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:nothing, gs:nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24    08                           lea     eax, [e
sp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                               mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                        call    ??0e
xception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08    BB 42 00                   mov     dword
ptr [esi],    offset off_42BB08
.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_42BB08
.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_42BB08
.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_40
104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E    00 00                        call    ??3@
YAXPAX@Z    ; operator delete(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 classif
y a given a data point => Multi 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 Learning Objectives and Constraints

2.2.3. Machine Learning 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."

3. Exploratory Data Analysis

In [7]:

```
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
import scipy
from tqdm import tqdm
from joblib import dump,load
import pickle
from sklearn.feature_extraction.text import CountVectorizer
import array
import imageio
from lightgbm import LGBMClassifier
```

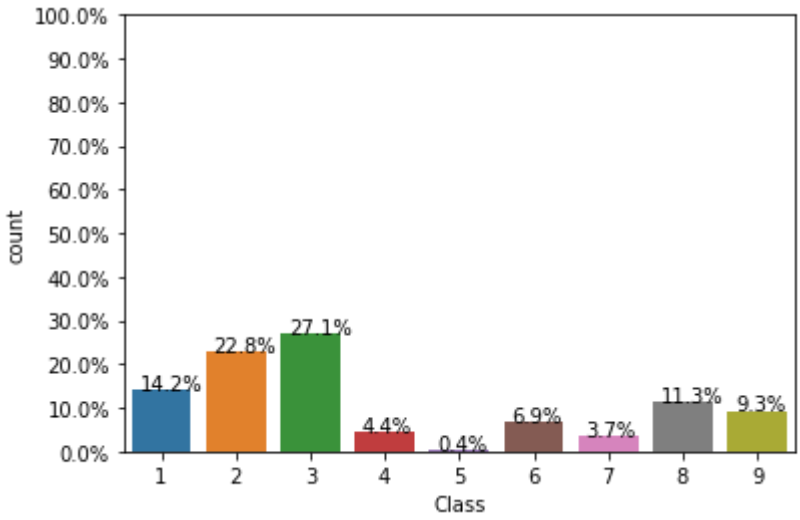
In [3]:

```
#separating byte files and asm files

source = 'assignment'
destination_1 = 'byteFiles'
destination_2 = 'asmFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a f
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

# if we have folder called 'train' (train folder contains both .asm files and .byte
# for every file that we have in our 'asmFiles' directory we check if it is ending
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    data_files = os.listdir(source)
    print(data_files[1:10])
    for file in data_files:
        #print(file)
        if (file.endswith("bytes")):
            shutil.move(source+'/'+file,destination_1)
        if (file.endswith("asm")):
            shutil.move(source+"/"+file,destination_2)
```

3.1. Distribution of malware classes in whole data set

In [4]:

```
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.g

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

#adjust the ticklabel to the desired format, without changing the position of the t
ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
plt.show()
```



In [5]:

```
Y.shape
```

Out[5]:

```
(10868, 2)
```

3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [5]:

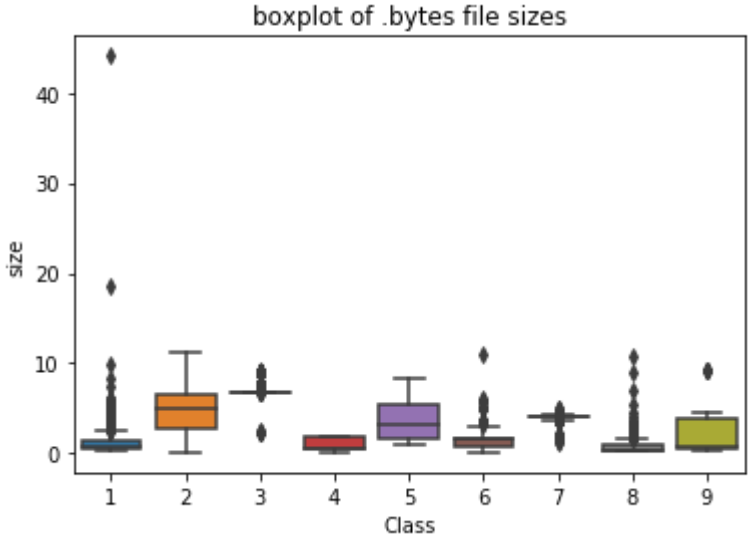
```
#file sizes of byte files

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_
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151963852
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.h
    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())
```

```
                     ID      size  Class
0  dHzVRuiXwMgU6xNIAaFr  1.675781      6
1  dJ1cMEjoYRts87lyD5PC  8.941406      3
2  0EAdHtLDypMcwjTFJziC  6.585938      2
3  g4Ls62ly8VRruEBZID9v  3.449043      2
4  bwRF4icVxTHzGnUKvu2s  3.808594      9
```

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

In [41]:

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



3.2.3 feature extraction from byte files

In [28]:

```
#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 tqdm(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()
```

In [98]:

```
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,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15
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()
```

Out[98]:

```
'\nfiles = os.listdir(\'byteFiles\')\nfilenames2=[]\nfeature_matrix
= np.zeros((len(files),257),dtype=int)\nk=0\n\n\n#program to convert
into bag of words of bytefiles\n#this is custom-built bag of words t
his is unigram bag of words\nbyte_feature_file=open(\'result.csv
\',\'w+\')\nbyte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0
c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,
23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,3
9,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,
50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,6
6,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,
7d,7e,7f,80,81,82,83,84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,9
3,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,
aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c
0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,
d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,e
d,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")\nbyte_f
eature_file.write("\n")\nfor file in files:\n    filenames2.append(f
ile)\n    byte_feature_file.write(file+",")\n    if(file.endswith("t
xt")):\n        with open(\'byteFiles/\'+file,"r") as byte_flie:\n
for lines in byte_flie:\n                line=lines.rstrip().split("
")\n                for hex_code in line:\n                    if he
```

```
x_code==\'??\':\n                    feature_matrix[k][256]+=1\n
else:\n                    feature_matrix[k][int(hex_code,16)]+=
1\n        byte_flie.close()\n    for i, row in enumerate(feature_ma
trix[k]):\n        if i!=len(feature_matrix[k])-1:\n            byte
_feature_file.write(str(row)+",")\n        else:\n            byte_f
eature_file.write(str(row))\n    byte_feature_file.write("\n")\n
\n    k += 1\n\nbyte_feature_file.close()'
```

In [42]:

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

Out[42]:

	ID	0	1	2	3	4	5	6	7	8	...	f7
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451

2 rows × 258 columns

In [43]:

```
data_size_byte.head(2)
```

Out[43]:

	ID	size	Class
0	dHzVRuiXwMgU6xNIAaFr	1.675781	6
1	dJ1cMEjoYRts87lyD5PC	8.941406	3

In [44]:

```
byte_features_with_size = byte_features.merge(data_size_byte, on='ID')
byte_features_with_size.head(2)
```

Out[44]:

	ID	0	1	2	3	4	5	6	7	8	...	f9
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439

2 rows × 260 columns

In [3]:

```
# 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 - m
    return result1
```

In [45]:

```
result = normalize(byte_features_with_size)
data_y = result['Class']
result.head(2)
```

Out[45]:

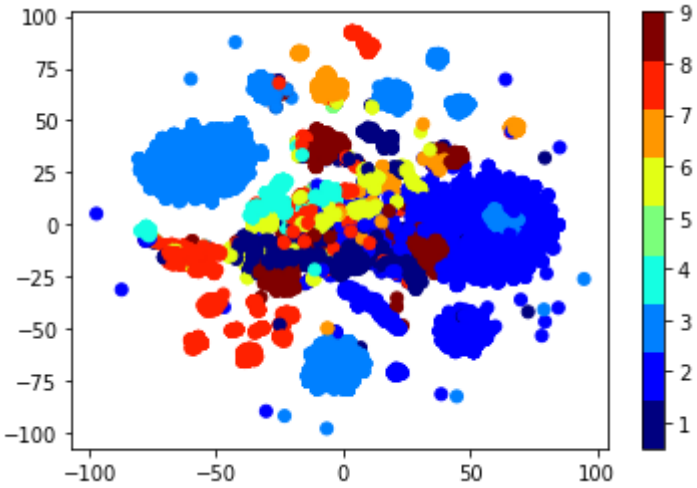
	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.00205
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.00474

2 rows × 260 columns

3.2.4 Multivariate Analysis

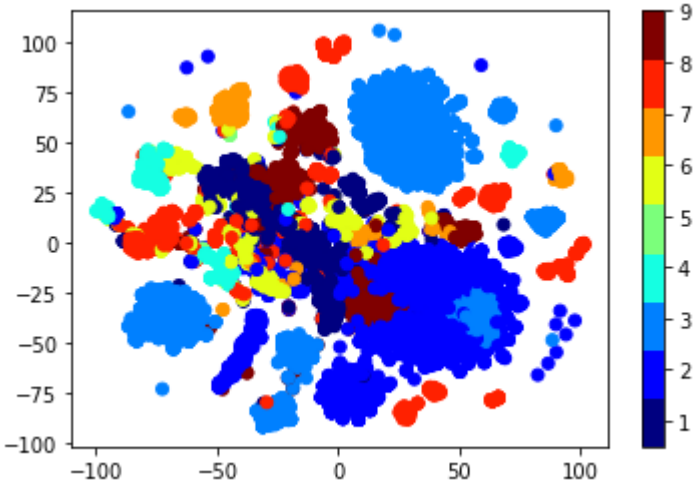
In [17]:

```
#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()
```



In [18]:

```
#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()
```



Train Test split

In [12]:

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output var
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axi
# split the train data into train and cross validation by maintaining same distribu
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,t
```

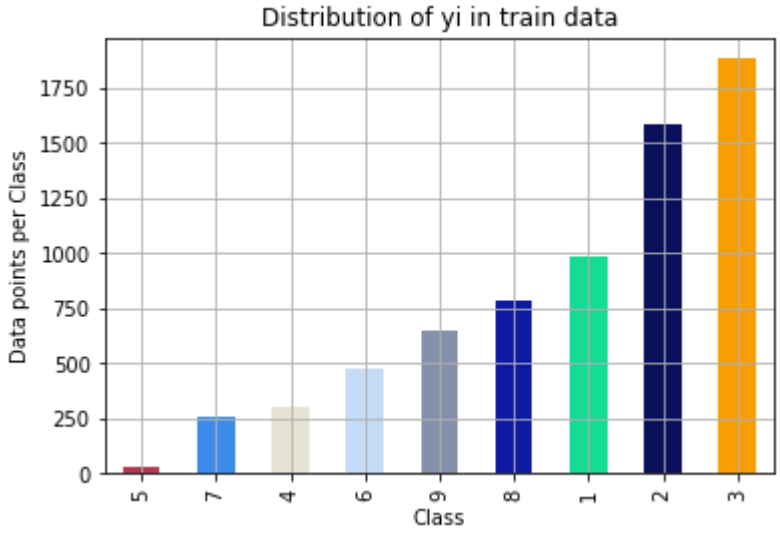
In [13]:

```
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])
```

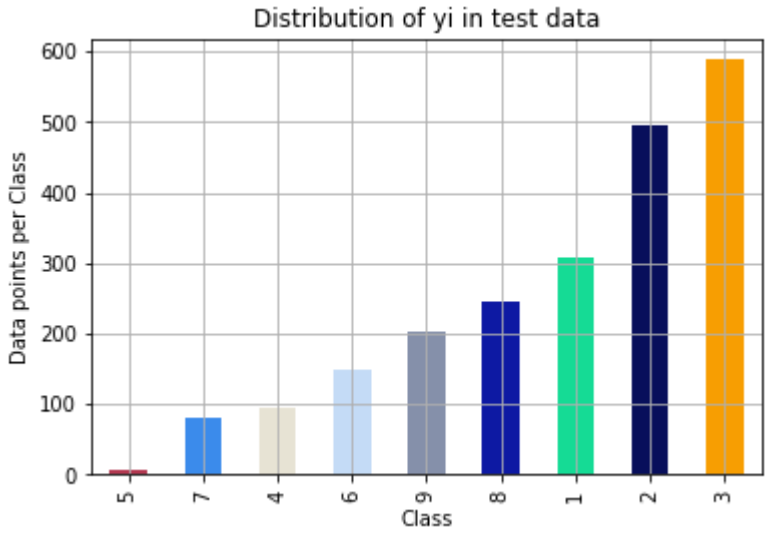
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [22]:

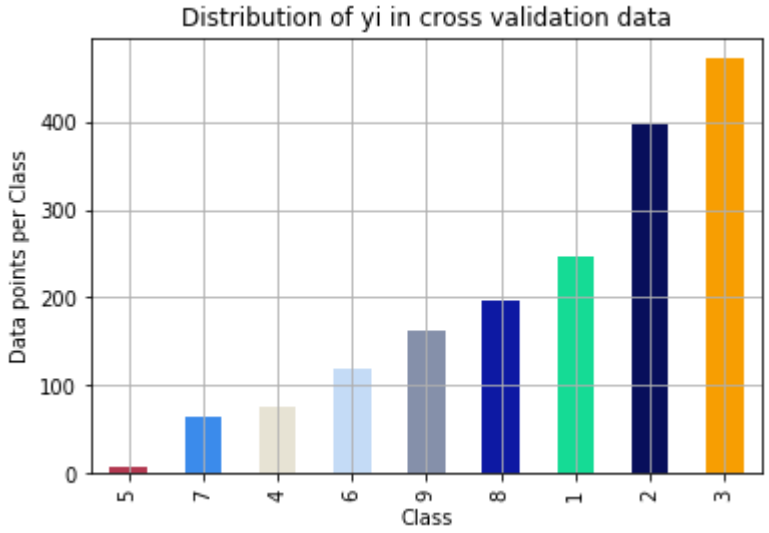
```
# it returns a dict, keys as class labels and values as the number of data points i
train_class_distribution = y_train.value_counts().sort_values()
test_class_distribution = y_test.value_counts().sort_values()
cv_class_distribution = y_cv.value_counts().sort_values()

my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15
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.h
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
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.value


print('-'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15
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.h
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
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

print('-'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15
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.h
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
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
```




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




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




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

In [14]:

```
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)*1
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pr

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

    # 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
    # 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 ro
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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, yticklabel
    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, yticklabel
    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, yticklabel
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in recall matrix",A.sum(axis=1))
```

4. Machine Learning Models

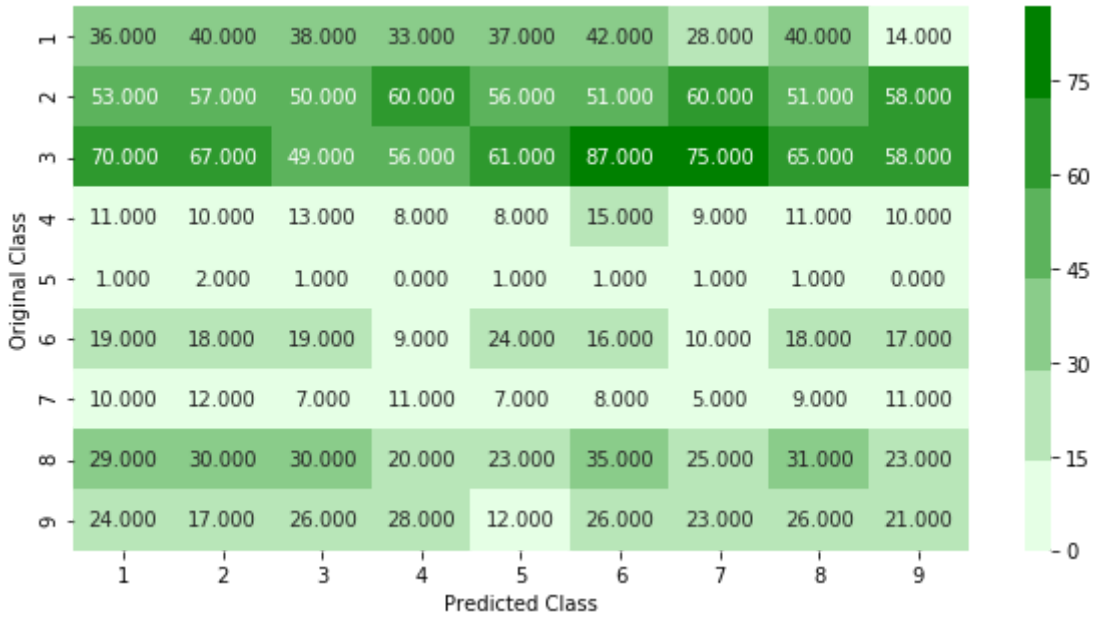
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

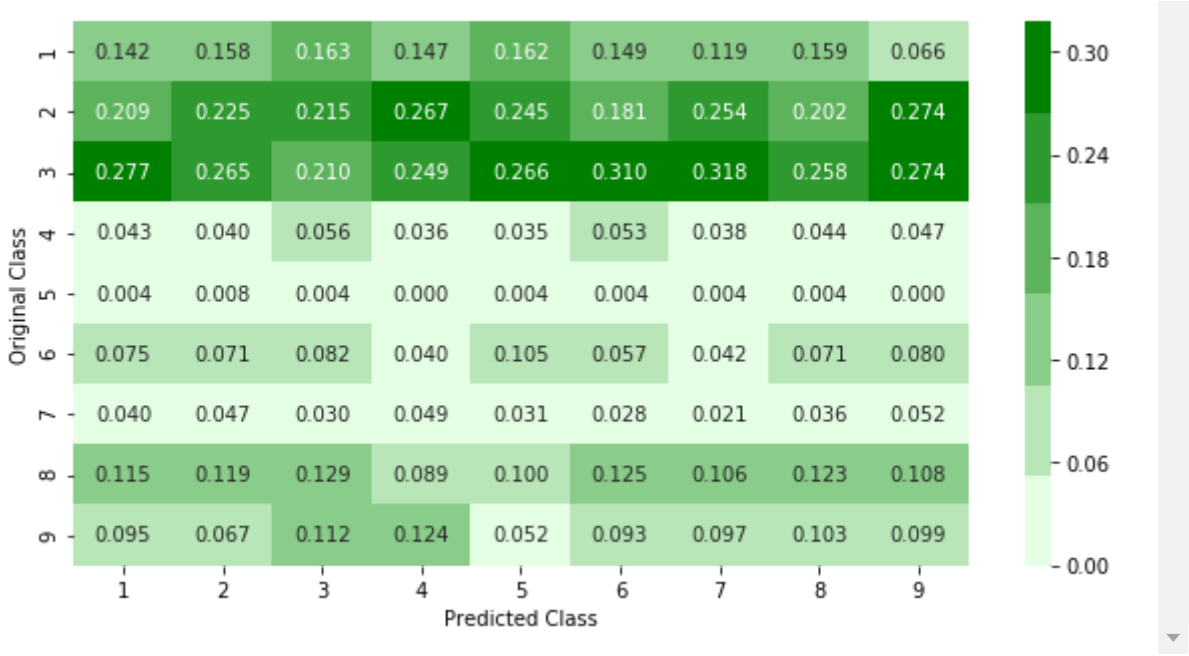
In [24]:

```
# 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_predi


# 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,

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

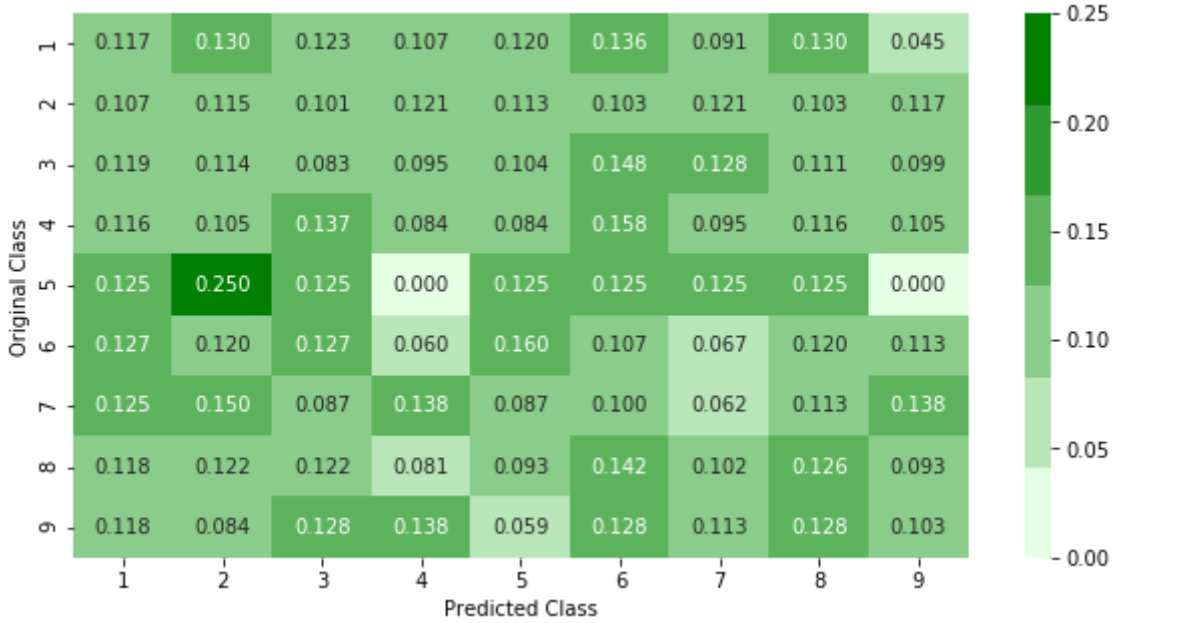
```
Log loss on Cross Validation Data using Random Model 2.474065213206862
3
Log loss on Test Data using Random Model 2.4898150115295388
Number of misclassified points  89.69641214351427
---------------------------------------------------- Confusion matrix --
-----------------------------------------------------
```




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




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




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

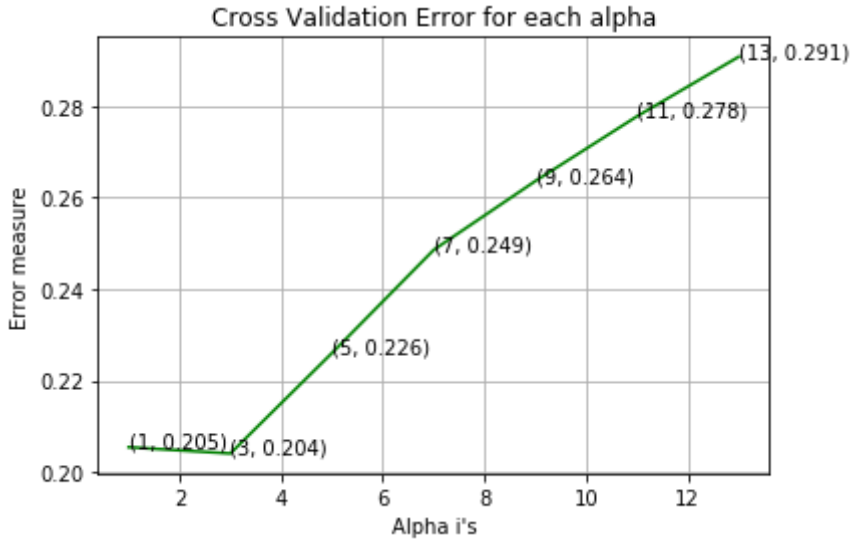
4.1.2. K Nearest Neighbour Classification

In [25]:

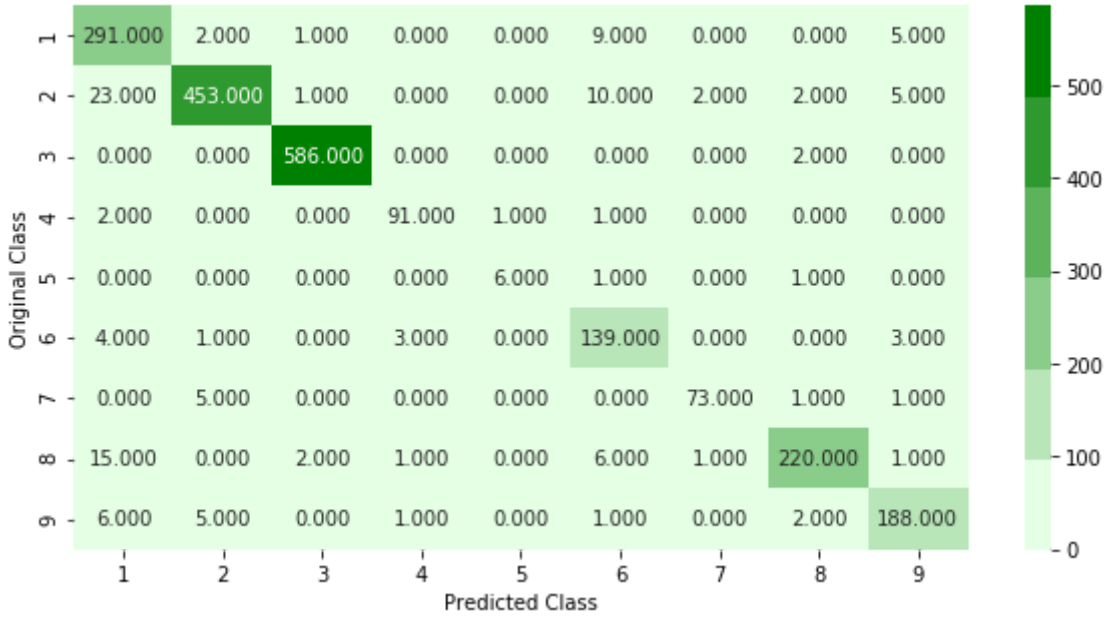
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# 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/lesso
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid',
#
# 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=

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

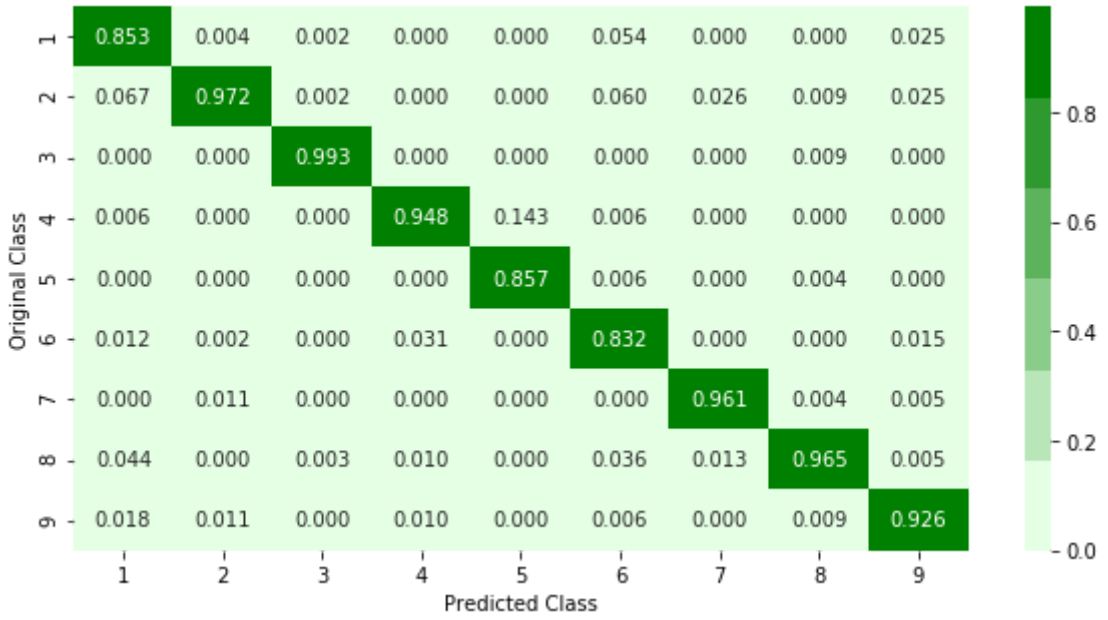
```
log_loss for k =  1 is 0.205428363261183
log_loss for k =  3 is 0.20405562528266954
log_loss for k =  5 is 0.2260168416273385
log_loss for k =  7 is 0.24855420153055857
log_loss for k =  9 is 0.2636045523619304
log_loss for k =  11 is 0.27776651594096863
log_loss for k =  13 is 0.29078377609317063
```



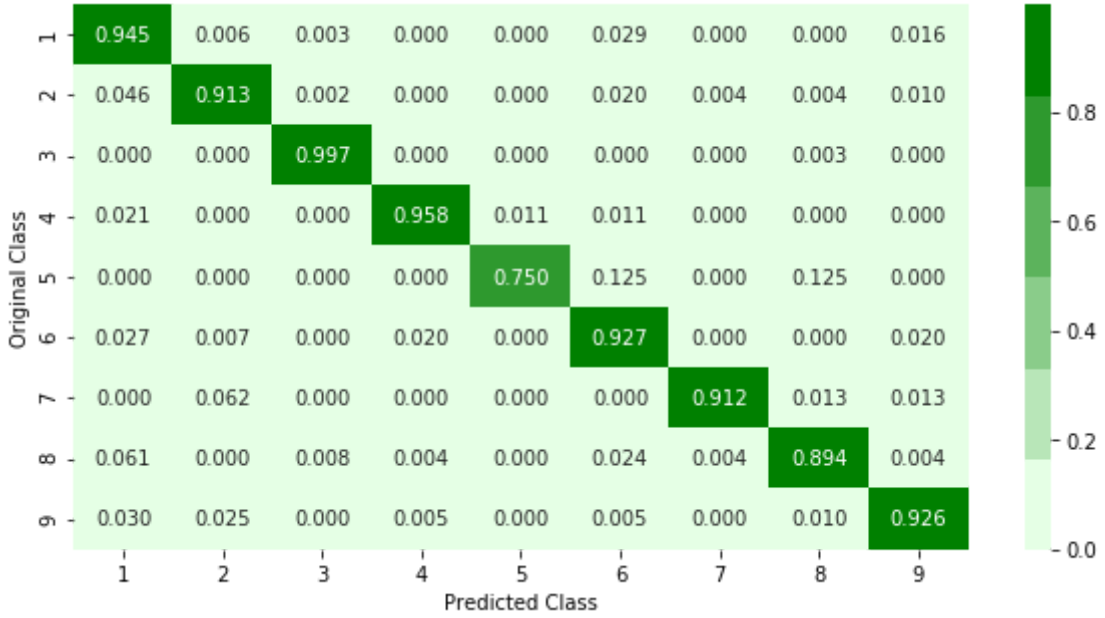

```
For values of best alpha =  3 The train log loss is: 0.125418169273613
04
For values of best alpha =  3 The cross validation log loss is: 0.2040
5562528266954
For values of best alpha =  3 The test log loss is: 0.2409284540242848
Number of misclassified points  5.841766329346826
---------------------------------------------------- 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 recall matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

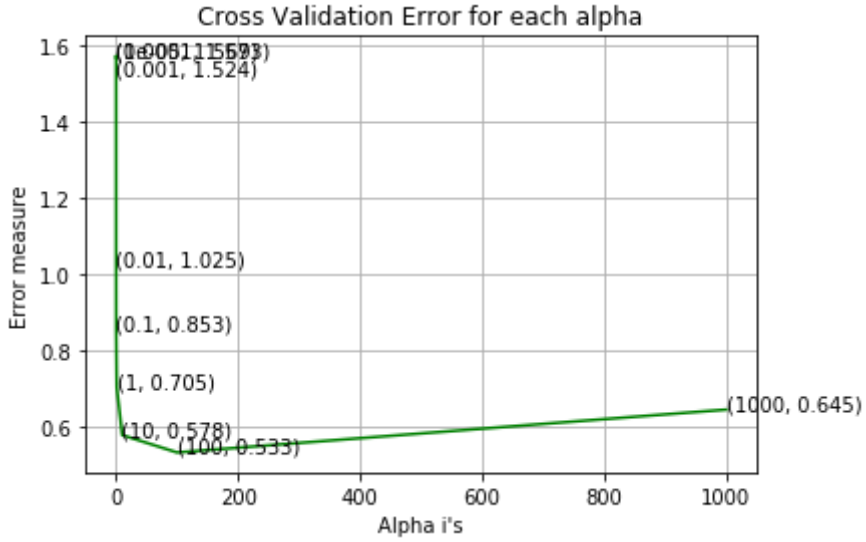
4.1.3. Logistic Regression

In [26]:

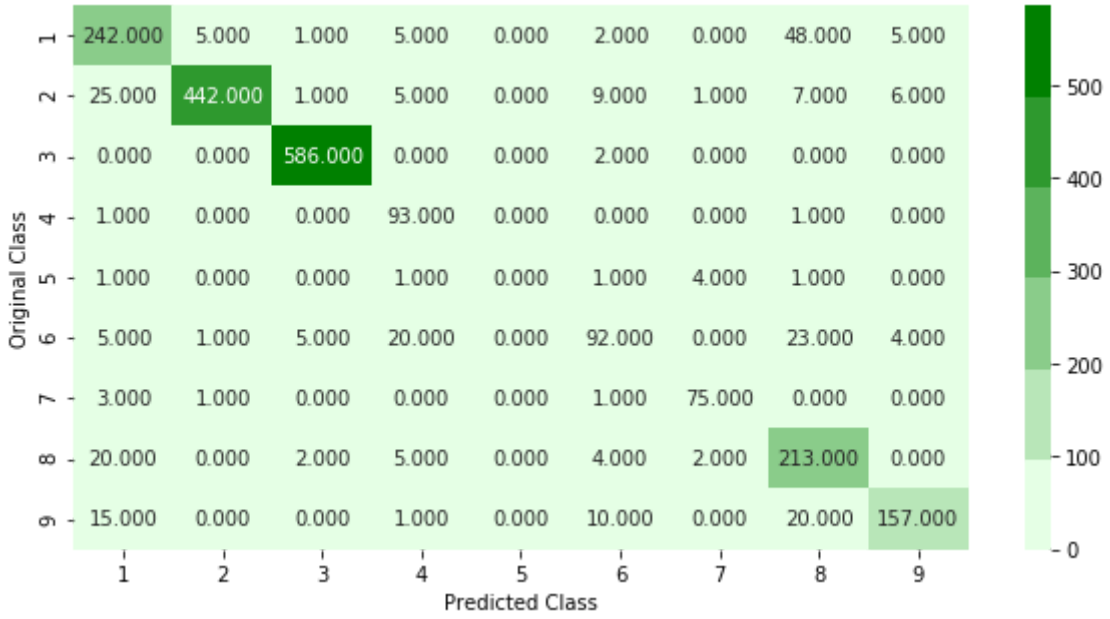
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generat
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_interc
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=
# 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 Gradie
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
#-------------------------------

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_,

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='balance
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.clas
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_,
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classe
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.5693063927886541
log_loss for c =  0.0001 is 1.5730518810253828
log_loss for c =  0.001 is 1.5238337767208936
```

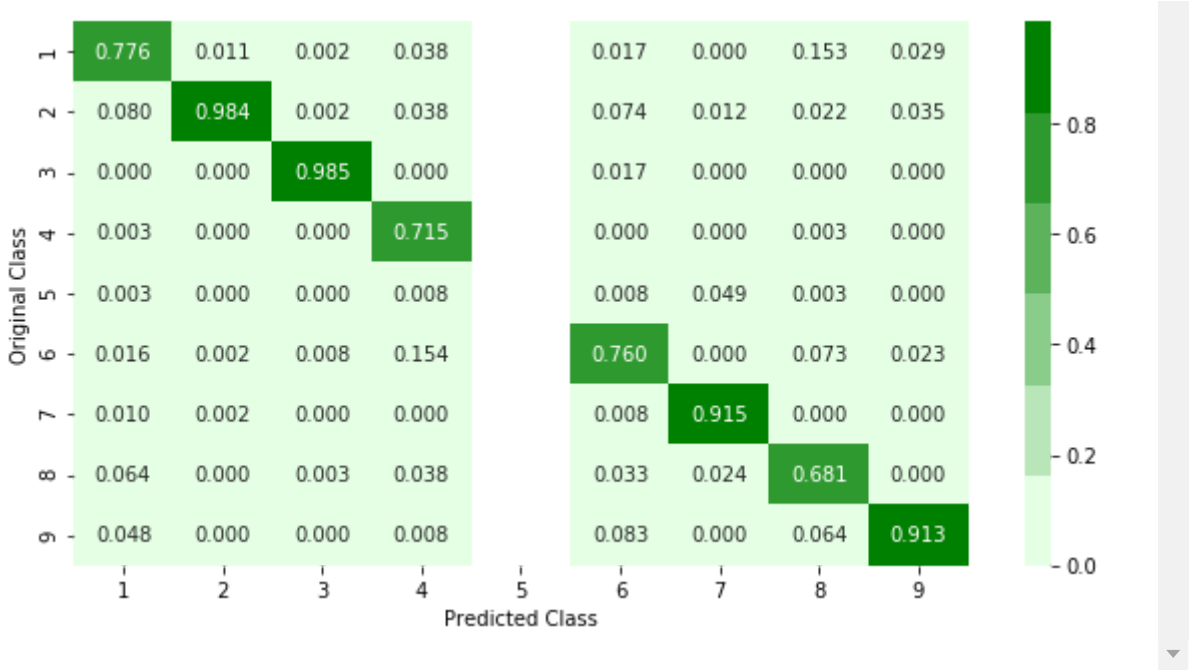
```
log_loss for c =  0.01 is 1.0252334273651493
log_loss for c =  0.1 is 0.8528754595842606
log_loss for c =  1 is 0.7046527747004966
log_loss for c =  10 is 0.5783689279117374
log_loss for c =  100 is 0.5327432675244749
log_loss for c =  1000 is 0.6451128610304003
```



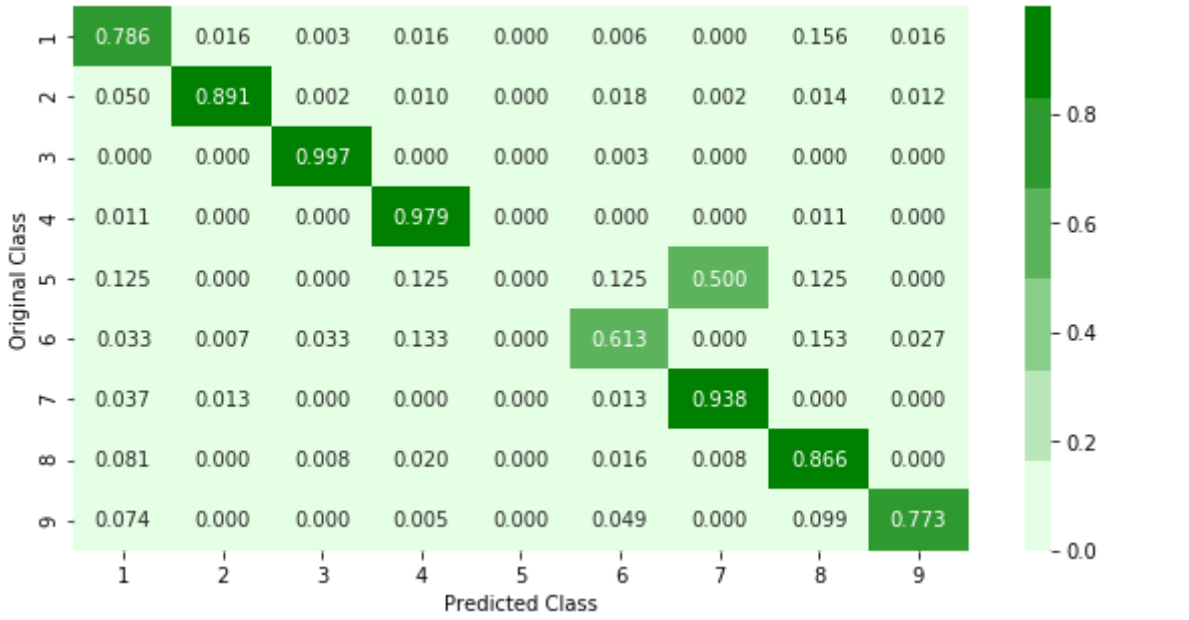

```
log loss for train data 0.49208169709572386
log loss for cv data 0.5327432675244749
log loss for test data 0.5364385204491362
Number of misclassified points  12.603495860165593
---------------------------------------------------- Confusion matrix --
-----------------------------------------------------
```




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




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




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

4.1.4. Random Forest Classifier

In [27]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_de
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_n
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training d
# 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/lesso
# ------------------------------

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,verbose=2
    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=

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=
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:",lo
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(X_test)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
building tree 1 of 10building tree 2 of 10

building tree 3 of 10
building tree 4 of 10
building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurr
ent workers.

building tree 9 of 10
building tree 10 of 10
building tree 1 of 10building tree 2 of 10

building tree 3 of 10
building tree 4 of 10
building tree 5 of 10
building tree 6 of 10
```

4.1.5. XgBoost Classification

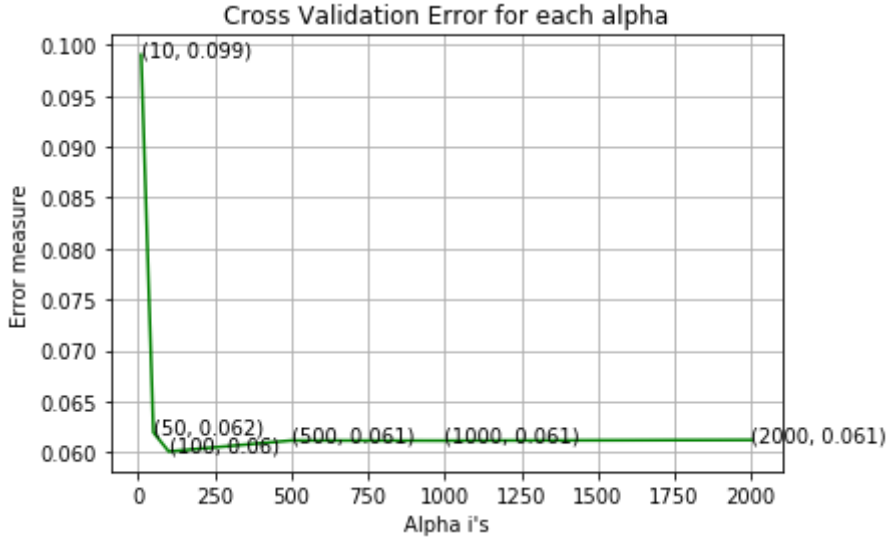
In [15]:

```
from lightgbm import LGBMClassifier
```

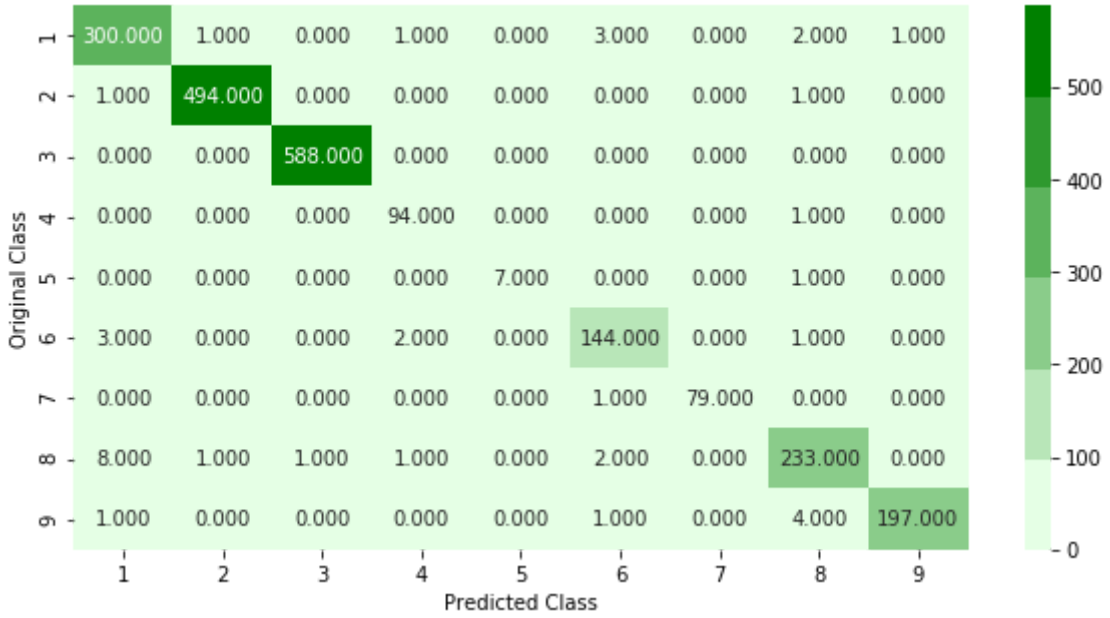
In [ ]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/late
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, sil
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, m
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kw

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rou
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# -----------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=LGBMClassifier(n_estimators=i,n_jobs=-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=

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= LGBMClassifier(n_estimators=alpha[best_alpha],n_jobs=-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:",l
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

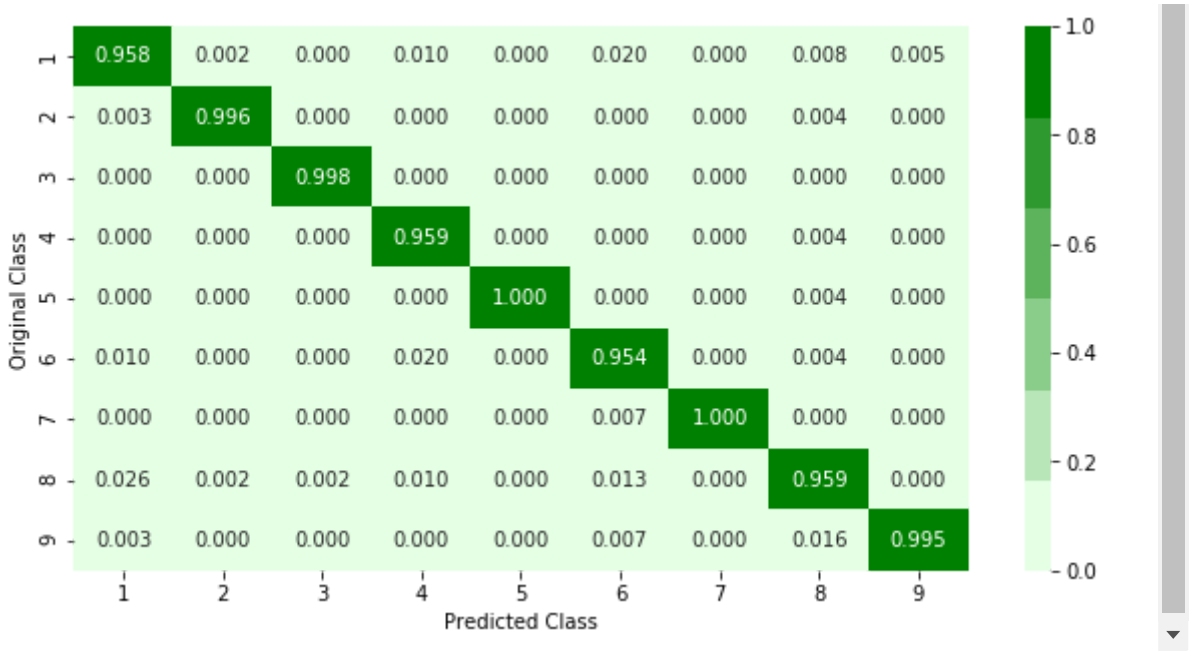
```
100%|██████████| 6/6 [06:54<00:00, 87.91s/it]

log_loss for c =  10 is 0.09904338601699424
log_loss for c =  50 is 0.061944460261453685
log_loss for c =  100 is 0.060072666500316424
log_loss for c =  500 is 0.061152025066837985
log_loss for c =  1000 is 0.061136929347111575
log_loss for c =  2000 is 0.06120613741488375
```



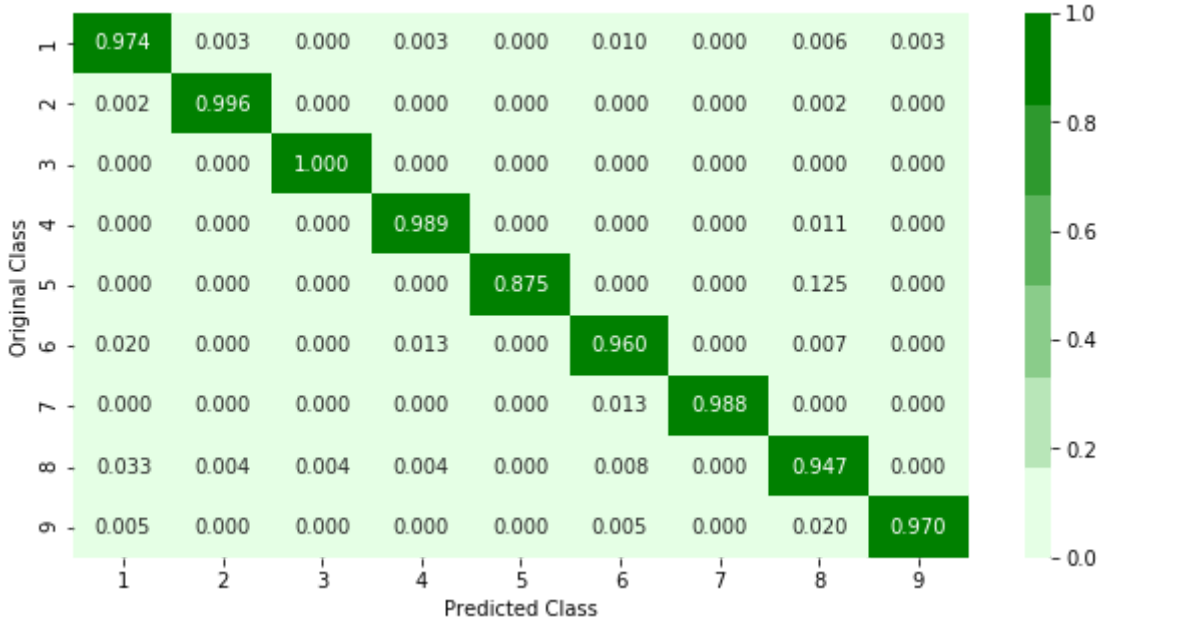

```
For values of best alpha =  100 The train log loss is: 0.0228039199261
5053
For values of best alpha =  100 The cross validation log loss is: 0.06
0072666500316424
For values of best alpha =  100 The test log log loss is: 0.09304329270495
677
Number of misclassified points  1.7479300827966882
---------------------------------------------------- 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 recall matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [ ]:

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgbo
x_cfl=LGBMClassifier()

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)]: Using backend LokyBackend with 4 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:   31.7s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:   58.6s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  5.0min remainin
g:   33.5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  9.0min finished
```

Out[30]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=LGBMClassifier(boosting_type='gbdt', class_weight=
None, colsample_bytree=1.0,
        importance_type='split', learning_rate=0.1, max_depth=-1,
        min_child_samples=20, min_child_weight=0.001, min_split_gain=
0.0,
        n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
        random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
        subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
          fit_params=None, iid='warn', 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, 200, 500, 1000, 2000], 'max_dept
h': [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='warn', scoring=None, verbose=10)
```

In [ ]:

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

```
{'subsample': 0.1, 'n_estimators': 2000, 'max_depth': 3, 'learning_rat
e': 0.03, 'colsample_bytree': 0.3}
```

In [16]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/late
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, sil
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, m
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kw

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rou
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# -----------------------

x_cfl=LGBMClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=0.3, m
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))
```

```
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the ove
rhead of testing was 0.011647 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
n: -inf
[LightGBM] [Warning] No further splits with positive gain, best gai
```

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 paral
lel 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/vid
eos/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
#third
#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=''
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]],'third')
    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:','.e
    #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', 'i
    #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:','.e
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'i
    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:','.e
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'i
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('third')
    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('third/'+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:','.e
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'i
    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:','.e
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'i
    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 [27]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the
# 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[27]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [28]:

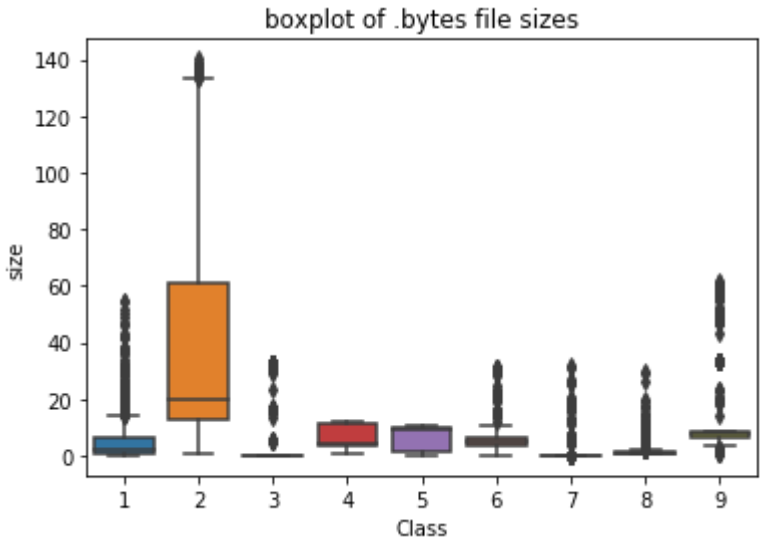
```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151963852
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.h
    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())
```

```
                     ID      size  Class
0  hSvEjCqKdFGUgplHiX3s  7.244310      1
1  gBUSDYdZqJMGCQuzWTrX  8.662652      9
2  Fxsq8u9hg0Ncl2TzLQtS  1.382896      8
3  86kXs2U1IEtu7d3fvxJY  0.176888      7
4  i4asgqmJlfer58VvktWz  0.121600      3
```

4.2.1.2 Distribution of .asm file sizes

In [29]:

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



In [30]:

```
# add the file size feature to previous extracted features
result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='ID', ho
result_asm.to_csv("result_asm_size.csv")
```

In [31]:

```
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

Out[31]:

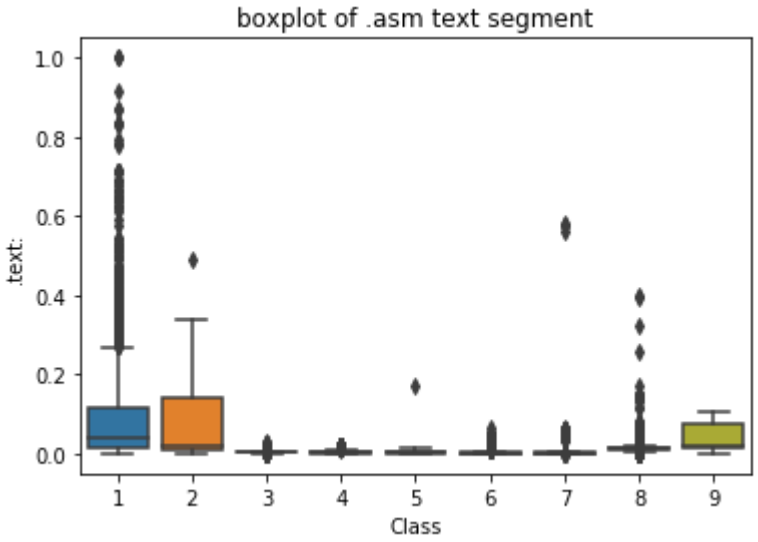
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.e
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [32]:

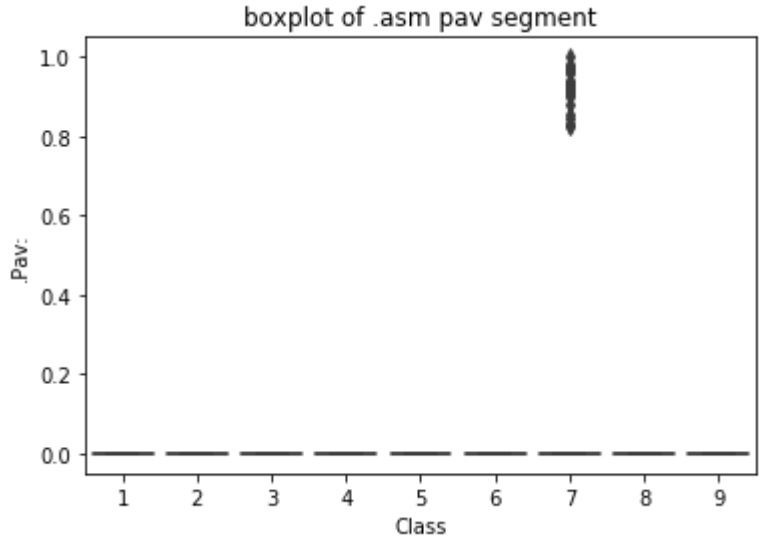
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```



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

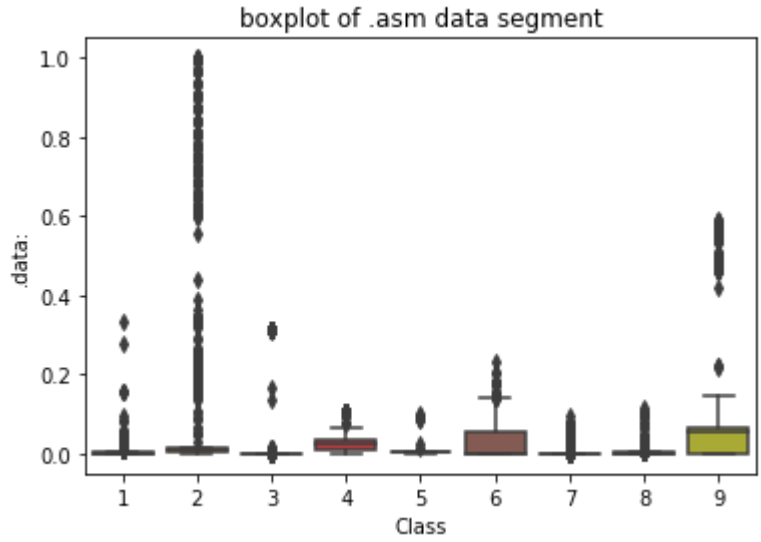
In [23]:

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



In [24]:

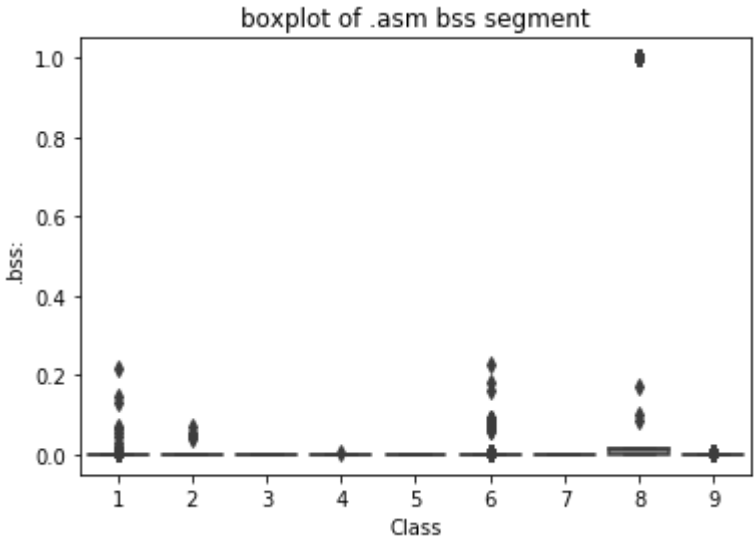
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [25]:

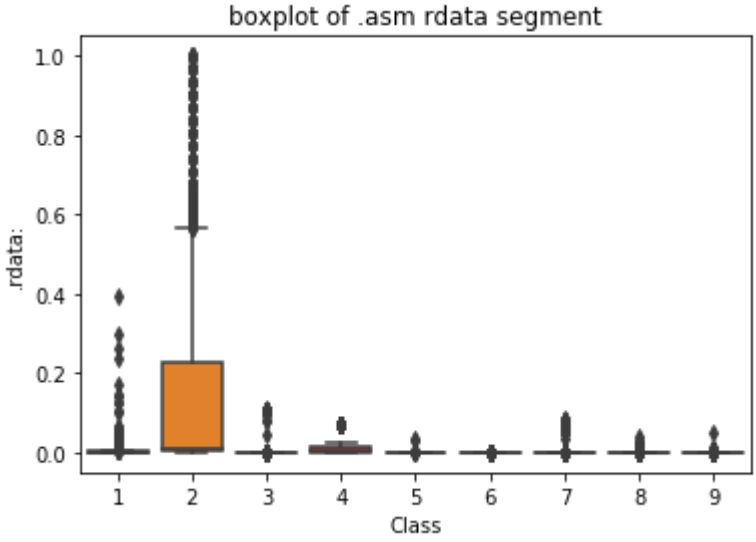
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [26]:

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

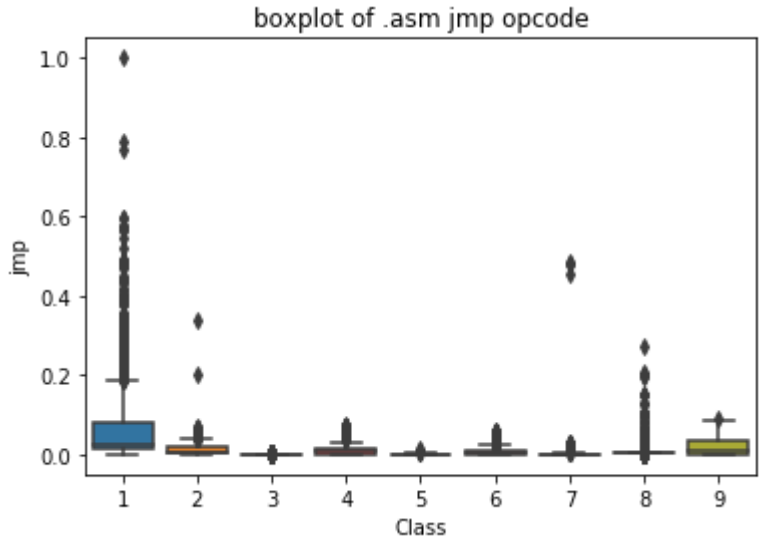


Plot between rdata segment and Class segment

Class 2 can be easily separated 75 pecentile files are having 1M rdata line
s

In [27]:

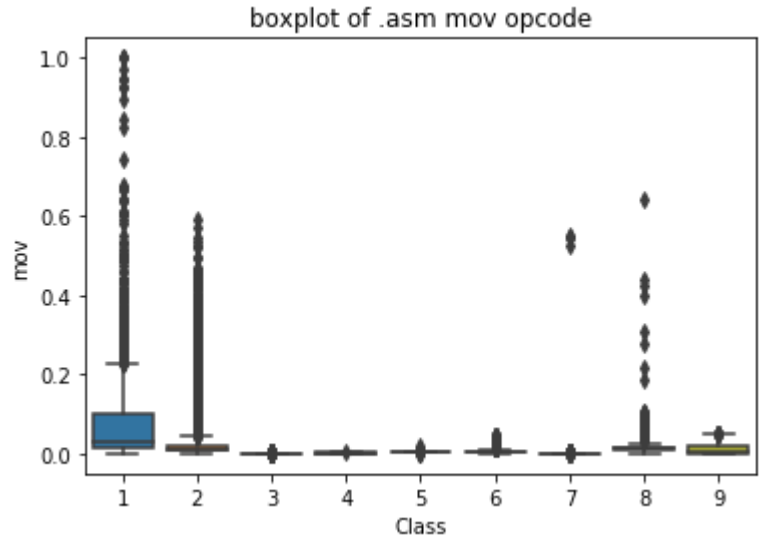
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



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

In [28]:

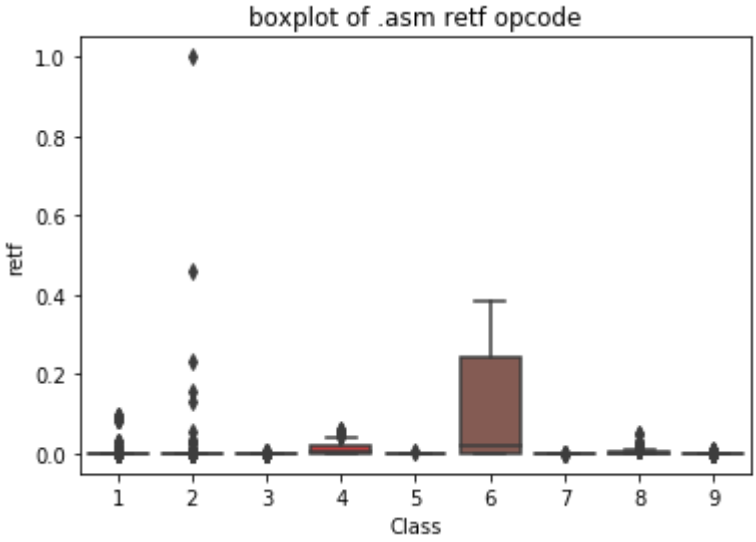
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



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

In [29]:

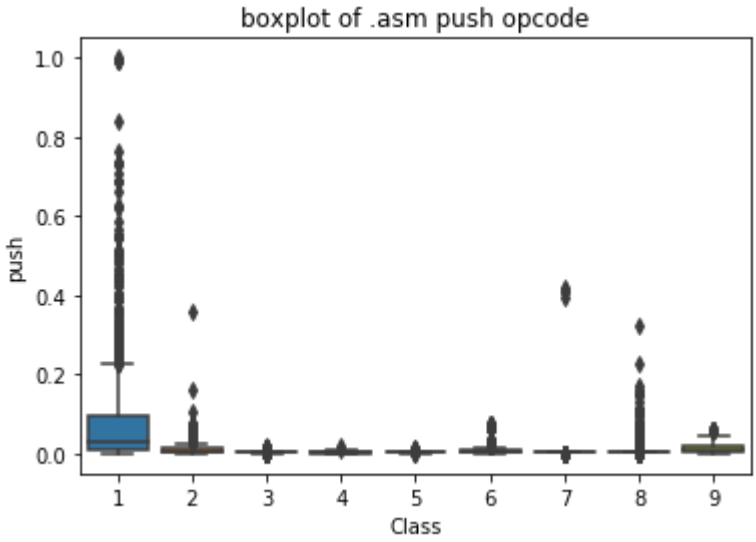
```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```



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

In [30]:

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



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

4.2.2 Multivariate Analysis on Asm file features

In [31]:

```
result.Class
```

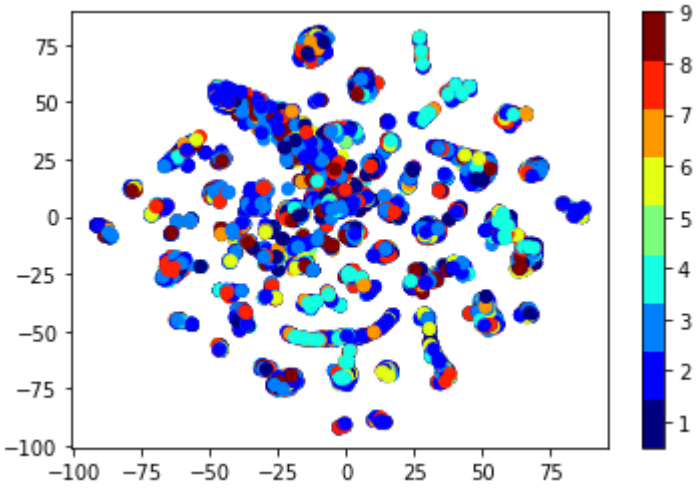
Out[31]:

```
0        9
1        2
2        9
3        1
4        8
5        6
6        2
7        2
8        6
9        2
10       2
11       6
12       3
13       3
14       1
15       8
16       9
17       3
18       3
19       2
20       1
21       2
22       2
23       6
24       1
25       2
26       2
27       3
28       3
29       3
        ..
10838    4
10839    4
10840    4
10841    4
10842    4
10843    4
10844    4
10845    4
10846    4
10847    4
10848    4
10849    4
10850    4
10851    4
10852    4
10853    4
10854    4
10855    4
10856    4
10857    4
10858    4
10859    4
10860    4
10861    4
```

```
10862    4
10863    4
10864    4
10865    4
10866    4
10867    4
Name: Class, Length: 10868, dtype: int64
```

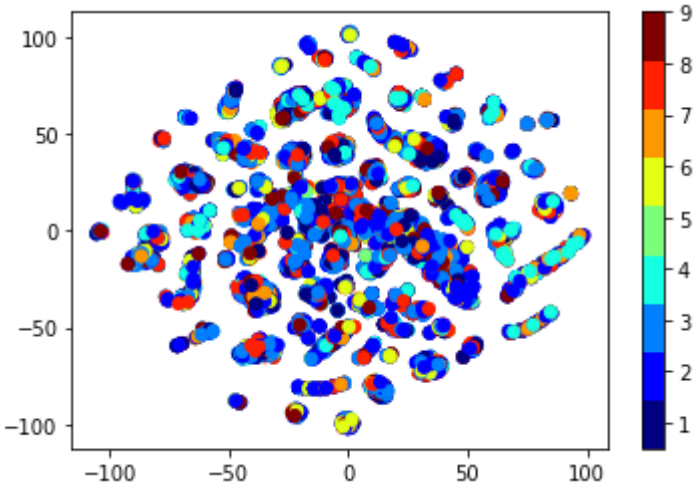
In [32]:

```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distrib

#multivariate analysis on byte files
#this is with perplexity 50
from sklearn.manifold import TSNE
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()
```



In [33]:

```
# by univariate analysis on the .asm file features we are getting very negligible i
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis aft
# the plot looks very messy

xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.CODE',
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()
```



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 [15]:

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

In [16]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,st
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_trai
```

In [36]:

```
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

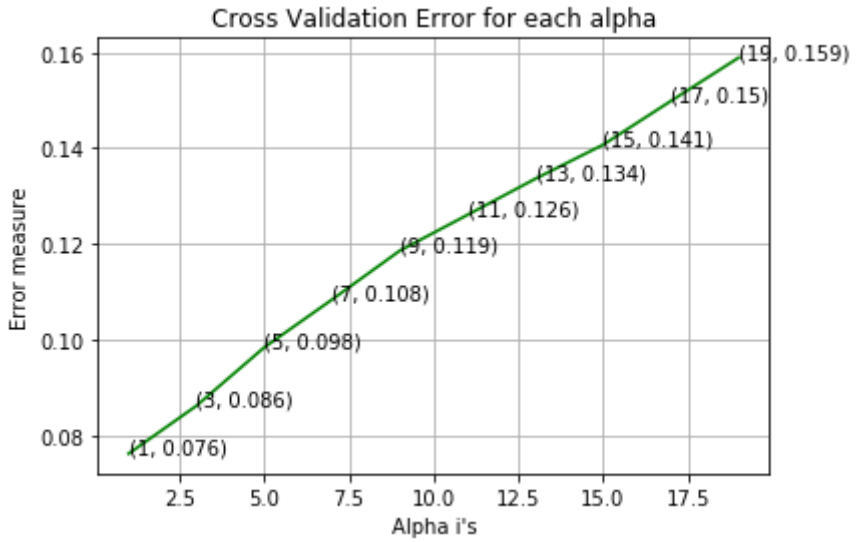
4.4.1 K-Nearest Neigbors

In [17]:

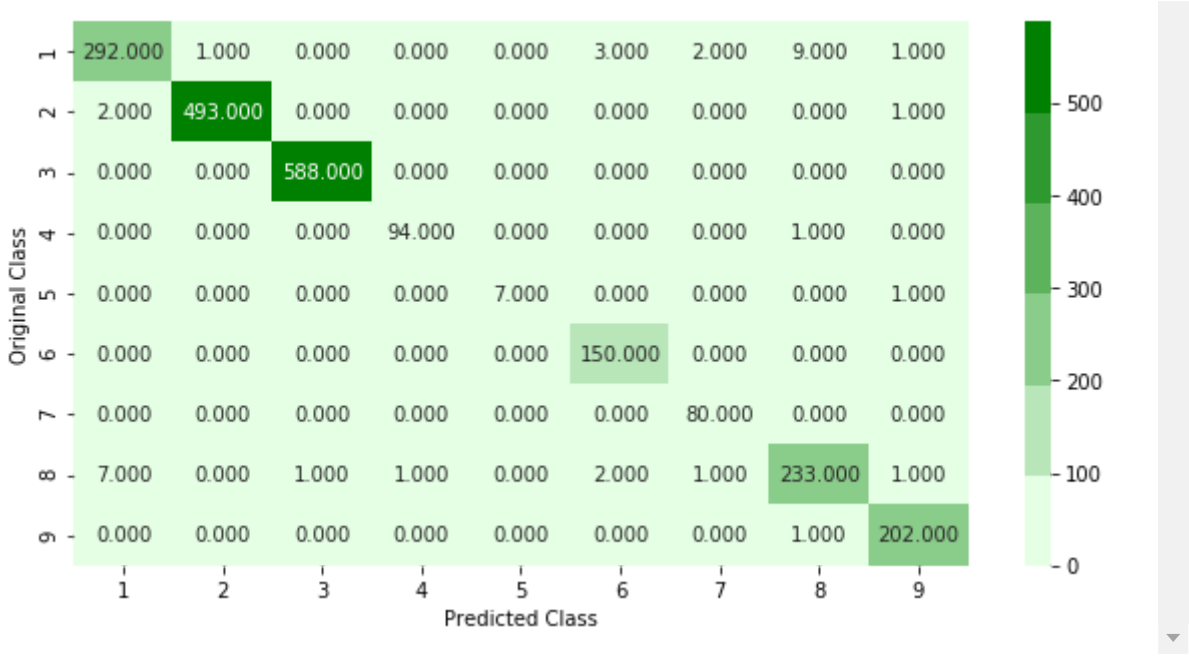
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# 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/lesso
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid',
#
# 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_,

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))
```

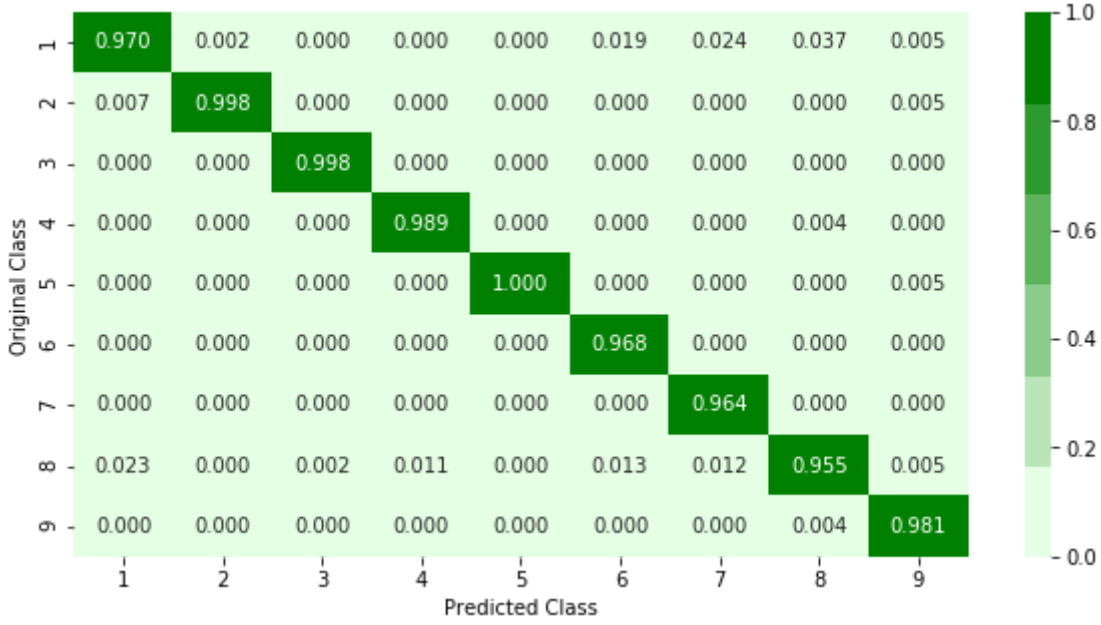
```
log_loss for k =  1 is 0.07620741135589394
log_loss for k =  3 is 0.08620371041242234
log_loss for k =  5 is 0.09839697036597038
log_loss for k =  7 is 0.10838693015526082
log_loss for k =  9 is 0.11860744752099392
log_loss for k =  11 is 0.12609794949487751
log_loss for k =  13 is 0.13363131889215418
log_loss for k =  15 is 0.14073958280016766
log_loss for k =  17 is 0.14988030849288878
log_loss for k =  19 is 0.15893430449611998
```




```
log loss for train data 0.029741804913606
log loss for cv data 0.07620741135589394
log loss for test data 0.083282587460718
Number of misclassified points  1.609935602575897
---------------------------------------------------- Confusion matrix --
-----------------------------------------------
```

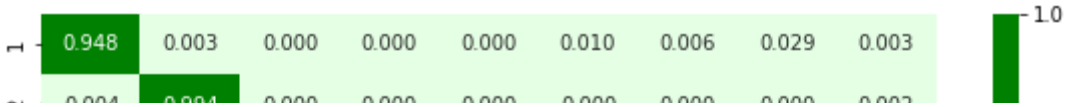


---------- Precision matrix ----------



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

---------- Recall matrix -----------




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

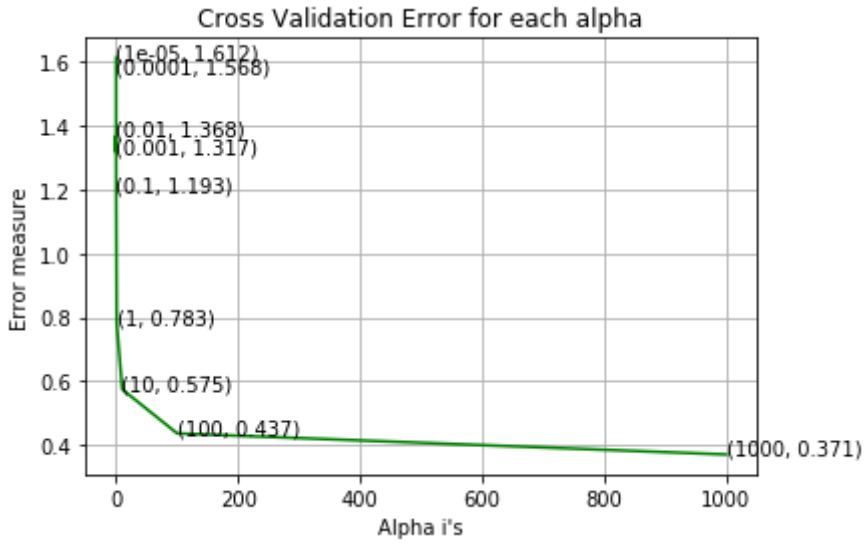
4.4.2 Logistic Regression

In [21]:

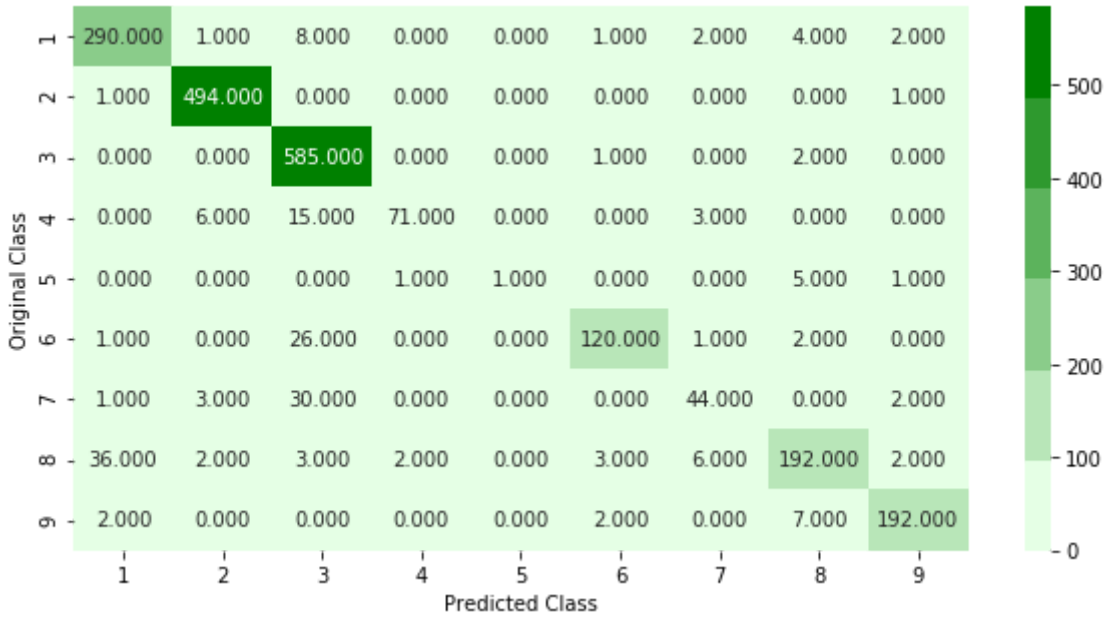
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generat
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_interc
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=
# 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 Gradie
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
#-------------------------------


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',verbose=1
    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.classe

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='balance
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
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.class
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.c
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
[LibLinear][LibLinear][LibLinear][LibLinear][LibLinear][LibLinear][L
ibLinear][LibLinear][LibLinear][LibLinear][LibLinear][LibLinear][Lib
Linear][LibLinear][LibLinear][LibLinear][LibLinear][LibLinear][LibLi
```

```
near][LibLinear][LibLinear][LibLinear][LibLinear][LibLinear][LibLine
ar][LibLinear][LibLinear][LibLinear][LibLinear][LibLinear][LibLinea
r][LibLinear][LibLinear][LibLinear][LibLinear][LibLinear]log_loss fo
r c =  1e-05 is 1.6121536139480213
log_loss for c =  0.0001 is 1.5683060179449233
log_loss for c =  0.001 is 1.3165096860524672
log_loss for c =  0.01 is 1.3682823763658911
log_loss for c =  0.1 is 1.1925915386490682
log_loss for c =  1 is 0.7825982195925653
log_loss for c =  10 is 0.5748532038142328
log_loss for c =  100 is 0.43695680580815416
log_loss for c =  1000 is 0.3710098944626412
```

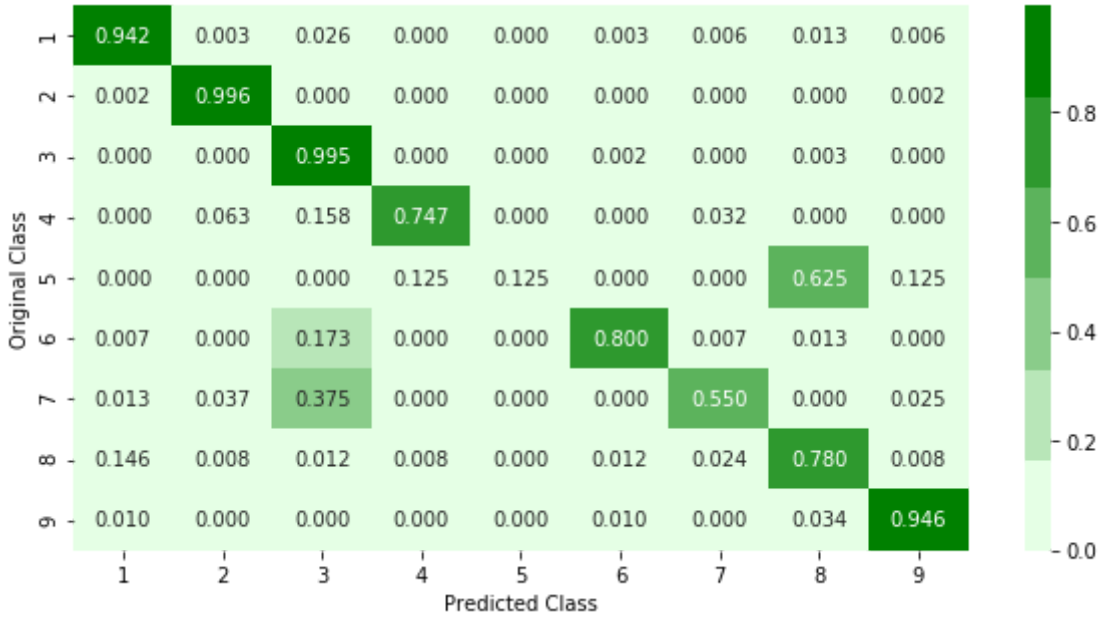



```
log loss for train data 0.351868664573977
log loss for cv data 0.3710098944626412
log loss for test data 0.3688909124059504
Number of misclassified points  8.509659613615456
---------------------------------------------------- Confusion matrix --
-----------------------------------------------------
```




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


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




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

4.4.3 Random Forest Classifier

In [41]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_de
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_n
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training d
# 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/lesso
# --------------------------------

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,verbose=3
    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_,

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=
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.c
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
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.cla
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

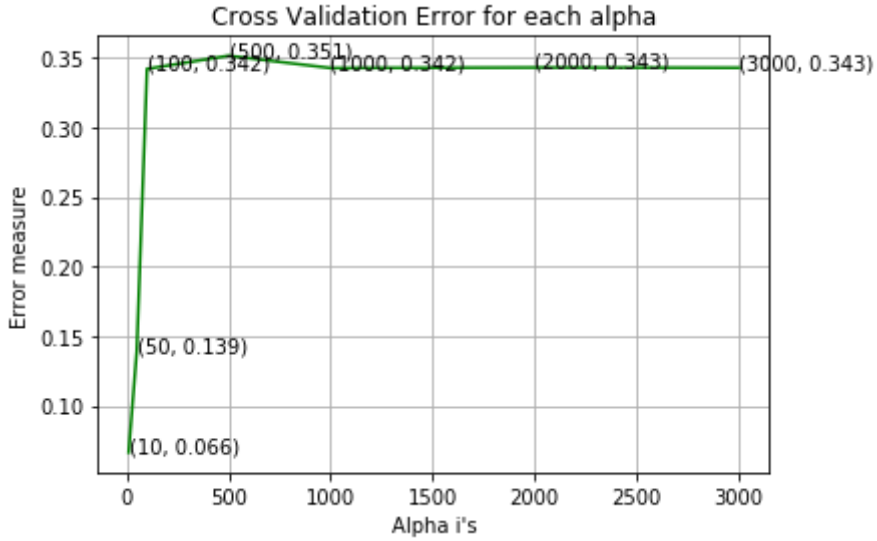
```
building tree 1 of 10
building tree 2 of 10
building tree 3 of 10building tree 4 of 10

building tree 5 of 10
building tree 6 of 10building tree 7 of 10

building tree 8 of 10
building tree 9 of 10
building tree 10 of 10
building tree 1 of 10building tree 2 of 10
building tree 3 of 10

building tree 4 of 10
building tree 5 of 10
building tree 6 of 10building tree 7 of 10

building tree 8 of 10
building tree 9 of 10
```

4.4.4 XgBoost Classifier

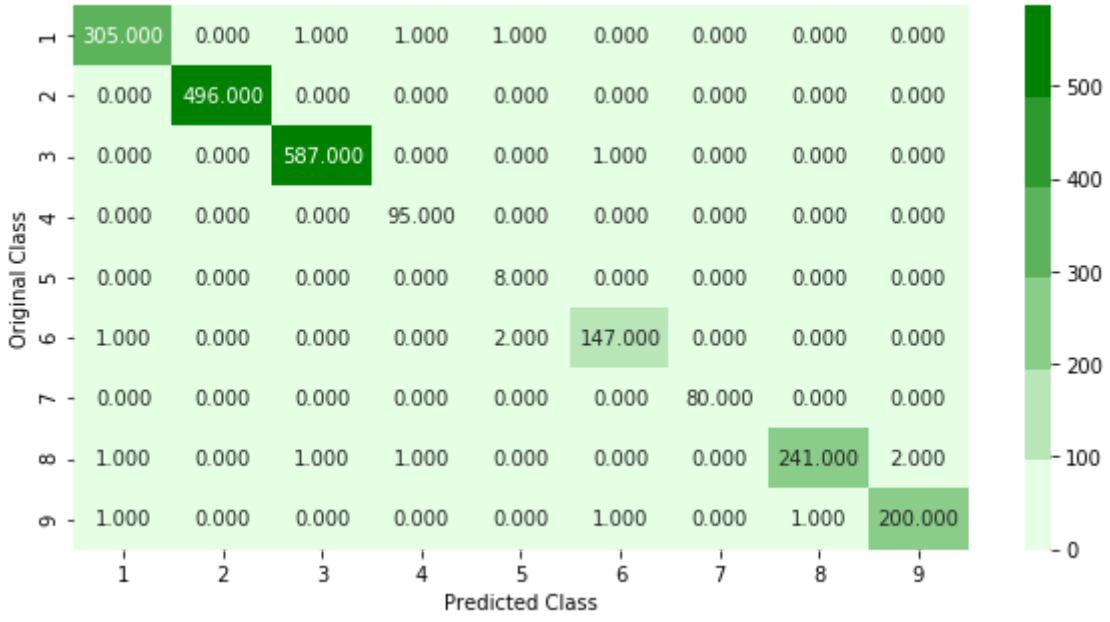
In [43]:

```
from lightgbm import LGBMClassifier
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/late
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, sil
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, m
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kw

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rou
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# -----------------------

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

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

predict_y = sig_clf.predict_proba(X_train_asm.values)

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

```
log_loss for c =  10 is 0.06637690305951909
log_loss for c =  50 is 0.13943783842366378
log_loss for c =  100 is 0.34179953870401236
log_loss for c =  500 is 0.35127330126226614
log_loss for c =  1000 is 0.34243567972588734
log_loss for c =  2000 is 0.342794862853109
log_loss for c =  3000 is 0.3426092070482434
```

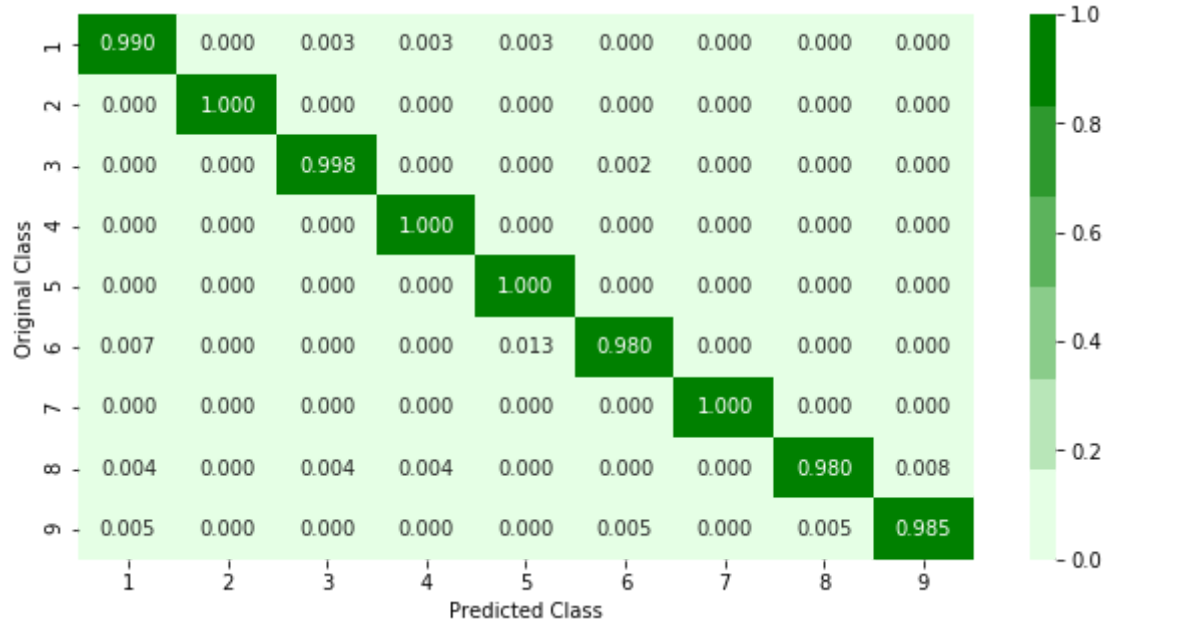



```
For values of best alpha =  10 The train log loss is: 0.03617392691636
318
For values of best alpha =  10 The cross validation log loss is: 0.066
37690305951909
For values of best alpha =  10 The test log loss is: 0.050433504163215
49
Number of misclassified points  0.6899724011039559
---------------------------------------------------- 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 recall matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.5 Xgboost Classifier with best hyperparameters

In [46]:

```
#from xgbooost import XGBClassifier
x_cfl=LGBMClassifier()

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.values,y_train_asm)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:   30.8s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:   41.2s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:   56.6s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  2.0min remainin
g:   13.1s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.2min finished
/home/mahaseth_rahul/anaconda3/lib/python3.7/site-packages/sklearn/mod
el_selection/_search.py:841: DeprecationWarning: The default of the `i
id` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set siz
es are unequal.
  DeprecationWarning)
```

Out[46]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=LGBMClassifier(boosting_type='gbdt', class_weight=
None, colsample_bytree=1.0,
        importance_type='split', learning_rate=0.1, max_depth=-1,
        min_child_samples=20, min_child_weight=0.001, min_split_gain=
0.0,
        n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
        random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
        subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
          fit_params=None, iid='warn', 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, 200, 500, 1000, 2000], 'max_dept
h': [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='warn', scoring=None, verbose=10)
```

In [47]:

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

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 5, 'learning_rat
e': 0.01, 'colsample_bytree': 0.5}
```

In [50]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/late
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, sil
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, m
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kw

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rou
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# -----------------------

x_cfl=LGBMClassifier(n_estimators=1000,subsample=1,learning_rate=0.1,colsample_bytr
x_cfl.fit(X_train_asm.values,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm.values,y_train_asm)

predict_y = c_cfl.predict_proba(X_train_asm.values)
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 0.013521233246951148
cv loss 0.042514727996085515
test loss 0.02512483668302163
```

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

4.5.1. Merging both asm and byte file features

In [46]:

```
result.head()
```

Out[46]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002(
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0047
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005(
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0003
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0001

5 rows × 260 columns

In [47]:

```
result_asm.head()
```

Out[47]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.e
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	

5 rows × 54 columns

In [48]:

```
print(result.shape)
print(result_asm.shape)
```

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

In [49]:

```
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[49]:

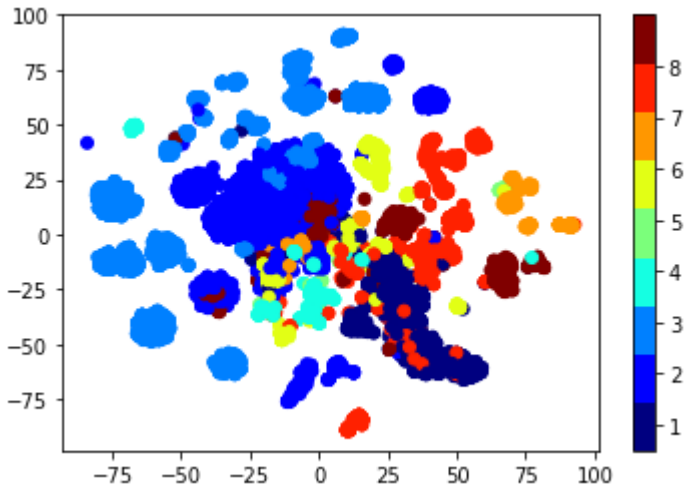
	0	1	2	3	4	5	6	7	8	
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	(
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	(
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	(
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	(
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	(

5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

In [55]:

```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x,)
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()
```



4.5.3. Train and Test split

In [64]:

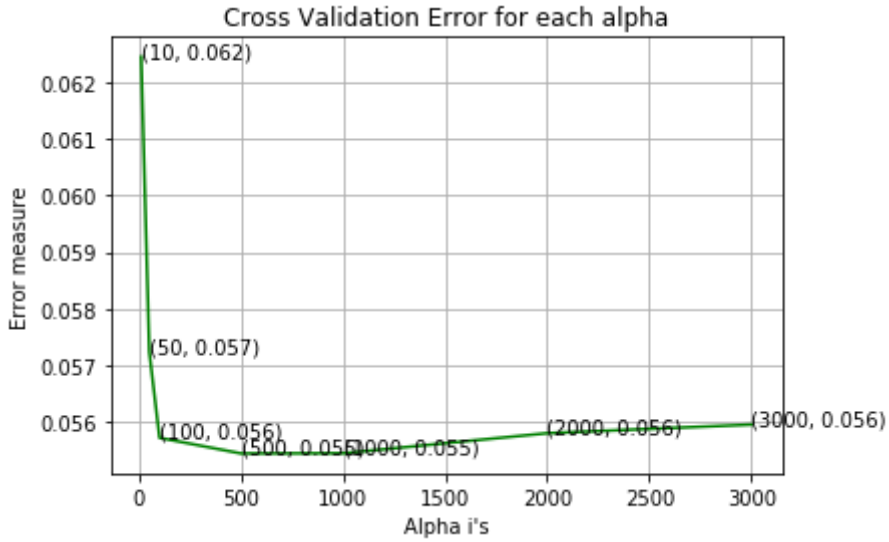
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_
```

4.5.4. Random Forest Classifier on final features

In [66]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_de
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_n
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training d
# 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/lesso
# ------------------------------

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_

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=
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:",l
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
log_loss for c =  10 is 0.06245644547521164
log_loss for c =  50 is 0.057231250044264576
log_loss for c =  100 is 0.05571376715236463
log_loss for c =  500 is 0.05543634421980848
log_loss for c =  1000 is 0.05543957607601424
log_loss for c =  2000 is 0.05579477366801665
log_loss for c =  3000 is 0.05594718422164594
```



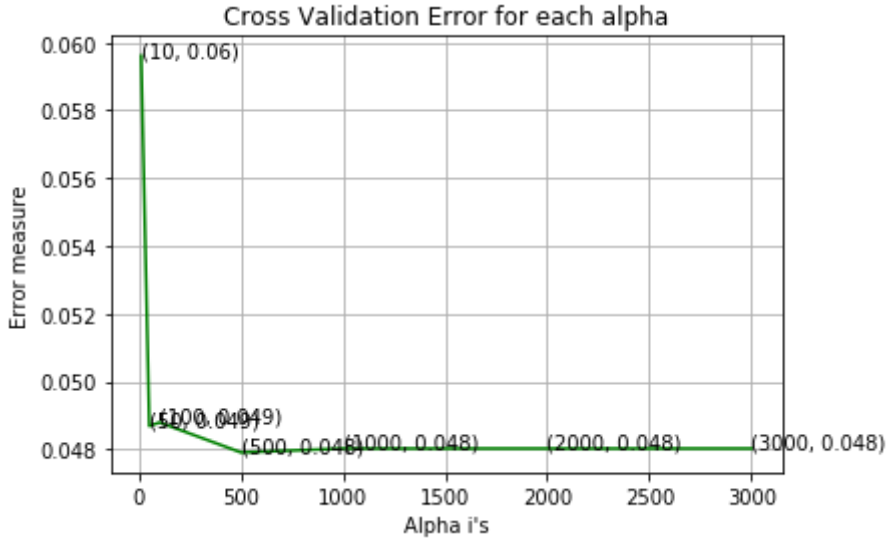

```
For values of best alpha =  500 The train log loss is: 0.0158282052202
85343
For values of best alpha =  500 The cross validation log loss is: 0.05
543634421980848
For values of best alpha =  500 The test log loss is: 0.04205544069825
785
```

4.5.5. XgBoost Classifier on final features

In [67]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/late
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, sil
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, m
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kw

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rou
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# -----------------------

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

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

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

```
log_loss for c =  10 is 0.05961034469807378
log_loss for c =  50 is 0.048687326872981884
log_loss for c =  100 is 0.04879660918266446
log_loss for c =  500 is 0.047909309516666644
log_loss for c =  1000 is 0.04802570243762141
log_loss for c =  2000 is 0.048025938408010095
log_loss for c =  3000 is 0.048025938408010095
```




```
[LightGBM] [Warning] num_threads is set with nthread=-1, will be overr
idden by n_jobs=-1. Current value: num_threads=-1
[LightGBM] [Warning] num_threads is set with nthread=-1, will be overr
idden by n_jobs=-1. Current value: num_threads=-1
[LightGBM] [Warning] num_threads is set with nthread=-1, will be overr
idden by n_jobs=-1. Current value: num_threads=-1
For values of best alpha =  500 The train log loss is: 0.0107575798371
81284
For values of best alpha =  500 The cross validation log loss is: 0.04
8025938422813906
For values of best alpha =  500 The test log loss is: 0.03305533366828
903
```

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

In [73]:

```
x_cfl=LGBMClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[1,3,5,10],
    '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.values, y_train_merge)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:   30.1s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:   54.3s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  3.2min remainin
g:   21.1s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  3.5min finished
```

Out[73]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=LGBMClassifier(boosting_type='gbdt', class_weight=
None, colsample_bytree=1.0,
        importance_type='split', learning_rate=0.1, max_depth=-1,
        min_child_samples=20, min_child_weight=0.001, min_split_gain=
0.0,
        n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
        random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
        subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
          fit_params=None, iid='warn', 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, 200, 500, 1000, 2000], 'max_dept
h': [1, 3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsampl
e': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [69]:

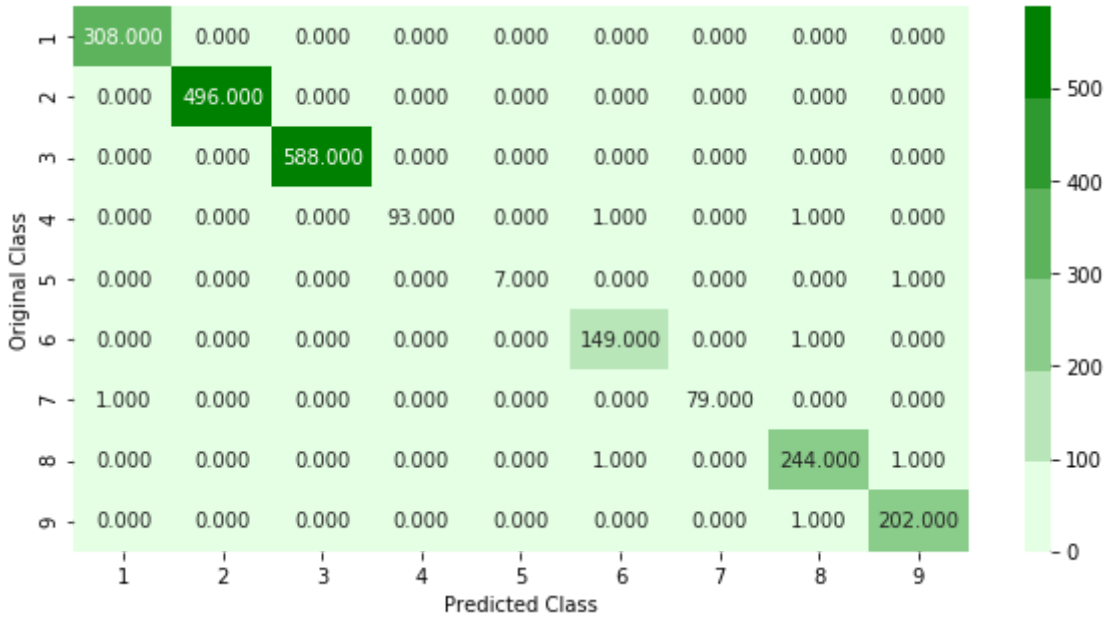
```
print (random_cfl.best_params_)
```

```
{'subsample': 0.1, 'n_estimators': 1000, 'max_depth': 3, 'learning_rat
e': 0.03, 'colsample_bytree': 0.3}
```

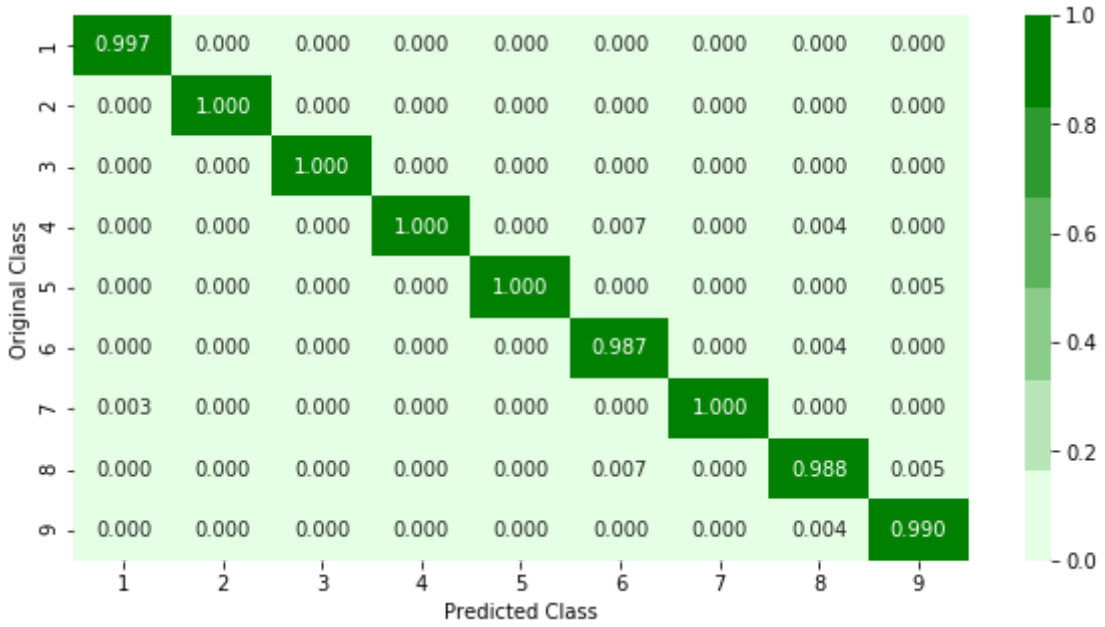
In [75]:

```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/late
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, sil
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, m
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kw

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rou
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/less
# -----------------------

x_cfl=LGBMClassifier(n_estimators=1000,max_depth=3,learning_rate=0.03,colsample_byt
x_cfl.fit(X_train_merge.values,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge.values, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge.values)
print ("The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge.values)
print( "The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge.values)
print("The test log loss is:",log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge.values))
```

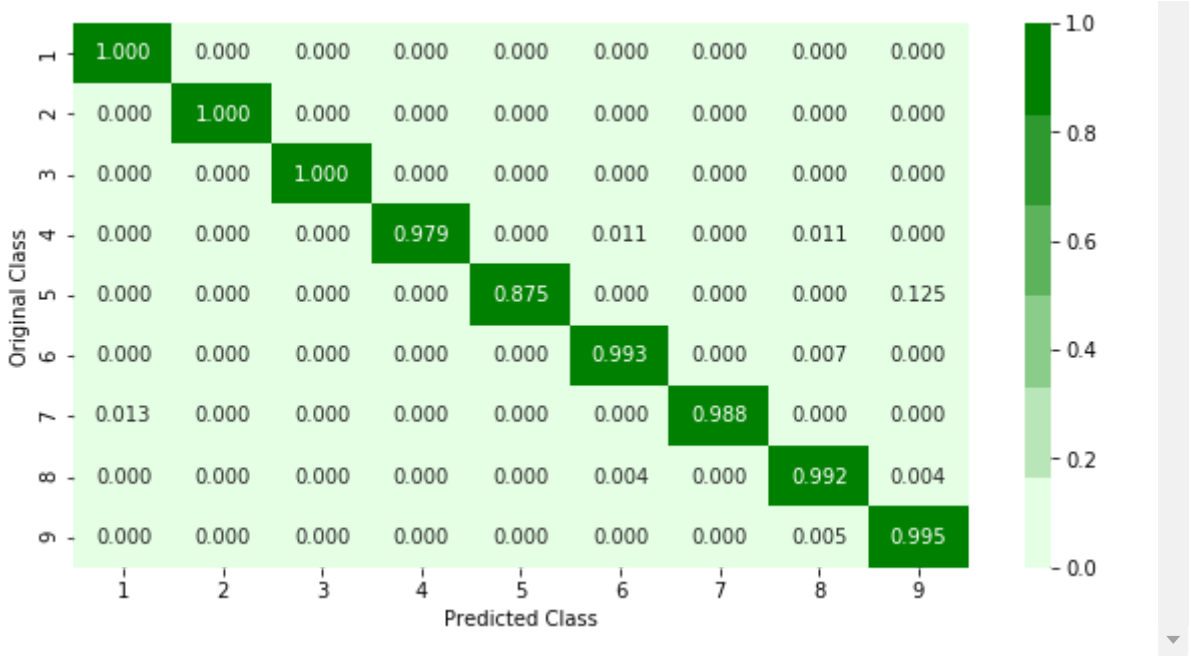
```
The train log loss is: 0.009920733822338627
The cross validation log loss is: 0.044422857517764446
The test log loss is: 0.025582587897521512
Number of misclassified points  0.36798528058877644
---------------------------------------------------- Confusion matrix --
--------------------------------------------------
```



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




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




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

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video (https://www.youtube.com/watch?v=VLQTRlLGz5Y#t=13m11s)) and include pixel intensity features to improve the logloss

```
1. you need to donwload the train from kaggle, which is of size ~17GB, afte
r extracting it will occupy ~128GB data your dirve

2. if you are having computation power limitations, you can try using googl
e colab, with GPU option enabled (you can search for how to enable GPU in c
olab) or you can work with the Google Cloud, check this tutorials by one of
our student: https://www.youtube.com/channel/UCRH_z-oM0LROvHPe_KYR4Wg (we s
uggest you to use GCP over Colab)

3. To Extract the .7z file in google cloud, once after you upload the file
into server, in your ipython notebook create a new cell and write thess com
mands
  a. !sudo apt-get install p7zip
  b. !7z x file_name.7z -o path/where/you/want/to/extract

https://askubuntu.com/a/341637
```

Extracting Bigram feature from byte files

In [6]:

```
# this part of code create a list of all possible bigram feature
byte_feature_="00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16
list_feature=byte_feature_.split(",")
bigram_feature_vocab=[]
for i in tqdm(list_feature):
    for j in list_feature:
        bigram_feature_vocab.append(" ".join([i,j]))
bigram_feature_vocab.append("?? ??")
array_bigram_feature=np.array(bigram_feature_vocab)
print(array_bigram_feature.shape)
```

```
100%|██████████| 256/256 [00:00<00:00, 14030.89it/s]

(65537,)
```

In [1]:

```
files = os.listdir('byteFiles')
filenames2=[]
feature_matrix = np.zeros((10868, 65537),dtype=int)
k=0
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is bi-gram bag of words
for file in tqdm(files):
    filenames2.append(file.split(".")[0])
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                #print(line)
                for i in range(len(line)-1):
                    mnemo=[]
                    mnemo.extend([line[i],line[i+1]])
                    if '??' in mnemo:
                        feature_matrix[k,65536]+=1
                    else:
                        hex_code="".join([line[i],line[i+1]])
                        feature_matrix[k,int(hex_code,16)]+=1

        byte_flie.close()
    k += 1
```

...

In [45]:

```
scipy.sparse.save_npz('feature.npz',scipy.sparse.csr_matrix(feature_matrix))
with open('index_feature.pkl','wb') as file_:
    pickle.dump(filenames2,file_)
```

In [7]:

```
with open('index_feature.pkl','rb') as file_:
    filenames2=pickle.load(file_)
    file_.close()
feature_matrix= scipy.sparse.load_npz('feature.npz')
dense_mat=scipy.sparse.csr_matrix.todense(feature_matrix)
```

In [8]:

```
bigram_dataframe=pd.DataFrame(dense_mat,columns=array_bigram_feature)
bigram_dataframe["ID"]=pd.Series(filenames2)
byte_bigram_w_size=pd.merge(bigram_dataframe,data_size_byte,on='ID',how='left')
byte_bigram_w_size.head()
```

Out[8]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	...	ff fa	ff fb	ff fc	ff fd	ff fe	ff f
0	66052	111	47	52	33	38	36	55	19	15	...	9	10	28	7	17	695
1	6189	52	34	103	26	54	34	44	32	58	...	32	22	24	20	16	101
2	16937	1169	812	795	557	558	333	395	643	13	...	6	48	18	3	24	8503
3	29335	783	470	436	249	113	139	117	209	237	...	23	63	53	29	33	4675
4	107822	676	416	434	665	493	511	434	640	339	...	222	101	137	104	155	11133

5 rows × 65540 columns

Image feature extraction from asm files

In [4]:

```
# refernece: https://github.com/saicharanarishanapally/microsoft-malware-detection
#converting asm into image

files=os.listdir('asmFiles')
for file in tqdm(files):
    name_file = file.split('.')[0]
    if(file.endswith('asm')):
            f=codecs.open("asmFiles/" +file, 'rb')
            filelen=os.path.getsize("asmFiles/"+file)
            width = int(filelen ** 0.5)
            rem = int(filelen / width)
            arr = array.array('B')
            arr.frombytes(f.read())
            f.close()
            reshaped = np.reshape(arr[:width * width], (width, width))
            reshaped = np.uint8(reshaped)
            imageio.imwrite('asm_image/' + name_file + '.png',reshaped)
```

Extracting top 800 pixel values

In [6]:

```
# reference: https://www.youtube.com/watch?v=VLQTRlLGz5Y

image_feature_matrix=np.zeros((10868,800))
pics=os.listdir("asm_image")
pic_id=[]
for i,pic in tqdm(enumerate(pics)):
    if(pic.endswith('png')):
        pic_id.append(pic.split(".")[0])
        image=imageio.imread('asm_image/'+pic)
        #print(image.shape)
        feature_vector=image.ravel()[:800]
        #print(feature_vector)
        image_feature_matrix[i,:]=feature_vector
```

```
10868it [14:34, 13.04it/s]
```

In [7]:

```
scipy.sparse.save_npz('pixel.npz',scipy.sparse.csr_matrix(image_feature_matrix))
with open('pic_id.pkl','wb') as file_:
    pickle.dump(pic_id,file_)
    file_.close()
```

In [8]:

```
with open('pic_id.pkl','rb') as file_:
    pic_id=pickle.load(file_)
    file_.close()
image_feature_matrix= scipy.sparse.load_npz('pixel.npz')
dense_pixel_mat=scipy.sparse.csr_matrix.todense(image_feature_matrix)
```

In [9]:

```
pixel_list=np.array(["pixel "+str(i) for i in range(800)])
asm_result=pd.read_csv("result_asm_size.csv")
```

In [10]:

```
asm_pixel=pd.DataFrame(dense_pixel_mat,columns=pixel_list,dtype=float)
asm_pixel["ID"]=pd.Series(pic_id)
result_asm_pixel=pd.merge(asm_result.drop("Unnamed: 0",axis=1),asm_pixel,on='ID',ho
result_asm_pixel.head()
```

Out[10]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 854 columns

Feature selection

Bigram features

In [9]:

```
byte_bigram_w_size_y=byte_bigram_w_size['Class']
byte_bigram_w_size_x=byte_bigram_w_size.drop(['ID','Class'],axis=1)
```

In [11]:

