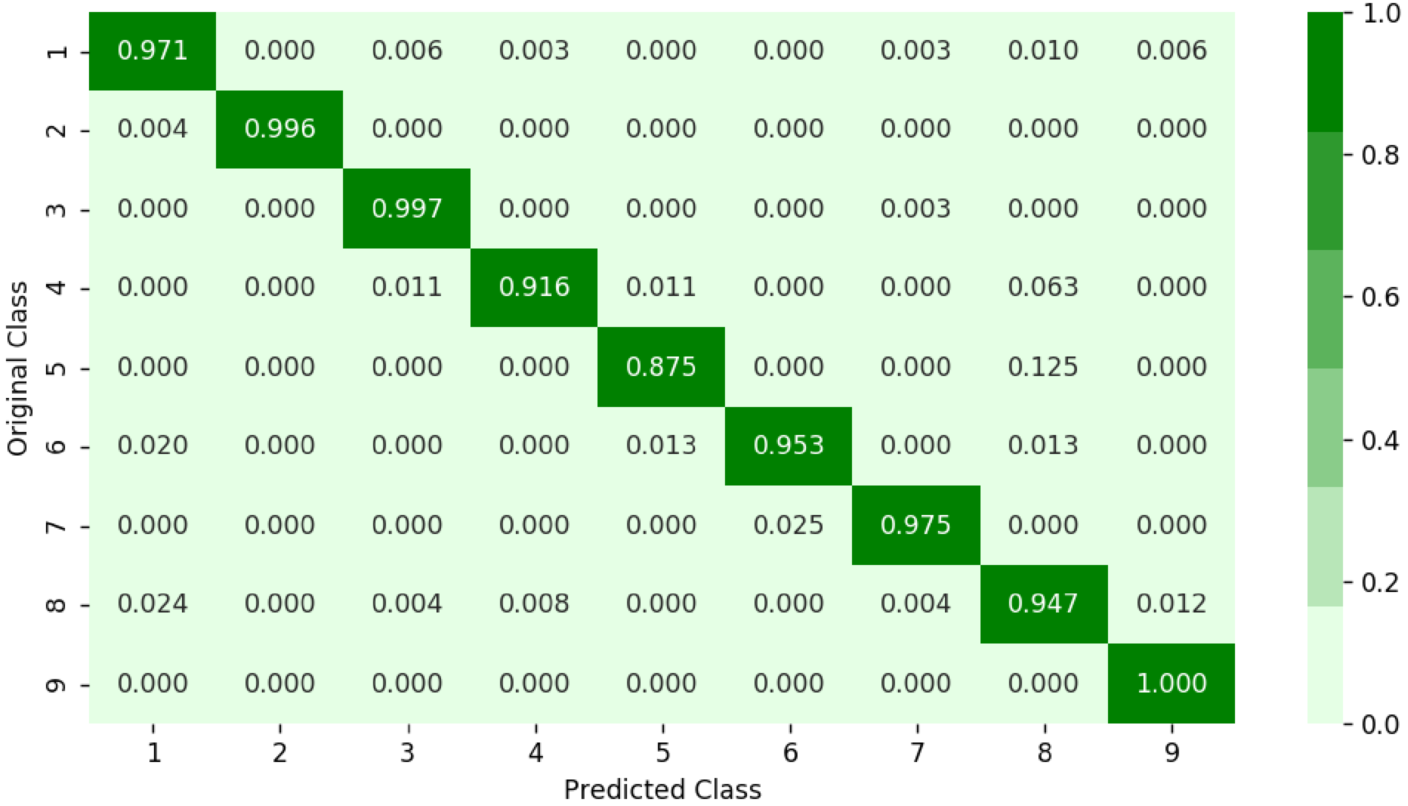

```
------------------------------------------------- Precision matrix -----------------------------------------
---------

<IPython.core.display.Javascript object>
```




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

```
-------------------------------------------------- Recall matrix --------------------------------------------
------
```

```
<IPython.core.display.Javascript object>
```



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

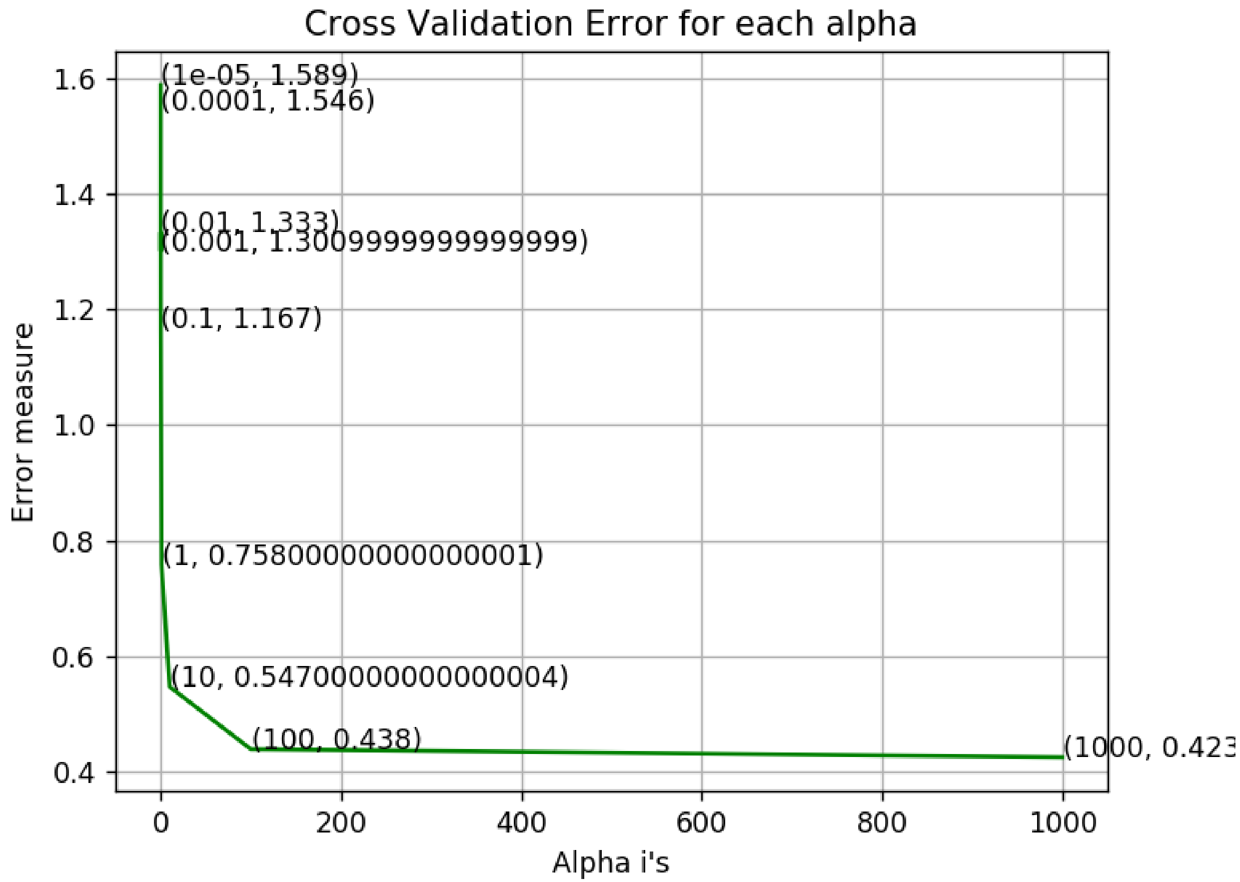
4.4.2 Logistic Regression

In [160]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDC
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------



alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

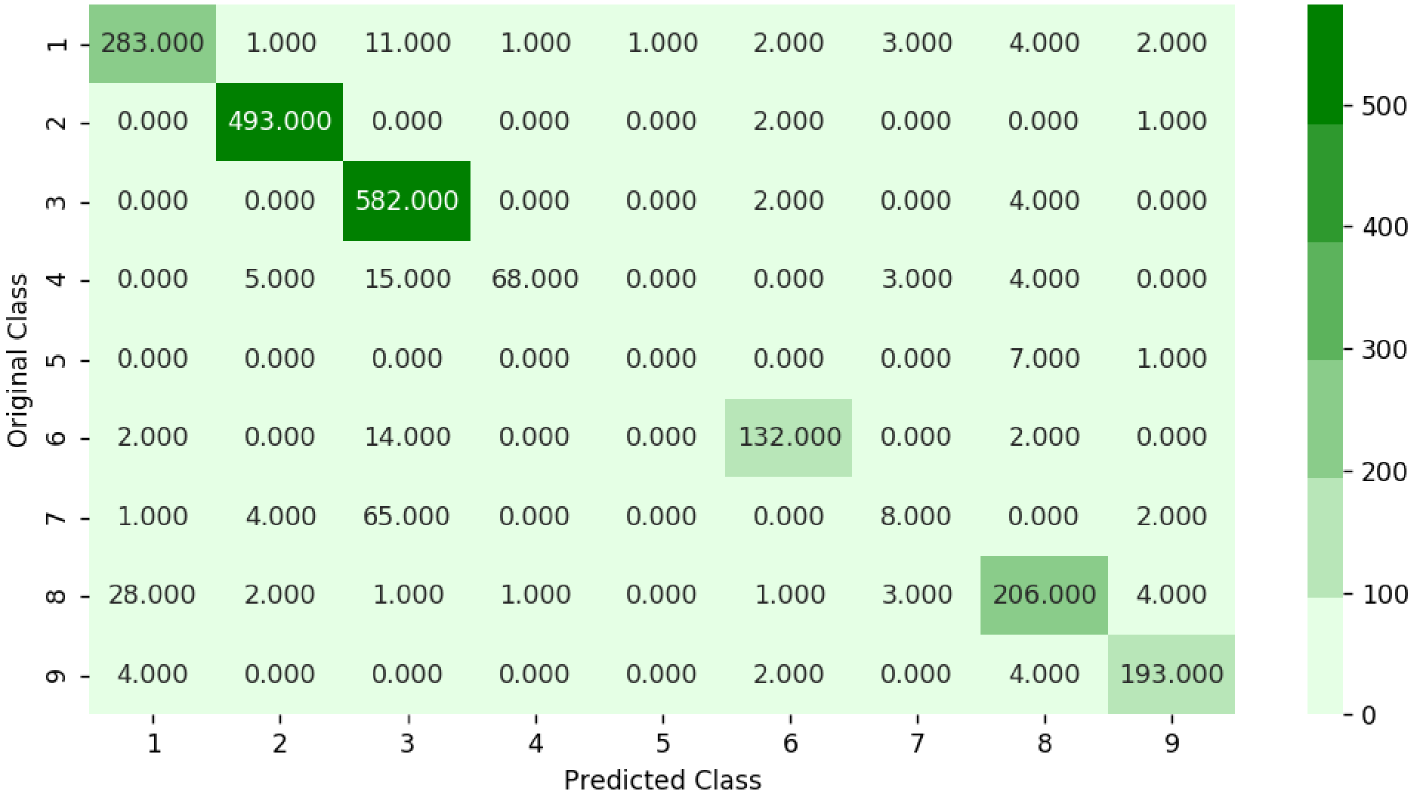
```
log_loss for c =  1e-05 is 1.58867274165
log_loss for c =  0.0001 is 1.54560797884
log_loss for c =  0.001 is 1.30137786807
log_loss for c =  0.01 is 1.33317456931
log_loss for c =  0.1 is 1.16705751378
log_loss for c =  1 is 0.757667807779
log_loss for c =  10 is 0.546533939819
log_loss for c =  100 is 0.438414998062
log_loss for c =  1000 is 0.424423536526
```

```
<IPython.core.display.Javascript object>
```

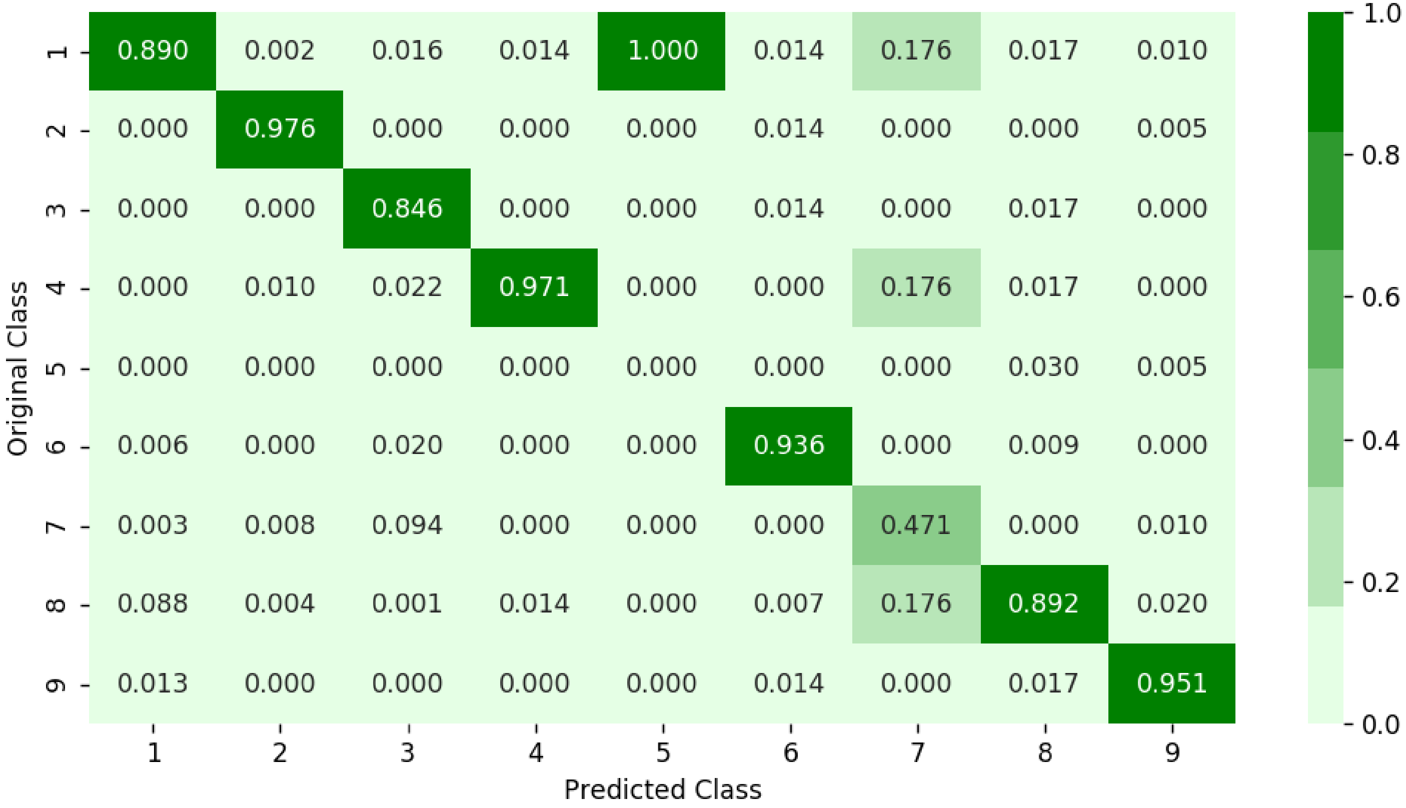



```
log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points  9.61361545538
-------------------------------------------------- Confusion matrix -----------------------------------------
---------
```

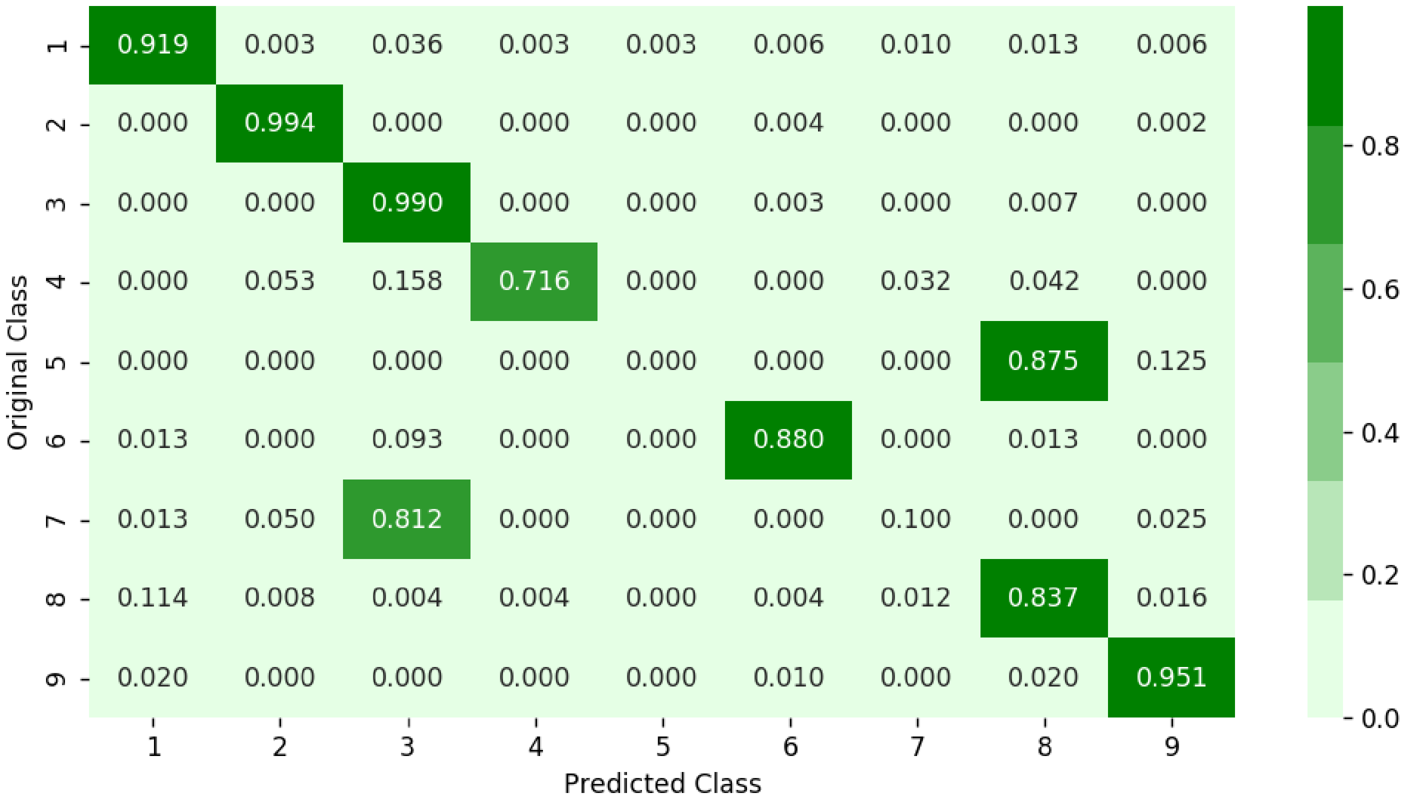
```
<IPython.core.display.Javascript object>
```



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

<IPython.core.display.Javascript object>
```




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

<IPython.core.display.Javascript object>
```


Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
1 0.919 0.003 0.036 0.003 0.003 0.006 0.010 0.013 0.006
2 0.000 0.994 0.000 0.000 0.000 0.004 0.000 0.000 0.002
3 0.000 0.000 0.990 0.000 0.000 0.003 0.000 0.007 0.000
4 0.000 0.053 0.158 0.716 0.000 0.000 0.032 0.042 0.000
5 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.875 0.125
6 0.013 0.000 0.093 0.000 0.000 0.880 0.000 0.013 0.000
7 0.013 0.050 0.812 0.000 0.000 0.000 0.100 0.000 0.025
8 0.114 0.008 0.004 0.004 0.000 0.004 0.012 0.837 0.016
9 0.020 0.000 0.000 0.000 0.000 0.010 0.000 0.020 0.951
0.8
0.6
0.4
0.2
0.0

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

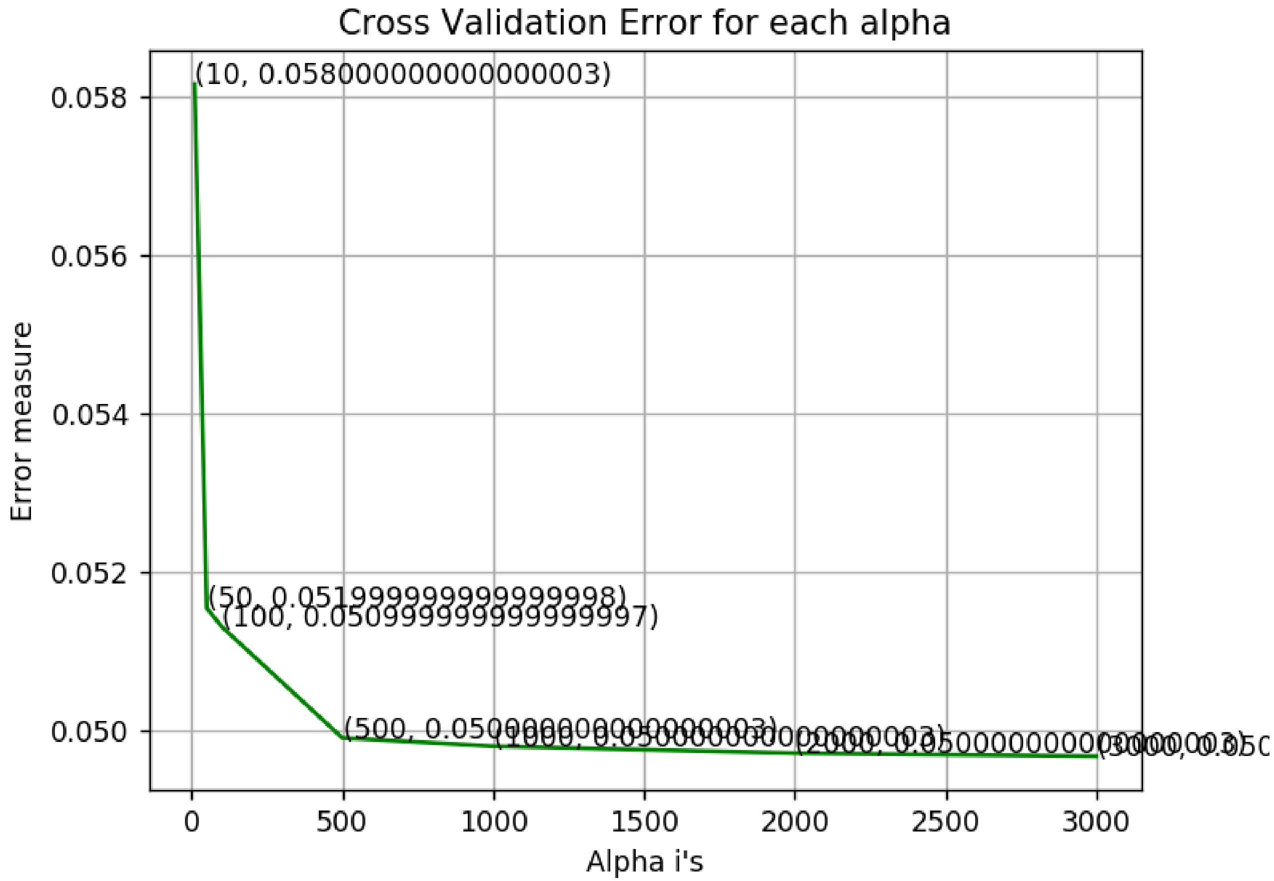
4.4.3 Random Forest Classifier

In [161]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decr
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-c
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
```

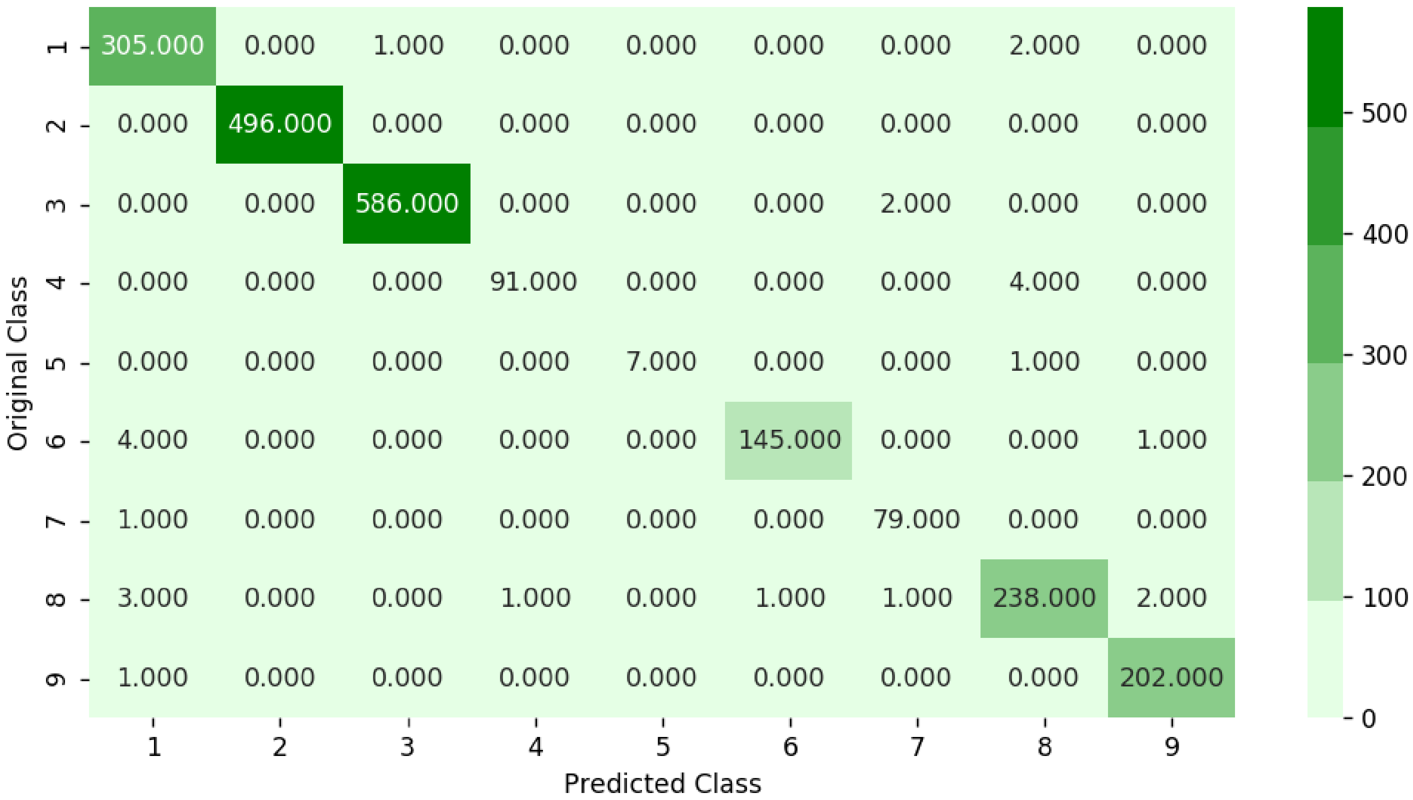
```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.0581657906023
log_loss for c =  50 is 0.0515443148419
log_loss for c =  100 is 0.0513084973231
log_loss for c =  500 is 0.0499021761479
log_loss for c =  1000 is 0.0497972474298
log_loss for c =  2000 is 0.0497091690815
log_loss for c =  3000 is 0.0496706817633
```

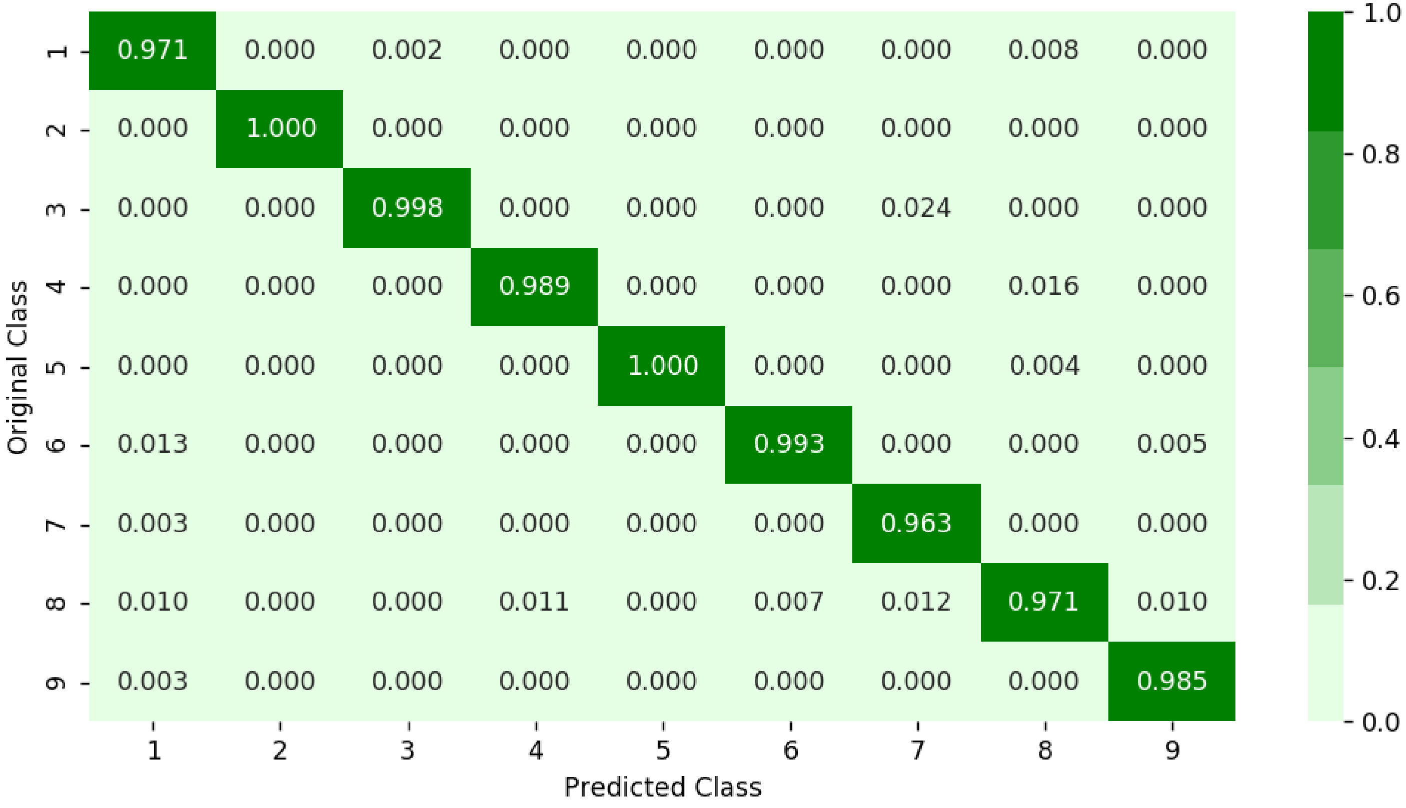
```
<IPython.core.display.Javascript object>
```




```
log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points  1.14995400184
-------------------------------------------------- Confusion matrix --------------------------------------------------
---------

<IPython.core.display.Javascript object>
```

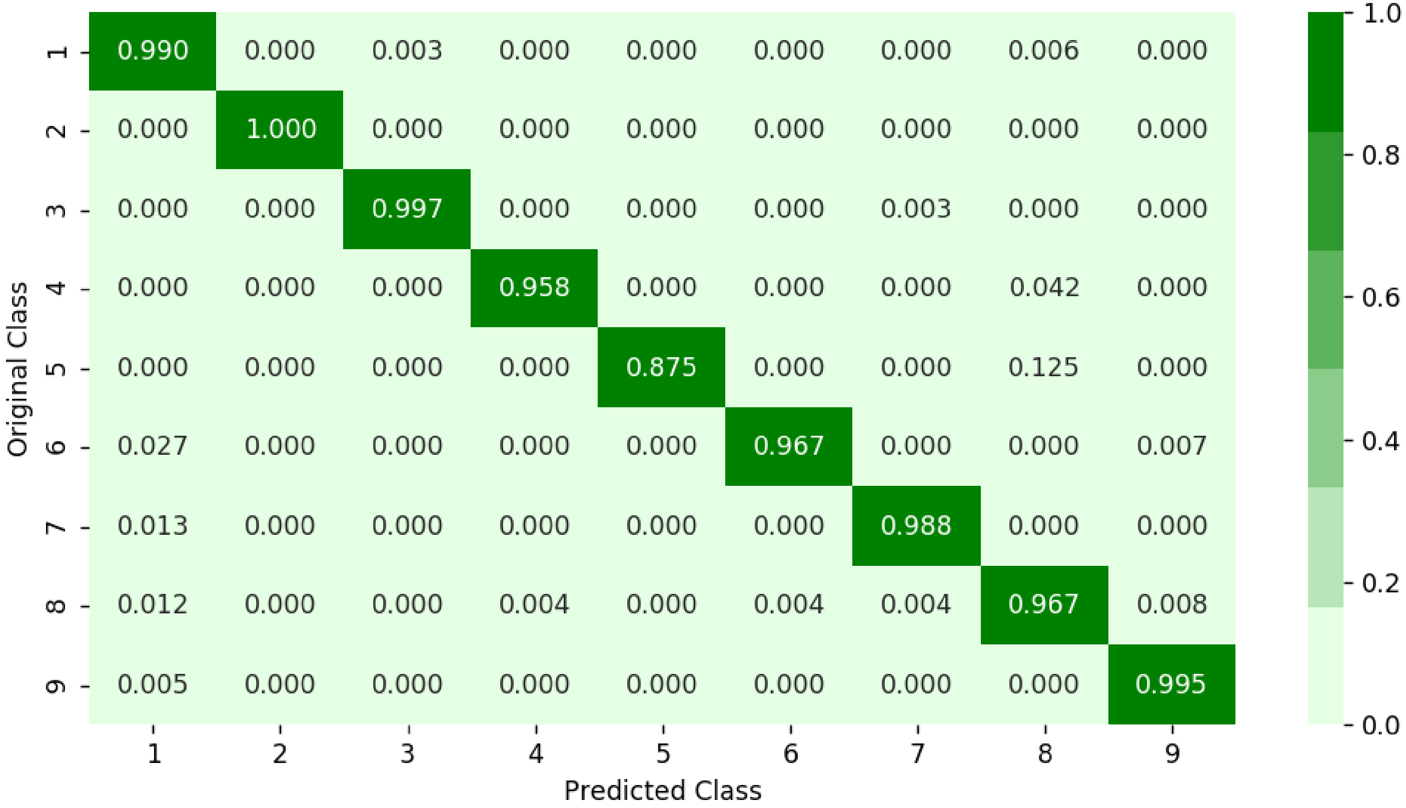



```
-------------------------------------------------- Precision matrix --------------------------------------------------
---------

<IPython.core.display.Javascript object>
```



Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
0.971 0.000 0.002 0.000 0.000 0.000 0.000 0.008 0.000
0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.998 0.000 0.000 0.000 0.024 0.000 0.000
0.000 0.000 0.000 0.989 0.000 0.000 0.000 0.016 0.000
0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.004 0.000
0.013 0.000 0.000 0.000 0.000 0.993 0.000 0.000 0.005
0.003 0.000 0.000 0.000 0.000 0.000 0.963 0.000 0.000
0.010 0.000 0.000 0.011 0.000 0.007 0.012 0.971 0.010
0.003 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.985
1.0 0.8 0.6 0.4 0.2 0.0

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

<IPython.core.display.Javascript object>
```



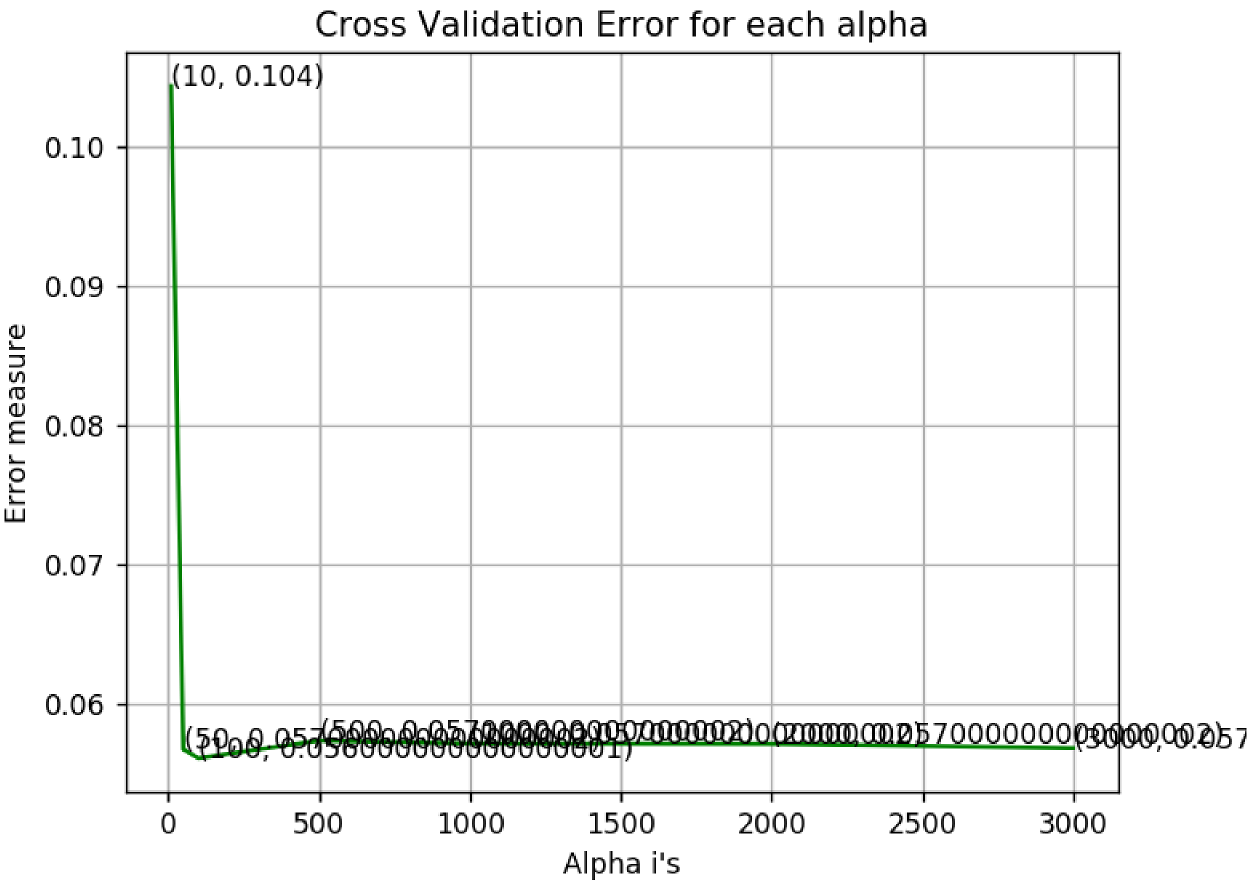

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.4.4 XgBoost Classifier

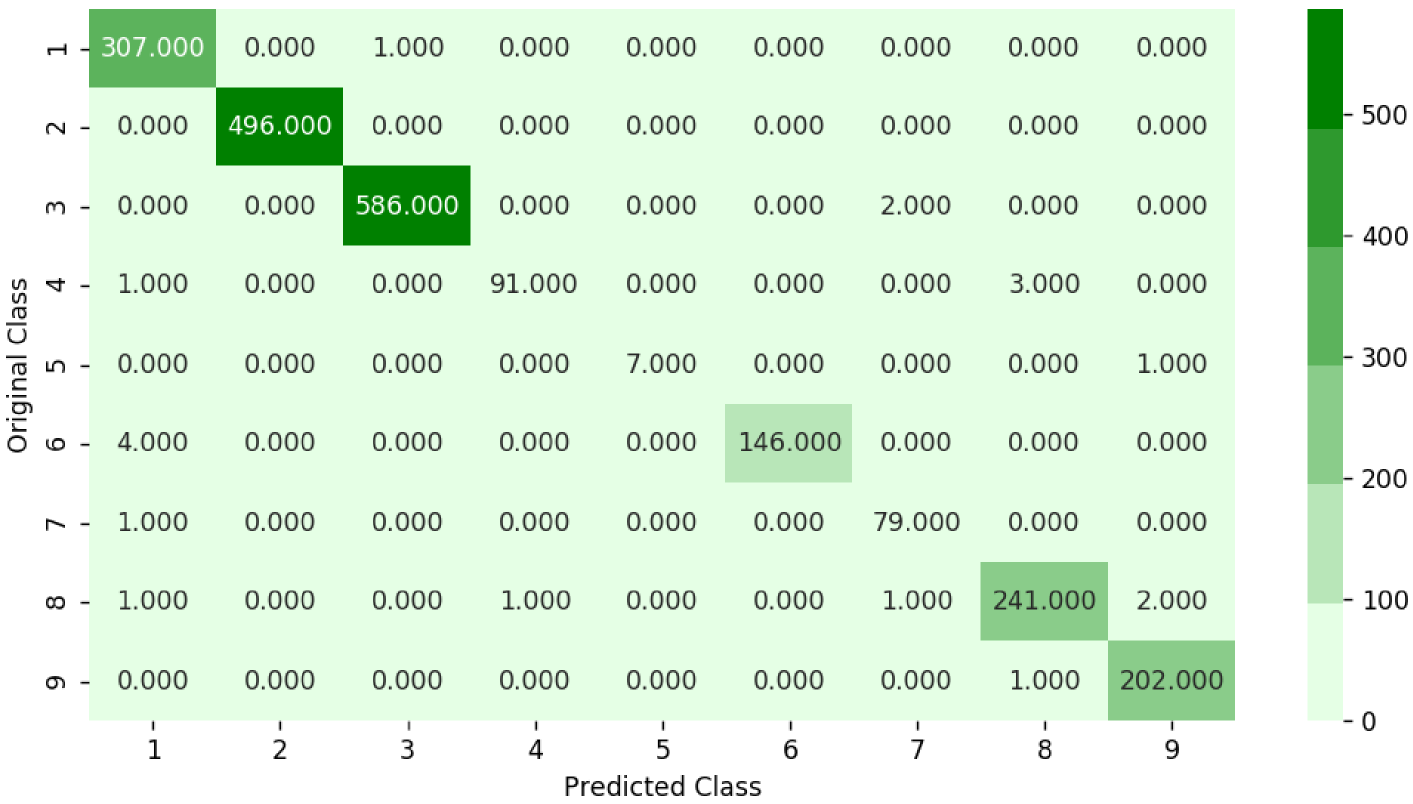
In [162]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#x
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
```

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_asm, predict_
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_asm, 
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_asm, predict_y)
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.104344888454
log_loss for c =  50 is 0.0567190635611
log_loss for c =  100 is 0.056075038646
log_loss for c =  500 is 0.057336051683
log_loss for c =  1000 is 0.0571265109903
log_loss for c =  2000 is 0.057103406781
log_loss for c =  3000 is 0.0567993215778

<IPython.core.display.Javascript object>
```

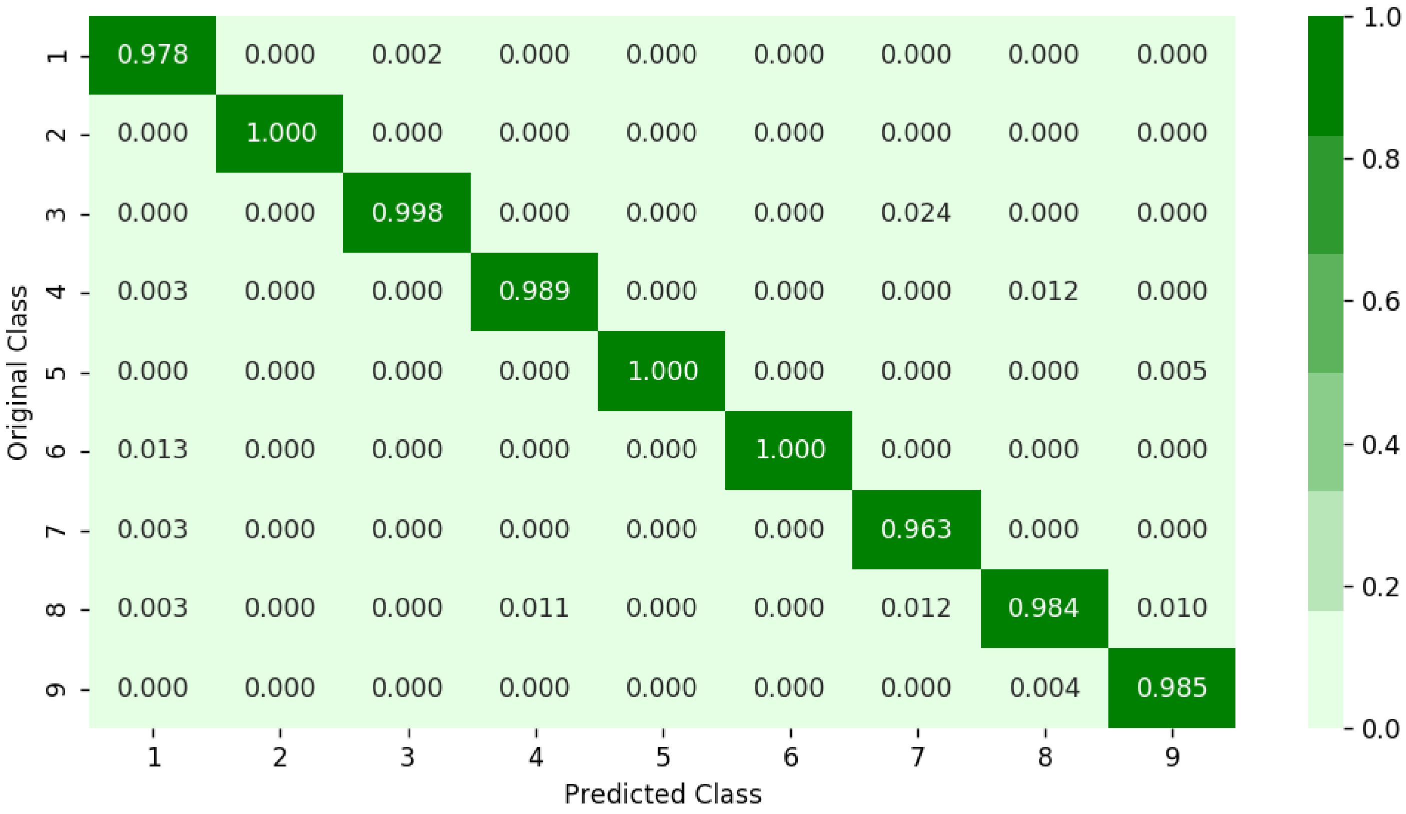



```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.056075038646
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
-------------------------------------------------- Confusion matrix ----------------------------------------------------------
<IPython.core.display.Javascript object>
```




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

<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



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

<IPython.core.display.Javascript object>
```

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

4.4.5 Xgboost Classifier with best hyperparameters

In [163]:

```python
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    '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_asm,y_train_asm)
```

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  1.1min remaining:   39.3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.3min remaining:   23.0s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  1.4min remaining:    9.2s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.3min finished
```

Out[163]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=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, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 20
0, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3,
0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [164]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.15, 'colsample_bytree': 0.5}
```

In [170]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#x
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5,max_depth=3)
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

predict_y = c_cfl.predict_proba(X_train_asm)
print ('train loss',log_loss(y_train_asm, predict_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict_y))
```

```
train loss 0.0102661325822
cv loss 0.0501201796687
test loss 0.0483908764397
```

4.5. Machine Learning models on features of both .asm and .bytes files

4.5.1. Merging both asm and byte file features

In [21]:
```
result.head()
```

Out[21]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9	fa	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.013560	0.013107	0.01
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.001920	0.001147	0.00
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	...	0.009804	0.011777	0.01
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	...	0.002121	0.001886	0.00
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	...	0.001530	0.000853	0.00

5 rows × 261 columns

In [22]:
```
result_asm.head()
```

Out[22]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.00
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.00
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.00
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.00
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.00

5 rows × 54 columns

In [173]:
```
print(result.shape)
print(result_asm.shape)
```

```
(10868, 260)
(10868, 54)
```

In [25]:
```
result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[25]:

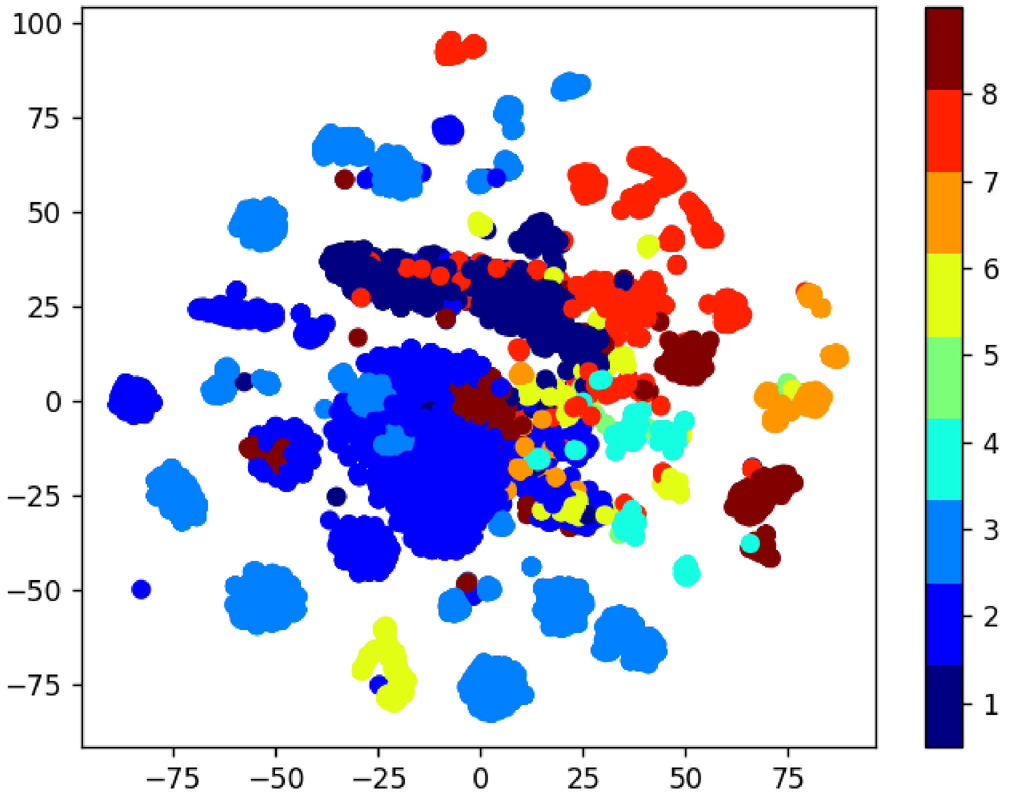
	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	edx	esi	eax	ebx
0	0.000000	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.015418	0.025875	0.025744	0.004910
1	0.000092	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.004961	0.012316	0.007858	0.007570
2	0.000184	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.000095	0.006181	0.000100	0.003773
3	0.000276	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.000343	0.000746	0.000301	0.000360
4	0.000368	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.000343	0.013875	0.000482	0.012932

5 rows × 308 columns

4.5.2. Multivariate Analysis on final fearures

In [181]:
```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x, axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```

<IPython.core.display.Javascript object>



4.5.3. Train and Test split

In [183]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_
```

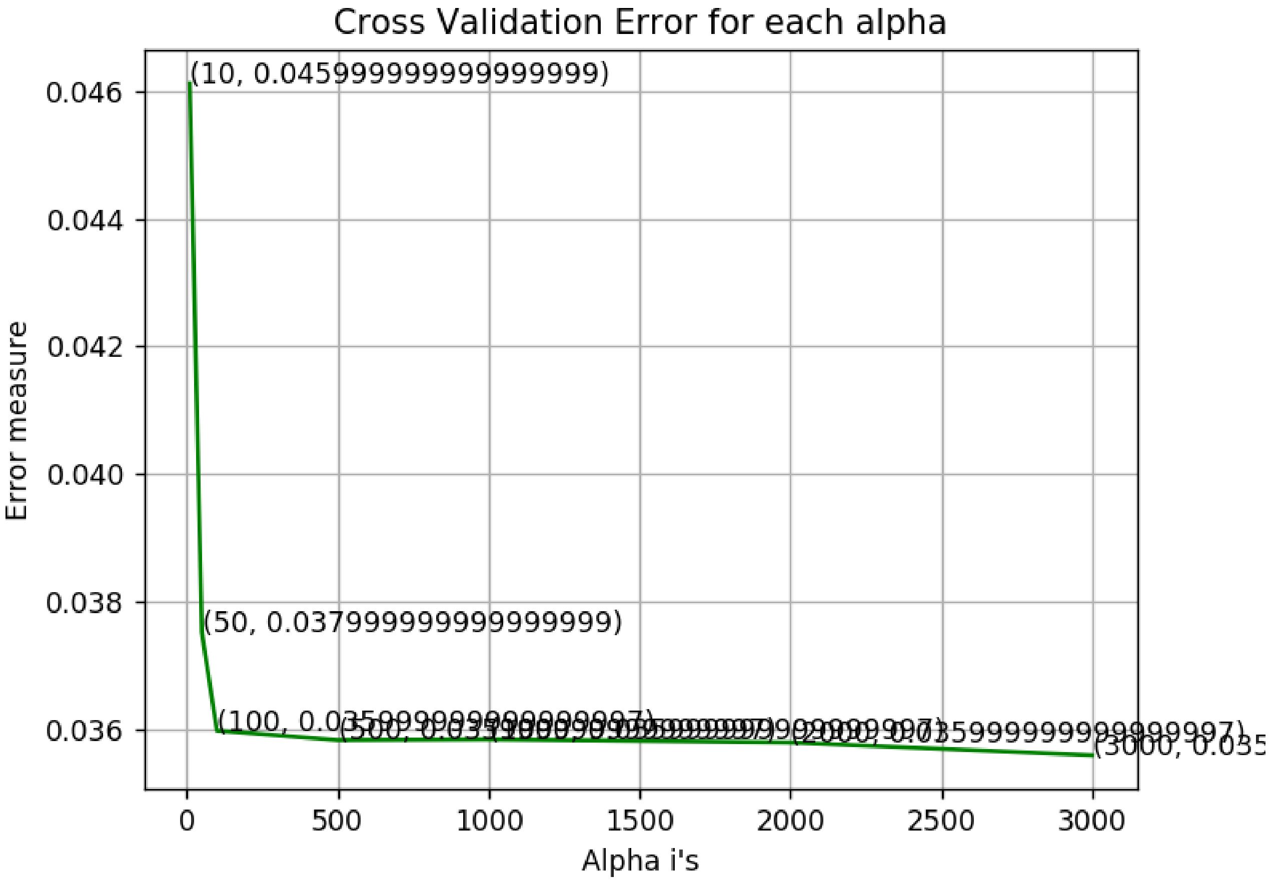
4.5.4. Random Forest Classifier on final features

In [185]:
```
# ----------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decr
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# ----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-c
# ----------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
```

```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predic
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_
```

```
log_loss for c =  10 is 0.0461221662017
log_loss for c =  50 is 0.0375229563452
log_loss for c =  100 is 0.0359765822455
log_loss for c =  500 is 0.0358291883873
log_loss for c =  1000 is 0.0358403093496
log_loss for c =  2000 is 0.0357908022178
log_loss for c =  3000 is 0.0355909487962
```

```
<IPython.core.display.Javascript object>
```




```
For values of best alpha =  3000 The train log loss is: 0.0166267614753
For values of best alpha =  3000 The cross validation log loss is: 0.0355909487962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

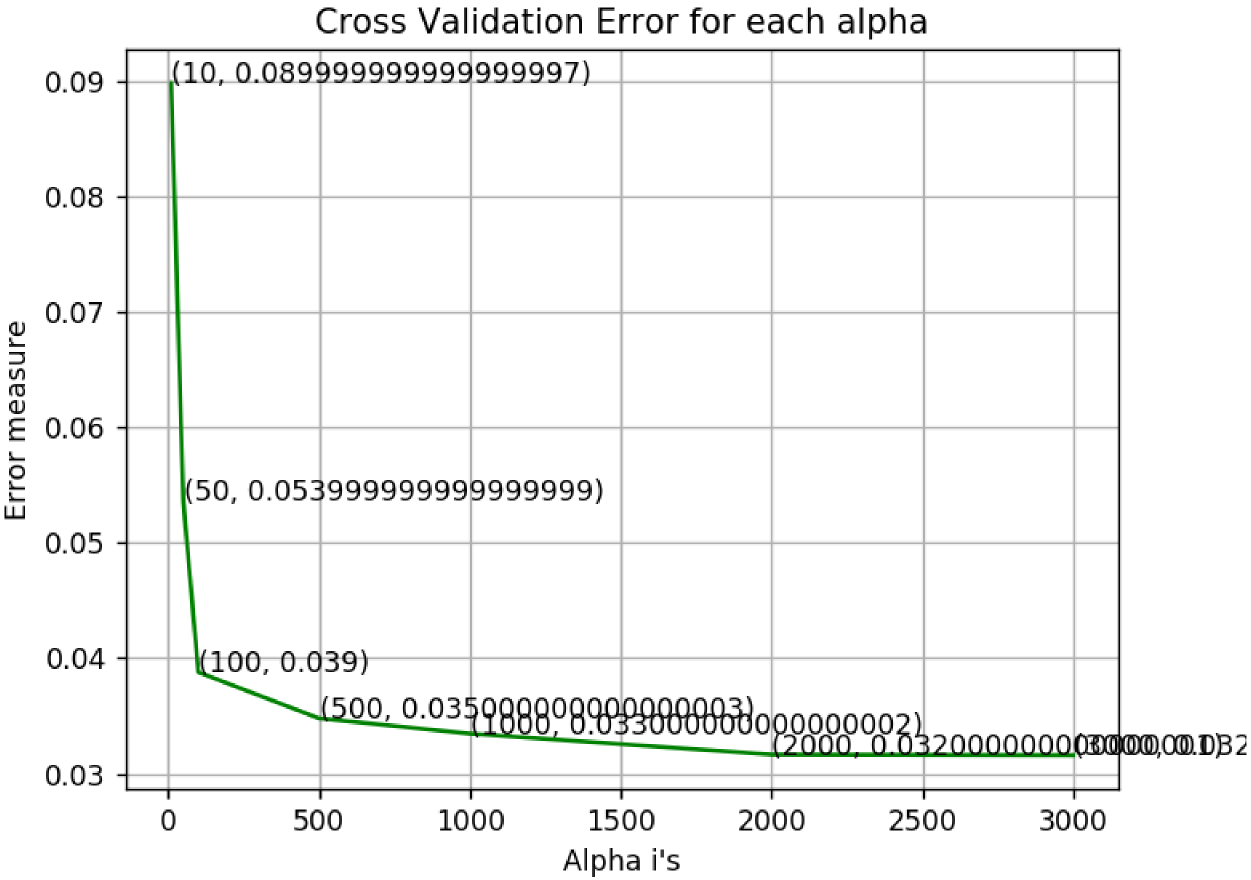
4.5.5. XgBoost Classifier on final features

In [186]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#x
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
```

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=3000,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predi
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_
```

```
log_loss for c =  10 is 0.0898979446265
log_loss for c =  50 is 0.0536946658041
log_loss for c =  100 is 0.0387968186177
log_loss for c =  500 is 0.0347960327293
log_loss for c =  1000 is 0.0334668083237
log_loss for c =  2000 is 0.0316569078846
log_loss for c =  3000 is 0.0315972694477
```

```
<IPython.core.display.Javascript object>
```




```
For values of best alpha =  3000 The train log loss is: 0.0111918809342
For values of best alpha =  3000 The cross validation log loss is: 0.0315972694477
For values of best alpha =  3000 The test log loss is: 0.0323978515915
```

4.5.5. XgBoost Classifier on final features with best hyper parameters using Random search

In [187]:
```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    '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_merge, y_train_merge)
```

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  4.5min remaining:  2.6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remaining:  1.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remaining:   44.5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

Out[187]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=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, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 20
0, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3,
0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [188]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 10, 'learning_rate': 0.15, 'colsample_bytree': 0.3}
```

In [189]:

```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#x
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.3,subsample=1,nthread=
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predi
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
For values of best alpha =  3000 The train log loss is: 0.0121922832297
For values of best alpha =  3000 The cross validation log loss is: 0.0344955487471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

5. Assignments

1. Add bi-grams and n-gram features on byte files and improve the log-loss

2. Using the 'dchad' github account (https://github.com/dchad/malware-detection), decrease the logloss to <=0.01
3. Watch the video (https://www.youtube.com/watch?v=VLQTRlLGz5Y) that was in reference section and implement the image features to improve the logloss

In [ ]:
```
# Task 1 : Adding bi grams, n grams i.e tri or 4 gram features:
# we have our Byte File vocab in form of list
# ID,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,20,21,22,2

# Lets use this to generate bi gram, tri gram and 4 gram features then combine them together using Count vector
# PS We didnt use Count Vectoriser earlier to generate Vocab as it will take big time and Space but We can Expl
# Pass the vocab as argument and the function will do its work.
# To make it work for Our task and better understanding of n_range i refered
# https://stats.stackexchange.com/questions/291297/countvectorizer-as-n-gram-presence-and-count-feature
```

In [ ]:
```
# Lets Create Bigram, Trigram and 4 gram (as stated in the vid by Say no to Overfitting approach they went till
# We will store the vocab in List with custom made loops

# To understand the working I experimented with this code
# a = ['00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20', '28 00 1C 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C
# from sklearn.feature_extraction.text import CountVectorizer
# vect = CountVectorizer(ngram_range=(2, 2), vocabulary = finalBigram)
# k = vect.transform(a)
# print(k.toarray())
# import numpy as np
# keys = ['little inspiration', 'time time', 'occasion', 'creativity', 'innovation']
# text = ('Everyone needs a little inspiration from time time to time')
# cv1 = CountVectorizer(vocabulary = keys, ngram_range=(2,2))
# data = cv1.fit_transform([text]).toarray()
# vec1 = np.array(data)
# print(vec1)
```

In [5]:
```
# Bi GRAMS
byte_feature_string = "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,1
k = byte_feature_string.split(",")
# k[0]
finalBigram = []
for i in range(len(k)):
    for j in range(len(k)):
        f = k[i] +" "+ k[j]
        finalBigram.append(f)

# Tri Gram
byte_feature_string = "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,1
k = byte_feature_string.split(",")
finalTrigram = []

for i in range(len(k)):
    for j in range(len(k)):
        for d in range(len(k)):
            f = k[i] +" "+ k[j] + " " + k[d]
            finalTrigram.append(f)


# Created Vocab
print(finalBigram[:3], finalTrigram[:3])
```

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

In [6]:
```
# Pickle to save all those Vocabs
# https://www.datacamp.com/community/tutorials/pickle-python-tutorial#pickling
import pickle

outfile = open('finalBigram', 'wb')
pickle.dump(finalBigram,outfile)
outfile.close()

outfile = open('finalTrigram', 'wb')
pickle.dump(finalTrigram,outfile)
outfile.close()

###############
# To Retrieve #
##############

# infile = open('finalBigram','rb')
# finalBigram = pickle.load(infile)
# infile.close()

# infile = open('finalTrigram','rb')
# finalTrigram = pickle.load(infile)
# infile.close()
```

In [7]:
```
len(finalBigram)
```

Out[7]: 66049

In [ ]:
```
import scipy.sparse

# Lets Count no. of values in each Category, As we know We will get huge Sparse Matrices So we will
# save them in harddisk so it will be easy to them retrieve
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
bow = CountVectorizer(ngram_range=(2, 2), vocabulary = finalBigram)
total_byte_files = 10868
# Now to store a Csr matrix we cann use https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.save_n
# Intitalizing Empty Matrix
bigram_Matrix = scipy.sparse.csr_matrix((total_byte_files, len(finalBigram)))
# https://stackoverflow.com/questions/8369219/how-to-read-a-text-file-into-a-string-variable-and-strip-newlines
i = 0
for index, file in tqdm(enumerate(os.listdir('./byteFiles'))):
    f = open('./byteFiles/' + file)
    k = bow.fit_transform([f.read().replace('\n', ' ')])
#     for every separate file (index here) we are storing its sparse vector/matrix
    bigram_Matrix[index] = scipy.sparse.csr_matrix(k)
    f.close()
#     print(index)
```

```
3548it [2:14:13,  3.25s/it]
```

In [ ]:
```
import scipy.sparse
scipy.sparse.save_npz('bigram_Matrix.npz', bigram_Matrix)
```

In [52]:
```
bigram_Matrix.shape
# df_bigram = pd.DataFrame(bigram_Matrix.toarray()) doing this makes processing real slow
df_bigram.shape
```

Out[52]:
```
(10868, 66049)
```

In [2]:
```
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.save_npz.html
import scipy.sparse
bigram_Matrix = scipy.sparse.load_npz('bigram_Matrix.npz')
```

In [28]:
```
# https://docs.python.org/2/library/array.html
# https://www.geeksforgeeks.org/working-images-python/
#
import array
from PIL import Image, ImageDraw
import imageio

for asmfile in os.listdir("./asmFiles"):
#     here we are first spiliting the name with extension
    asmfile_name = asmfile.split('.')[0]
    asmfile_open = codecs.open("./asmFiles/" + asmfile, 'rb')
    asmfile_len = os.path.getsize("./asmFiles/" + asmfile)
#     getting file length
    width = int(asmfile_len ** 0.5)
#     reducing the width
    rem = int(asmfile_len / width)
    img_arr = array.array('B')
#     B : unsigned char  https://docs.python.org/3.2/library/array.html
    img_arr.frombytes(asmfile_open.read())
#    from bytes: Appends items from the string, interpreting the string as an array of machine values
#     (as if it had been read from a file using the fromfile() method).
    asmfile_open.close()
    reshaped_array = np.reshape(img_arr[:width * width], (width, width))
#     square image
    reshaped_array = np.uint8(reshaped_array)
#     https://stackoverflow.com/a/56446053/7437264
    imageio.imwrite('./asm_images/' + asmfile_name + '.png',reshaped_array)
```

In [5]:
```
# Now as per the youtube video of Winner solution "Say no to overfitting" they said first 200 features are impo
first_200 = np.zeros((10868, 200))
import cv2
for i, asmfile in tqdm(enumerate(os.listdir("asmFiles"))):
    image = cv2.imread("asm_images/" + asmfile.split('.')[0] + '.png')
    image_array = image.flatten()[:200]
    first_200[i, :] += image_array
```

```
10868it [23:18,  7.77it/s]
```

In [24]:
```
# Now lets Normalize the data
from sklearn.preprocessing import normalize
image_features_200 = []
for i in range(200):
    image_features_200.append('pixle' + str(i))
image_final = pd.DataFrame(normalize(first_200, axis = 0), columns = image_features_200)
image_final['ID'] = result.ID
image_final.head(2)
```

Out[24]:

	pixle0	pixle1	pixle2	pixle3	pixle4	pixle5	pixle6	pixle7	pixle8	pixle9	...	pixle191	pixle192	pixle193	pixle194	pix
0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.00832	0.00832	0.00832	0.007913	...	0.009593	0.009593	0.009593	0.009593	0.0(
1	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.00832	0.00832	0.00832	0.007913	...	0.009593	0.009593	0.009593	0.009593	0.0(

2 rows × 201 columns

In [26]:
```
# make pickle of image features and pickle of result_x for future
image_final.to_pickle("image_features_200")
result_x.to_pickle("result_x")
```

In [2]:
```
import scipy.sparse
bigram_Matrix = scipy.sparse.load_npz('bigram_Matrix.npz')
```

In [3]:
```
# Lets combine bigrams and image features and result_x
# import scipy.sparse
# bigram_Matrix = scipy.sparse.load_npz('bigram_Matrix.npz')

result_x = pd.read_pickle("result_x")
image_final = pd.read_pickle("image_features_200")
result_x = result_x.drop('Unnamed: 0', axis = 1)
image_final = image_final.drop('ID', axis = 1)

image_final.shape, result_x.shape, bigram_Matrix.shape
```

Out[3]:
```
((10868, 200), (10868, 307), (10868, 66049))
```

In [14]:
```
from scipy.sparse import hstack
total_data = hstack((result_x,image_final,bigram_Matrix))
print(total_data.shape)
```

```
(10868, 66556)
```

In [13]:
```
# import scipy.sparse
# scipy.sparse.save_npz('total_data.npz', total_data)

# Well to my suprise, while saving it in form of Dataframe it took 3GB of Space but when stored in sparse form
# It took 1 GB of space, thats 1/3rd.
```

In [2]:
```
import scipy.sparse
total_data = scipy.sparse.load_npz('total_data.npz')
```

In [5]:
```
# total_data =  scipy.sparse.load_npz('total_data.npz')
from scipy.sparse import hstack
total_data = hstack((result_x,image_final))
print(total_data.shape)
```

```
(10868, 507)
```

In [ ]:
```

```

In [6]:
```
# converting pd to csv so it might be useful later too
# bigram_imageFeatures.to_csv("bigram_imageFeatures.csv")
# bigram_imageFeatures = pd.read_csv("bigram_imageFeatures.csv")
```

In [2]:
```
# import pickle
# filename = 'data_y'
# outfile = open(filename,'wb')
# pickle.dump(data_y,outfile)
# outfile.close()
filename = 'data_y'
infile = open(filename,'rb')
data_y = pickle.load(infile)
infile.close()
```

In [5]:
```
# Now lets split the data and start some modelling
# data_y = result['Class']
X_train_complete, X_test_complete, y_train_complete, y_test_complete = train_test_split(total_data, data_y,stra
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_t
X_train_complete, X_cv_complete, y_train_complete, y_cv_complete = train_test_split(X_train_complete, y_train_c
```

In [6]:
```
# Was Having TerminatedWorkerError While doing Random Search so have to redce the params
```

In [7]:
```
X_train_complete.shape, y_train_complete.shape
```

Out[7]:
```
((6955, 66556), (6955,))
```

In [ ]:

In [18]:

```
# Random Forest
from datetime import datetime
start = datetime.now()

alpha=[10,20,50,70]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    r_cfl=RandomForestClassifier(n_estimators=2000,random_state=42,n_jobs=-1, max_depth = i, verbose = 1)
    r_cfl.fit(X_train_complete,y_train_complete)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_complete, y_train_complete)
    predict_y = sig_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for depth = ',alpha[i],'is',cv_log_error_array[i])

print(datetime.now() - start)
```

```
  0%|          | 0/4 [00:00<?, ?it/s][Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:    6.1s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   11.0s
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed:   17.3s
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed:   25.0s
[Parallel(n_jobs=-1)]: Done 2000 out of 2000 | elapsed:   28.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:    7.0s
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed:   11.0s
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed:   15.9s
[Parallel(n_jobs=-1)]: Done 2000 out of 2000 | elapsed:   17.8s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
```

In [19]:

```
# Random Forest with image and result_x only
from datetime import datetime
start = datetime.now()

alpha=[100,500,1000,2000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1, max_depth = 50, verbose = 1)
    r_cfl.fit(X_train_complete,y_train_complete)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_complete, y_train_complete)
    predict_y = sig_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for depth = ',alpha[i],'is',cv_log_error_array[i])

print(datetime.now() - start)
```

```
  0%|          | 0/4 [00:00<?, ?it/s][Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    1.7s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    1.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    1.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    1.1s finished
```

log_loss for depth = 100 is 0.037334915374394706

log_loss for depth = 500 is 0.03578887524971817
log_loss for depth = 1000 is 0.03555860069809751
log_loss for depth = 2000 is 0.035505673816373935
0:03:05.949466

In [13]:

```
best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

In [8]:

```
# Random Forest with image, result_x and bigram
from datetime import datetime
start = datetime.now()
r_cfl=RandomForestClassifier(n_estimators=2000,random_state=42,n_jobs=-1, verbose = 1, max_depth = 10)
r_cfl.fit(X_train_complete,y_train_complete)
sig_clf = CalibratedClassifierCV(r_2cfl, method="sigmoid")
sig_clf.fit(X_train_complete, y_train_complete)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   24.2s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   56.4s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 2000 out of 2000 | elapsed:  4.3min finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   15.9s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   37.1s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 2000 out of 2000 | elapsed:  2.8min finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    1.3s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    6.9s
```

In [9]:
```
predict_y_train = sig_clf.predict_proba(X_train_complete)
print ('For values of best depth = ', 10, "The train log loss is:",log_loss(y_train_complete, predict_y_train))
predict_y_cv = sig_clf.predict_proba(X_cv_complete)
print('For values of best depth = ', 10, "The cross validation log loss is:",log_loss(y_cv_complete, predict_y_
predict_y_test = sig_clf.predict_proba(X_test_complete)
print('For values of best depth = ', 10, "The test log loss is:",log_loss(y_test_complete, predict_y_test))

print("Time taken : ",datetime.now() - start)
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    4.0s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:   19.8s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:   47.5s
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:  1.4min
[Parallel(n_jobs=8)]: Done 1234 tasks      | elapsed:  2.3min
[Parallel(n_jobs=8)]: Done 1784 tasks      | elapsed:  3.3min
[Parallel(n_jobs=8)]: Done 2000 out of 2000 | elapsed:  3.7min finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    3.9s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:   20.5s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:   47.6s
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:  1.4min
[Parallel(n_jobs=8)]: Done 1234 tasks      | elapsed:  2.3min
[Parallel(n_jobs=8)]: Done 1784 tasks      | elapsed:  3.2min
[Parallel(n_jobs=8)]: Done 2000 out of 2000 | elapsed:  3.6min finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    4.2s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:   20.2s
```

For values of best depth = 10 The train log loss is: 0.09333947958038523
For values of best depth = 10 The cross validation log loss is: 0.1530765067678385
For values of best depth = 10 The test log loss is: 0.1580199735425942
Time taken : 0:34:44.273779

In [11]:
```
%autosave 1
```

```
Autosaving every 1 seconds
```

In [ ]:
```
import scipy.sparse
bigram_Matrix = scipy.sparse.load_npz('bigram_Matrix.npz')
bigram_Matrix.shape, data_y.shape
```

In [5]:
```
from sklearn.feature_selection import SelectKBest, chi2
X_new = SelectKBest(chi2, k=500).fit_transform(bigram_Matrix, data_y)
image_new = SelectKBest(chi2, k=100).fit_transform(image_final, data_y)
results_new = SelectKBest(chi2, k=200).fit_transform(result_x, data_y)
X_new_200 = SelectKBest(chi2, k=200).fit_transform(bigram_Matrix, data_y)
```

In [6]:
```
X_new.shape, image_new.shape, results_new.shape
```

Out[6]:
```
((10868, 500), (10868, 100), (10868, 200))
```

In [9]:
```
import scipy.sparse
scipy.sparse.save_npz('X_new_200.npz', X_new_200)
scipy.sparse.save_npz('X_new.npz', X_new)
```

In [10]:
```
from scipy.sparse import hstack
total_data = hstack((results_new,image_new,X_new))
print(total_data.shape)
```

```
(10868, 800)
```

In [11]:
```
X_train_complete, X_test_complete, y_train_complete, y_test_complete = train_test_split(total_data, data_y,stra
X_train_complete, X_cv_complete, y_train_complete, y_cv_complete = train_test_split(X_train_complete, y_train_c
```

In [12]:
```
X_train_complete.shape, y_train_complete.shape
```

Out[12]:
```
((6955, 800), (6955,))
```

In [15]:
```
x_cfl = XGBClassifier()
prams={'n_estimators' : [50, 100, 150,200, 300, 500, 1000, 2000] }
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,cv = 2)
random_cfl.fit(X_train_complete, y_train_complete)
```

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of  16 | elapsed: 10.1min remaining: 43.6min
[Parallel(n_jobs=-1)]: Done   5 out of  16 | elapsed: 13.4min remaining: 29.5min
[Parallel(n_jobs=-1)]: Done   7 out of  16 | elapsed: 16.1min remaining: 20.7min
[Parallel(n_jobs=-1)]: Done   9 out of  16 | elapsed: 26.2min remaining: 20.4min
[Parallel(n_jobs=-1)]: Done  11 out of  16 | elapsed: 36.8min remaining: 16.7min
[Parallel(n_jobs=-1)]: Done  13 out of  16 | elapsed: 50.2min remaining: 11.6min
[Parallel(n_jobs=-1)]: Done  16 out of  16 | elapsed: 73.9min finished
```

Out[15]:
```
RandomizedSearchCV(cv=2, 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,
```

In [ ]:
```
alpha = [100, 500 2000]
```

In [21]:

```
from datetime import datetime
start = datetime.now()
# reconfirming random search
alpha = [50, 100, 500, 2000]
cv_log_error_array=[]

for i in tqdm(alpha):
    r_cfl=XGBClassifier(n_estimators=i,random_state=42,n_jobs=-1, verbose = 1)
    r_cfl.fit(X_train_complete,y_train_complete)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid", cv='prefit')
    sig_clf.fit(X_train_complete, y_train_complete)
    predict_y = sig_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for depth = ',alpha[i],'is',cv_log_error_array[i])

print(datetime.now() - start)
```

```
100%|██████████| 4/4 [1:29:07<00:00, 1336.90s/it]

log_loss for depth =  50 is 0.03560146693148561
log_loss for depth =  100 is 0.028788451295919667
log_loss for depth =  500 is 0.027825764223068
log_loss for depth =  2000 is 0.027810246622584436
1:29:07.604328
```

In [13]:

```
from datetime import datetime
start = datetime.now()
x_cfl=XGBClassifier(n_estimators=100)
x_cfl.fit(X_train_complete,y_train_complete,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid", cv='prefit')
sig_clf.fit(X_train_complete, y_train_complete)
print("Time taken : ",datetime.now() - start)
```

```
Time taken :  0:02:21.760478
```

In [14]:
```
predict_y_train = sig_clf.predict_proba(X_train_complete)
print ("The train log loss is:",log_loss(y_train_complete, predict_y_train))
predict_y_cv = sig_clf.predict_proba(X_cv_complete)
print("The cross validation log loss is:",log_loss(y_cv_complete, predict_y_cv))
predict_y_test = sig_clf.predict_proba(X_test_complete)
print("The test log loss is:",log_loss(y_test_complete, predict_y_test))
print("Time taken : ",datetime.now() - start)
```

```
The train log loss is: 0.0014912694104367488
The cross validation log loss is: 0.01910648786261139
The test log loss is: 0.04231201174408598
Time taken :  0:02:23.867292
```

In [15]:
```
# %autosave 600
loss_train = log_loss(y_train_complete, predict_y_train)
loss_cv = log_loss(y_cv_complete , predict_y_cv)
loss_test = log_loss(y_test_complete , predict_y_test)
```

In [22]:
```
x_cfl = XGBClassifier(n_estimators=500, n_jobs = -1)
```

In [ ]:
```
%autosave 600
```

In [26]:
```

```

In [ ]:
```
alpha=[50,100, 500,2000]
cv_log_error_array=[]
for i in alpha:
    clf=XGBClassifier(n_estimators=i, n_jobs = -1)
    stack_clf = StackingClassifier(classifiers=[x_cfl,x_cfl,x_cfl,x_cfl], meta_classifier=clf)
    stack_clf.fit(X_train_complete, y_train_complete)
    predict_y = stack_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete, predict_y, eps=1e-15))
#     print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])
```

In [ ]:
```
for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)
```

In [ ]:

In [ ]:
```
meta_clf = XGBClassifier(n_estimators=alpha[best_alpha], n_jobs=-1)
stack_clf=StackingClassifier(classifiers=[clf, clf, clf, clf, clf], meta_classifier=meta_clf)
print("Fitting up!!!")
stack_clf.fit(X_train_complete,y_train_complete)

predict_y_train_stack = stack_clf.predict_proba(X_train_complete)
loss_train = log_loss(y_train_complete, predict_y_train_stack)
# print ('For values of best estimators = ', alpha[best_estimators], "The train log loss is:",)

predict_y_cv_stack = stack_clf.predict_proba(X_cv_complete)
# print('For values of best estimators = ', alpha[best_estimators], "The cross validation log loss is:",)
loss_cv = log_loss(y_cv_complete , predict_y_cv_stack)

predict_y_test_stack = stack_clf.predict_proba(X_test_complete)
# print('For values of best estimators = ', alpha[best_estimators], "The test log loss is:",)
loss_test = log_loss(y_test_complete , predict_y_test_stack)
print("Time taken : ",datetime.now() - start)
```

In [ ]:
```
print("The Train Loss", loss_train)
print("The CV Loss", loss_cv)
print("The Test Loss", loss_test)
%autosave 600
```

In [ ]:
```
!pip3 install lightgbm
```

In [27]:
```
from lightgbm import LGBMClassifier
```

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

alpha=[10,20,30, 50,100]
cv_log_error_array=[]
for i in tqdm(alpha):
    lgbm_clf = LGBMClassifier(n_estimators=i, n_jobs=-1)
    lgbm_clf.fit(X_train_complete ,y_train_complete )
    sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid", cv = 'prefit')
    sig_clf.fit(X_train_complete , y_train_complete)
    predict_y = sig_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete , predict_y, labels=lgbm_clf.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for n_estimators = ',alpha[i],'is',cv_log_error_array[i])

best_estimators = np.argmin(cv_log_error_array)

lgbm_clf=LGBMClassifier(n_estimators=alpha[best_estimators],nthread=-1,n_jobs=-1)
lgbm_clf.fit(X_train_complete ,y_train_complete ,verbose=True)

sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid")
sig_clf.fit(X_train_complete , y_train_complete )

predict_y = sig_clf.predict_proba(X_train_complete)
print ('For values of best estimators = ', alpha[best_estimators], "The train log loss is:",log_loss(y_train_co

predict_y = sig_clf.predict_proba(X_cv_complete)
print('For values of best estimators = ', alpha[best_estimators], "The cross validation log loss is:",log_loss(

predict_y = sig_clf.predict_proba(X_test_complete)
print('For values of best estimators = ', alpha[best_estimators], "The test log loss is:",log_loss(y_test_compl
print("Time taken : ",datetime.now() - start)
```

```
100%|██████████| 5/5 [00:26<00:00,  5.25s/it]

log_loss for n_estimators =  10 is 0.06021080098743783
log_loss for n_estimators =  20 is 0.05967159009300233
log_loss for n_estimators =  30 is 0.054672145211381536
log_loss for n_estimators =  50 is 0.05297264337452534
log_loss for n_estimators =  100 is 0.05951021355906826

For values of best estimators =  50 The train log loss is: 0.01395830839101834
```

```
For values of best estimators =  50 The train log loss is: 0.01395830839101834
For values of best estimators =  50 The cross validation log loss is: 0.052861009618971895
For values of best estimators =  50 The test log loss is: 0.04664462644889236
Time taken :  0:00:48.091282
```

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

alpha=[500, 1000, 2000]
cv_log_error_array=[]
for i in tqdm(alpha):
    lgbm_clf = LGBMClassifier(n_estimators=i, n_jobs=-1)
    lgbm_clf.fit(X_train_complete ,y_train_complete )
    sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid", cv = 'prefit')
    sig_clf.fit(X_train_complete , y_train_complete)
    predict_y = sig_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete , predict_y, labels=lgbm_clf.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for n_estimators = ',alpha[i],'is',cv_log_error_array[i])

best_estimators = np.argmin(cv_log_error_array)

lgbm_clf=LGBMClassifier(n_estimators=alpha[best_estimators],nthread=-1,n_jobs=-1)
lgbm_clf.fit(X_train_complete ,y_train_complete ,verbose=True)

sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid")
sig_clf.fit(X_train_complete , y_train_complete )

predict_y = sig_clf.predict_proba(X_train_complete)
print ('For values of best estimators = ', alpha[best_estimators], "The train log loss is:",log_loss(y_train_co

predict_y = sig_clf.predict_proba(X_cv_complete)
print('For values of best estimators = ', alpha[best_estimators], "The cross validation log loss is:",log_loss(

predict_y = sig_clf.predict_proba(X_test_complete)
print('For values of best estimators = ', alpha[best_estimators], "The test log loss is:",log_loss(y_test_compl
print("Time taken : ",datetime.now() - start)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
 33%|███       | 1/3 [00:24<00:48, 24.23s/it]
 67%|██████    | 2/3 [00:59<00:27, 27.48s/it]
100%|██████████| 3/3 [01:53<00:00, 37.81s/it]

log_loss for n_estimators =  500 is 0.057154691913007274
log_loss for n_estimators =  1000 is 0.05784864673145087
```

```
log_loss for n_estimators =  2000 is 0.05813145109726991 5
```

In [ ]:
```
from datetime import datetime
start = datetime.now()
print("Time taken : ",datetime.now() - start)
```

In [ ]:
```

```

In [26]:
```
lgb_clf_1=LGBMClassifier(n_estimators=50, n_jobs=-1, nthread=-1)

lgb_clf_2=LGBMClassifier(n_estimators=60, n_jobs=-1, nthread=-1)

xgb_clf_1 = XGBClassifier(n_estimators=500, n_jobs=-1, nthread=-1)

xgb_clf_2 = XGBClassifier(n_estimators=150, n_jobs=-1, nthread=-1)
```

In [27]:
```
from datetime import datetime
start = datetime.now()
cv_log_error_array=[]
alpha = [150, 500, 1000, 20000]
for i in tqdm(alpha):
    clf=XGBClassifier(n_estimators=i, n_jobs = -1, nthreads= -1)
    stack_clf = StackingClassifier(classifiers=[lgb_clf_1, lgb_clf_2, xgb_clf_1, xgb_clf_2], meta_classifier=cl
    stack_clf.fit(X_train_complete, y_train_complete)
    predict_y = stack_clf.predict_proba(X_cv_complete)
    cv_log_error_array.append(log_loss(y_cv_complete, predict_y, eps=1e-15))
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
 25%|██▌       | 1/4 [02:20<07:02, 140.89s/it]
 50%|█████     | 2/4 [04:41<04:41, 140.80s/it]
 75%|███████▌  | 3/4 [07:03<02:21, 141.04s/it]
100%|██████████| 4/4 [09:56<00:00, 149.02s/it]
```

In [ ]:
```
for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)
```

In [24]:
```
meta_clf = XGBClassifier(n_estimators=alpha[best_alpha], n_jobs=-1)
stack_clf=StackingClassifier(classifiers=[lgb_clf_1, lgb_clf_2, xgb_clf_1, xgb_clf_2], meta_classifier=meta_clf
stack_clf.fit(X_train_complete,y_train_complete)

predict_y_train_stack = stack_clf.predict_proba(X_train_complete)
loss_train = log_loss(y_train_complete, predict_y_train_stack)
print ('For values of best estimators = ', alpha[best_estimators], "The train log loss is:",loss_train)

predict_y_cv_stack = stack_clf.predict_proba(X_cv_complete)
loss_cv = log_loss(y_cv_complete , predict_y_cv_stack)
print('For values of best estimators = ', alpha[best_estimators], "The cross validation log loss is:",loss_cv)

predict_y_test_stack = stack_clf.predict_proba(X_test_complete)
loss_test = log_loss(y_test_complete , predict_y_test_stack)
print('For values of best estimators = ', alpha[best_estimators], "The test log loss is:",loss_test)
print("Time taken : ",datetime.now() - start)
```

```
For values of best estimators =  50 The train log loss is: 0.010309939648310630
For values of best estimators =  50 The cross validation log loss is: 0.06289770497280146
For values of best estimators =  50 The test log loss is: 0.061757485593027385
Time taken :  0:06:06.179118
```

In [ ]:
```

```

In [3]:
```
# https://docs.python.org/2/library/array.html
# https://www.geeksforgeeks.org/working-images-python/
#
import array
from PIL import Image, ImageDraw
import imageio

for asmfile in os.listdir("./byteFiles"):
#     here we are first spiliting the name with extension
    asmfile_name = asmfile.split('.')[0]
    asmfile_open = codecs.open("./byteFiles/" + asmfile, 'rb')
    asmfile_len = os.path.getsize("./byteFiles/" + asmfile)
#     getting file length
    width = int(asmfile_len ** 0.5)
#     reducing the width
    rem = int(asmfile_len / width)
    img_arr = array.array('B')
#     B : unsigned char  https://docs.python.org/3.2/library/array.html
    img_arr.frombytes(asmfile_open.read())
#    from bytes: Appends items from the string, interpreting the string as an array of machine values
#     (as if it had been read from a file using the fromfile() method).
    asmfile_open.close()
    reshaped_array = np.reshape(img_arr[:width * width], (width, width))
#     square image
    reshaped_array = np.uint8(reshaped_array)
#     https://stackoverflow.com/a/56446053/7437264
    imageio.imwrite('./byte_images/' + asmfile_name + '.png',reshaped_array)

# Now as per the youtube video of Winner solution "Say no to overfitting" they said first 200 features are impo
first_200 = np.zeros((10868, 200))
import cv2
for i, asmfile in tqdm(enumerate(os.listdir("byteFiles"))):
    image = cv2.imread("byte_images/" + asmfile.split('.')[0] + '.png')
    image_array = image.flatten()[:200]
    first_200[i, :] += image_array
```

```
10868it [08:19, 21.74it/s]
```

In [5]:
```
# result=pd.read_csv("result.csv")
```

In [6]:
```
# Now lets Normalize the data
from sklearn.preprocessing import normalize
image_features_200 = []
for i in range(200):
    image_features_200.append('pixle' + str(i))
byteimage_final = pd.DataFrame(normalize(first_200, axis = 0), columns = image_features_200)
byteimage_final['ID'] = result.ID
byteimage_final.head(2)
```

Out[6]:

	pixle0	pixle1	pixle2	pixle3	pixle4	pixle5	pixle6	pixle7	pixle8	pixle9	...	pixle191	pixle192	pixle193	pixle194
0	0.009506	0.009506	0.009506	0.009424	0.009424	0.009424	0.009622	0.009622	0.009622	0.009466	...	0.009592	0.009592	0.009592	0.009592
1	0.009506	0.009506	0.009506	0.009424	0.009424	0.009424	0.009622	0.009622	0.009622	0.009466	...	0.009592	0.009592	0.009592	0.009592

2 rows × 201 columns

Adding some features from Dchad

In [3]:
```
asm_rowstat = pd.read_csv("dchad_data/train-asm-rowstats.csv")
```

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

Out[4]:

	filename	mean	std	min	max	total	logtotal
0	01IsoiSMh5gxyDYTl4CB	3220.544554	14985.141846	0.0	87555.0	4.225432e+12	29.072143
1	01SuzwMJEIXsK7A8dQbl	337.425743	990.494123	0.0	5817.0	1.944147e+09	21.388089
2	01azqd4InC7m9JpocGv5	27635.227723	191333.687686	0.0	1367070.0	7.228451e+15	36.516801
3	01jsnpXSAlgw6aPeDxrU	1411.188119	9219.899598	0.0	65928.0	8.577900e+11	27.477625
4	01kcPWA9K2BOxQeS5Rju	23.405941	54.519937	0.0	445.0	5.678602e+05	13.249631

In [5]:
```
image_asm_rowstat = pd.read_csv("dchad_data/train-image-asm-rowstats.csv")
image_asm_rowstat.head()
```

Out[5]:

	filename	tr_mean	tr_std	tr_min	tr_max	tr_total	tr_logtotal
0	01IsoiSMh5gxyDYTl4CB	73.60	44.988888	9.0	124.0	410586.583033	12.925342
1	01SuzwMJEIXsK7A8dQbl	46.72	40.429282	9.0	124.0	234218.152663	12.364008
2	01azqd4InC7m9JpocGv5	46.72	40.429282	9.0	124.0	234218.152663	12.364008
3	01jsnpXSAlgw6aPeDxrU	46.72	40.429282	9.0	124.0	234218.152663	12.364008
4	01kcPWA9K2BOxQeS5Rju	48.40	41.941518	9.0	124.0	251716.212779	12.436058

In [6]:
```
image_asm_rowstat.rename(columns={'filename':'ID'}, inplace=True)
asm_rowstat.rename(columns={'filename':'ID'}, inplace=True)
```

In [7]:
```
asm_features_all = pd.merge(image_asm_rowstat, asm_rowstat, on = "ID")
asm_features_all.head(2)
```

Out[7]:

	ID	tr_mean	tr_std	tr_min	tr_max	tr_total	tr_logtotal	mean	std	min	max	total	
0	01IsoiSMh5gxyDYTl4CB	73.60	44.988888	9.0	124.0	410586.583033	12.925342	3220.544554	14985.141846	0.0	87555.0	4.225432e+12	2
1	01SuzwMJEIXsK7A8dQbl	46.72	40.429282	9.0	124.0	234218.152663	12.364008	337.425743	990.494123	0.0	5817.0	1.944147e+09	2

In [31]:
```
asm_features_all[asm_features_all["logtotal"] < 0]
```

Out[31]:

	ID	tr_mean	tr_std	tr_min	tr_max	tr_total	tr_logtotal	mean	std	min	max	total	logtotal
1769	58kxhXouHzFd4g3rmlnB	48.71	28.969610	9.0	121.0	170744.271967	12.047922	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
2427	6tfw0xSL2FNHOCJBdlaA	49.42	29.114605	9.0	121.0	174100.094982	12.067386	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
6536	lidxQvXrlBkWPZAfcqKT	48.28	29.143698	9.0	121.0	170253.988316	12.045047	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
7248	a9olzfw03ED4lTBCt52Y	49.23	29.494685	9.0	121.0	175694.826609	12.076504	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
8081	cf4nzsoCmudt1kwleOTI	48.70	29.286636	9.0	121.0	172577.362222	12.058601	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
8208	d0iHC6ANYGon7myPFzBe	48.38	29.172001	9.0	121.0	170772.311841	12.048086	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
8432	da3XhOZzQEbKVtLgMYWv	48.99	29.376516	9.0	121.0	174137.818251	12.067602	0.009901	0.099504	0.0	1.0	0.000985	-6.922681
9044	fRLS3aKkijp4GH0Ds6Pv	48.69	29.285669	9.0	121.0	172536.225580	12.058362	0.009901	0.099504	0.0	1.0	0.000985	-6.922681

In [8]:
```
entropy_image = pd.read_csv("merged_image_entropy")
entropy_image.shape
```

Out[8]:
```
(10868, 204)
```

In [9]:
```
entropy_image_asm = pd.merge(entropy_image, asm_features_all, on = "ID")
entropy_image_asm.head(2)
```

Out[9]:

	Unnamed: 0	pixle0	pixle1	pixle2	pixle3	pixle4	pixle5	pixle6	pixle7	pixle8	...	tr_min	tr_max	tr_total	tr_logtotal
0	0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.00832	0.00832	0.00832	...	9.0	124.0	234218.152663	12.364008
1	1	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.00832	0.00832	0.00832	...	9.0	124.0	410586.583033	12.925342

2 rows × 216 columns

In [10]:
```
entropy_image_asm = entropy_image_asm.drop(['Unnamed: 0', 'ID', 'filesize'], axis = 1)
```

In [12]:
```
entropy_image_asm.to_csv("entropy_image_asm")
```

In [ ]:
```
# All the Features we have
# ASM features row stat, entropy, Asm image features 200 : entropy_image_asm
# Byte Image Features : byteimage_final
# Features Unigram and Filesize : result_x
# Bigram byte files : bigram_Matrix
# Lets Combine Them and perform Modelling
```

In [13]:
```
import scipy.sparse
entropy_image_asm = pd.read_csv("entropy_image_asm")
byteimage_final = pd.read_pickle("byteimage_final_200")
result_x = pd.read_pickle("result_x")
bigram_Matrix = scipy.sparse.load_npz('bigram_Matrix.npz')
data_y = pd.read_pickle("data_y")
```

In [16]:
```
byteimage_final = byteimage_final.drop("ID", axis = 1)
result_x = result_x.drop('Unnamed: 0', axis = 1)
```

In [32]:
```
entropy_image_asm = entropy_image_asm.drop('logtotal', axis = 1)
```

In [33]:
```
entropy_image_asm.shape, byteimage_final.shape, result_x.shape, bigram_Matrix.shape
```

Out[33]:
```
((10868, 213), (10868, 200), (10868, 307), (10868, 66049))
```

In [34]:
```
from sklearn.feature_selection import SelectKBest, chi2
X_new = SelectKBest(chi2, k=500).fit_transform(bigram_Matrix, data_y)
entropy_new = SelectKBest(chi2, k=150).fit_transform(entropy_image_asm, data_y)
result_x_new = SelectKBest(chi2, k=200).fit_transform(result_x, data_y)
byteimage_final_new = SelectKBest(chi2, k=100).fit_transform(byteimage_final, data_y)
```

In [35]:
```
X_new.shape, entropy_new.shape, result_x_new.shape, byteimage_final_new.shape
```

Out[35]:
```
((10868, 500), (10868, 150), (10868, 200), (10868, 100))
```

In [36]:
```
from scipy.sparse import hstack
total_data = hstack((result_x_new,entropy_new,byteimage_final_new,X_new))
print(total_data.shape)
```

```
(10868, 950)
```

In [ ]:

In [37]:
```
X_train, X_test, y_train, y_test = train_test_split(total_data, data_y,stratify=data_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)
```

In [38]:
```
X_train.shape, y_train.shape
```

Out[38]:
```
((6955, 950), (6955,))
```

In [42]:
```
from datetime import datetime
start = datetime.now()

alpha=[10,20,30, 50,100, 120, 140, 170, 200, 220, 500, 1000]
cv_log_error_array=[]
for i in tqdm(alpha):
    lgbm_clf = LGBMClassifier(n_estimators=i, n_jobs=-1)
    lgbm_clf.fit(X_train ,y_train)
    sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid", cv = 'prefit')
    sig_clf.fit(X_train , y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv , predict_y, labels=lgbm_clf.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for n_estimators = ',alpha[i],'is',cv_log_error_array[i])

best_estimators = np.argmin(cv_log_error_array)

lgbm_clf=LGBMClassifier(n_estimators=alpha[best_estimators],nthread=-1,n_jobs=-1)
lgbm_clf.fit(X_train ,y_train ,verbose=True)

sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid", cv = 'prefit')
sig_clf.fit(X_train , y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best estimators = ', alpha[best_estimators], "The train log loss is:",log_loss(y_train, p

predict_y = sig_clf.predict_proba(X_cv)
print('For values of best estimators = ', alpha[best_estimators], "The cross validation log loss is:",log_loss(

predict_y = sig_clf.predict_proba(X_test)
print('For values of best estimators = ', alpha[best_estimators], "The test log loss is:",log_loss(y_test , pre
print("Time taken : ",datetime.now() - start)
```

```
100%|██████████| 12/12 [03:10<00:00, 15.86s/it]

log_loss for n_estimators =  10 is 0.05729994061983901
log_loss for n_estimators =  20 is 0.04893373124017652
log_loss for n_estimators =  30 is 0.04356337059764339
log_loss for n_estimators =  50 is 0.029559941086457153
log_loss for n_estimators =  100 is 0.023495801694702625
log_loss for n_estimators =  120 is 0.024198947085786703
```

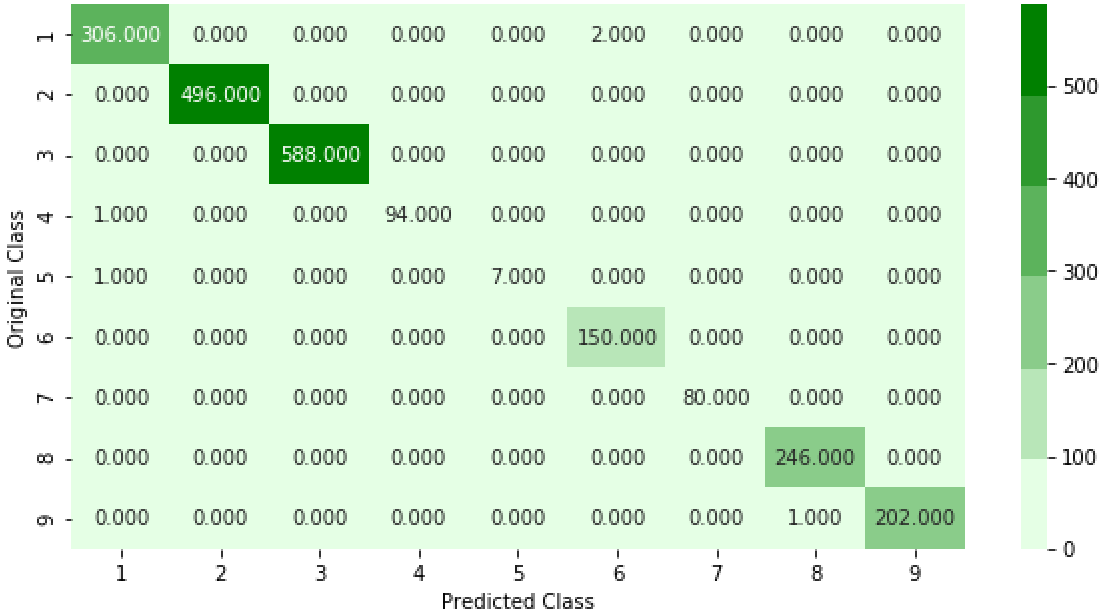
```
log_loss for n_estimators =  120 is 0.0241196917063730703
log_loss for n_estimators =  140 is 0.02517569308738004
log_loss for n_estimators =  170 is 0.02870337560939795
log_loss for n_estimators =  200 is 0.02923603487571137
log_loss for n_estimators =  220 is 0.02954594522334492
log_loss for n_estimators =  500 is 0.029790742655917645
log_loss for n_estimators =  1000 is 0.029805129243685224
For values of best estimators =  100 The train log loss is: 0.0012944546649396428
For values of best estimators =  100 The cross validation log loss is: 0.023495801694702625
For values of best estimators =  100 The test log loss is: 0.010368191752492434
Time taken :  0:03:24.585023
```

In [43]:

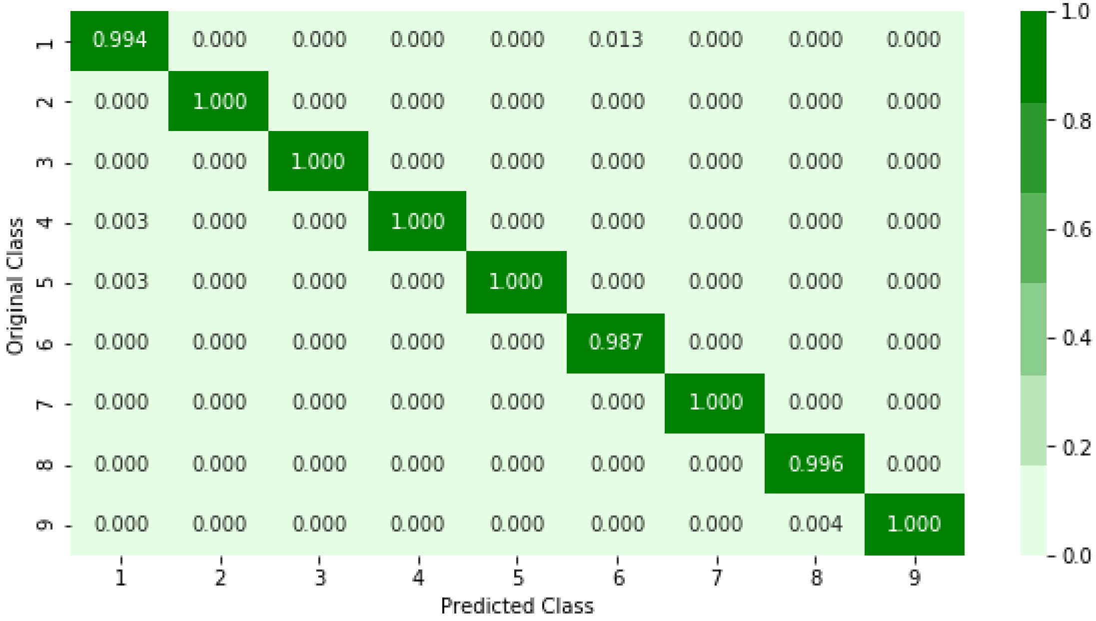
```
test_logloss_LGBM = log_loss(y_test , predict_y)
```

In [44]:
```
%matplotlib inline
plot_confusion_matrix(y_test, lgbm_clf.predict(X_test))
```

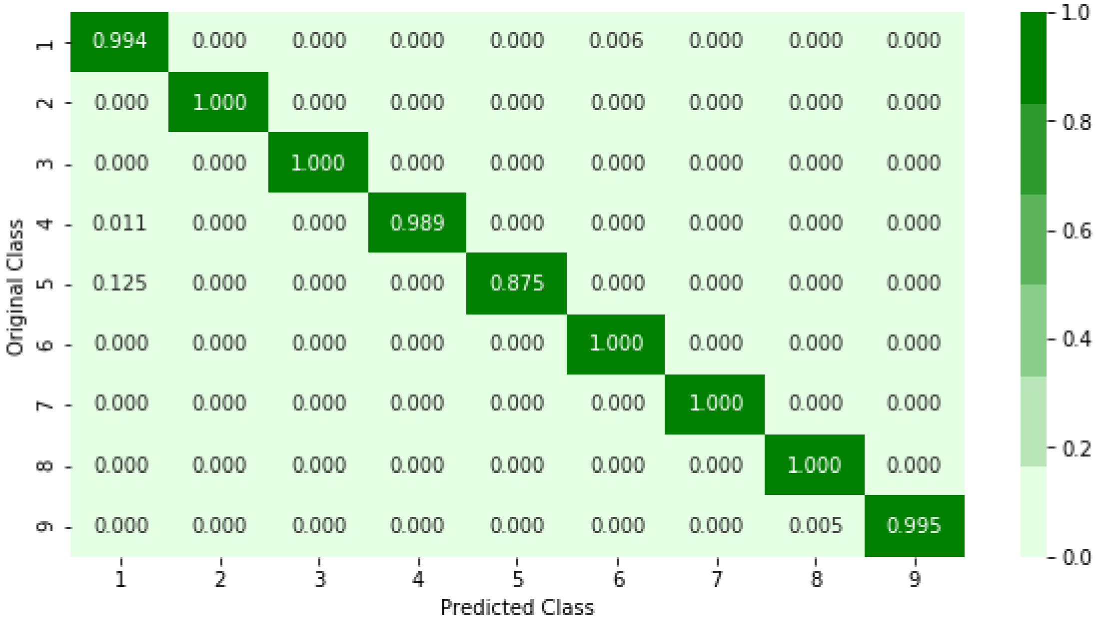
```
Number of misclassified points  0.22999080036798528
-------------------------------------------------- Confusion matrix --------------------------------------------------
```




```
-------------------------------------------------- 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 [49]:

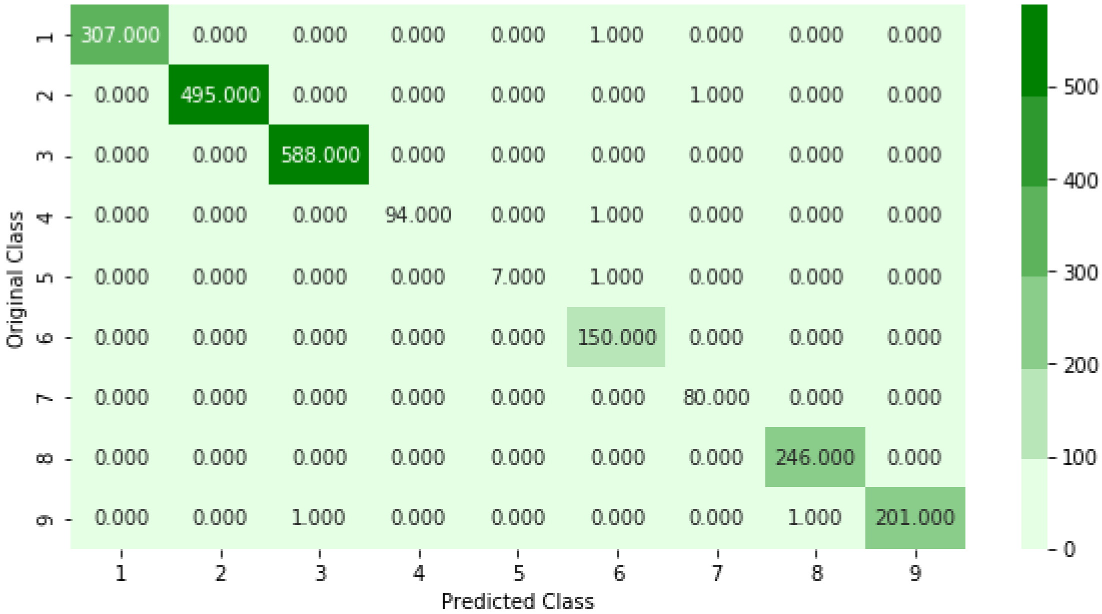
```
lgbm_clf=LGBMClassifier(n_estimators=100,nthread=-1,n_jobs=-1)
lgbm_clf.fit(X_train ,y_train ,verbose=True)

sig_clf = CalibratedClassifierCV(lgbm_clf, method="sigmoid", cv = 'prefit')
sig_clf.fit(X_train , y_train)
train_logloss_LGBM = log_loss(y_train, sig_clf.predict_proba(X_train))
```

In [45]:
```
from datetime import datetime
start = datetime.now()

alpha = [50, 100, 250, 500, 1000]
cv_log_error_array=[]

for i in tqdm(alpha):
    r_cfl=XGBClassifier(n_estimators=i,random_state=42,n_jobs=-1, verbose = 1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid", cv='prefit')
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for depth = ',alpha[i],'is',cv_log_error_array[i])

print(datetime.now() - start)

best_estimators = np.argmin(cv_log_error_array)

r_cfl=XGBClassifier(n_estimators=alpha[best_estimators],nthread=-1,n_jobs=-1)
r_cfl.fit(X_train ,y_train ,verbose=True)

sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid", cv = 'prefit')
sig_clf.fit(X_train , y_train)


predict_y = sig_clf.predict_proba(X_train)
print ('For values of best estimators = ', alpha[best_estimators], "The train log loss is:",log_loss(y_train, p

predict_y = sig_clf.predict_proba(X_cv)
print('For values of best estimators = ', alpha[best_estimators], "The cross validation log loss is:",log_loss(

predict_y = sig_clf.predict_proba(X_test)
print('For values of best estimators = ', alpha[best_estimators], "The test log loss is:",log_loss(y_test , pre
test_logloss_xgb = log_loss(y_test , predict_y)
print("Time taken : ",datetime.now() - start)

print(datetime.now() - start)
```

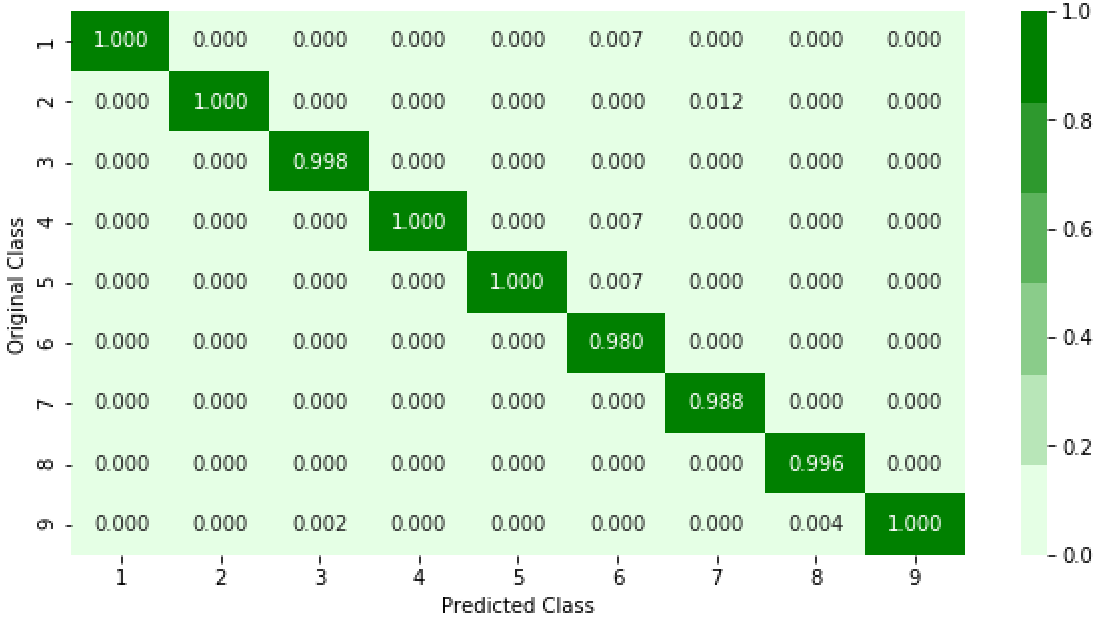
```
100%|██████████| 5/5 [06:09<00:00, 73.89s/it]

log_loss for depth =  50 is 0.04668351592283025
log_loss for depth =  100 is 0.03969018293927871
log_loss for depth =  250 is 0.036482488257369126
log_loss for depth =  500 is 0.03549085349504762
log_loss for depth =  1000 is 0.03527185576754028
0:06:09.462986
For values of best estimators =  1000 The train log loss is: 0.0012958357745138439
For values of best estimators =  1000 The cross validation log loss is: 0.03527185576754028
For values of best estimators =  1000 The test log loss is: 0.015125633915429895
Time taken :  0:08:46.736713
0:08:46.736853
```

In [47]:
```
%matplotlib inline
plot_confusion_matrix(y_test, r_cfl.predict(X_test))
```

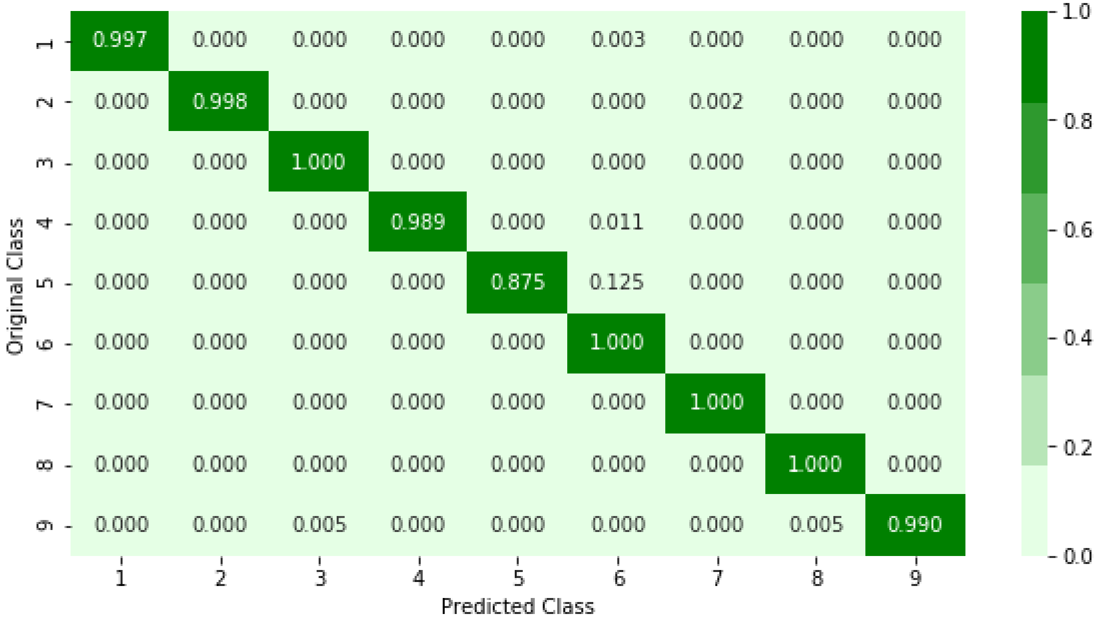
```
Number of misclassified points  0.27598896044158233
-------------------------------------------------- Confusion matrix -----------------------------------------
---------
```




```
-------------------------------------------------- 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 [48]:
```
train_logloss_xgb = log_loss(y_train, sig_clf.predict_proba(X_train))
```

In [55]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Train loss", "Test loss"]
x.add_row(["LGBM", np.round(train_logloss_LGBM,4), np.round(test_logloss_LGBM, 4)])
x.add_row(["XGB",  np.round(train_logloss_xgb,4),  np.round(test_logloss_xgb, 4)])
print(x)
```

```
+-------+------------+-----------+
| Model | Train loss | Test loss |
+-------+------------+-----------+
|  LGBM |   0.0013   |   0.0104  |
|  XGB  |   0.0013   |   0.0151  |
+-------+------------+-----------+
```

In [56]:
```
%autosave 600
```

```
Autosaving every 600 seconds
```

Conclusion

The most important learning from Malware Classification is how to deal with such large dataset and manage the long processing.

1. <u>It made me realise the importance of pickling the file :</u>
When working with large dataset there's often chances that notebok can crash and infact it did so many times, For calculating bigrams it took more than 24 hrs and it was killing the compute engine as i made it preamtible. But then I realised how big the task is.

2. <u>Doing pickling in accurate way :</u>
Earlier after calculating everything I created a dataframe by converting my bigrams feature into a dataframe that means bu using .toarray() function which inspite of doing good made a problem by converting sparse matrix back to non sparse form. How it effected? Well When i stored the the dataframe it took 3GB space thus making computation time huge. But then after team suggestion I stored in form of sparse matrix and to

my suprise it took 1GB of space thats like 1/3 of original. So Obviously it will take less space in RAM and will decrease the computation time.

3. This case study taught me patience and experimentaion aspect of the Data Science Field, on how converting a file into Image can yield such results.

4. Refernces : github.com/dchad

In [ ]:

In [ ]: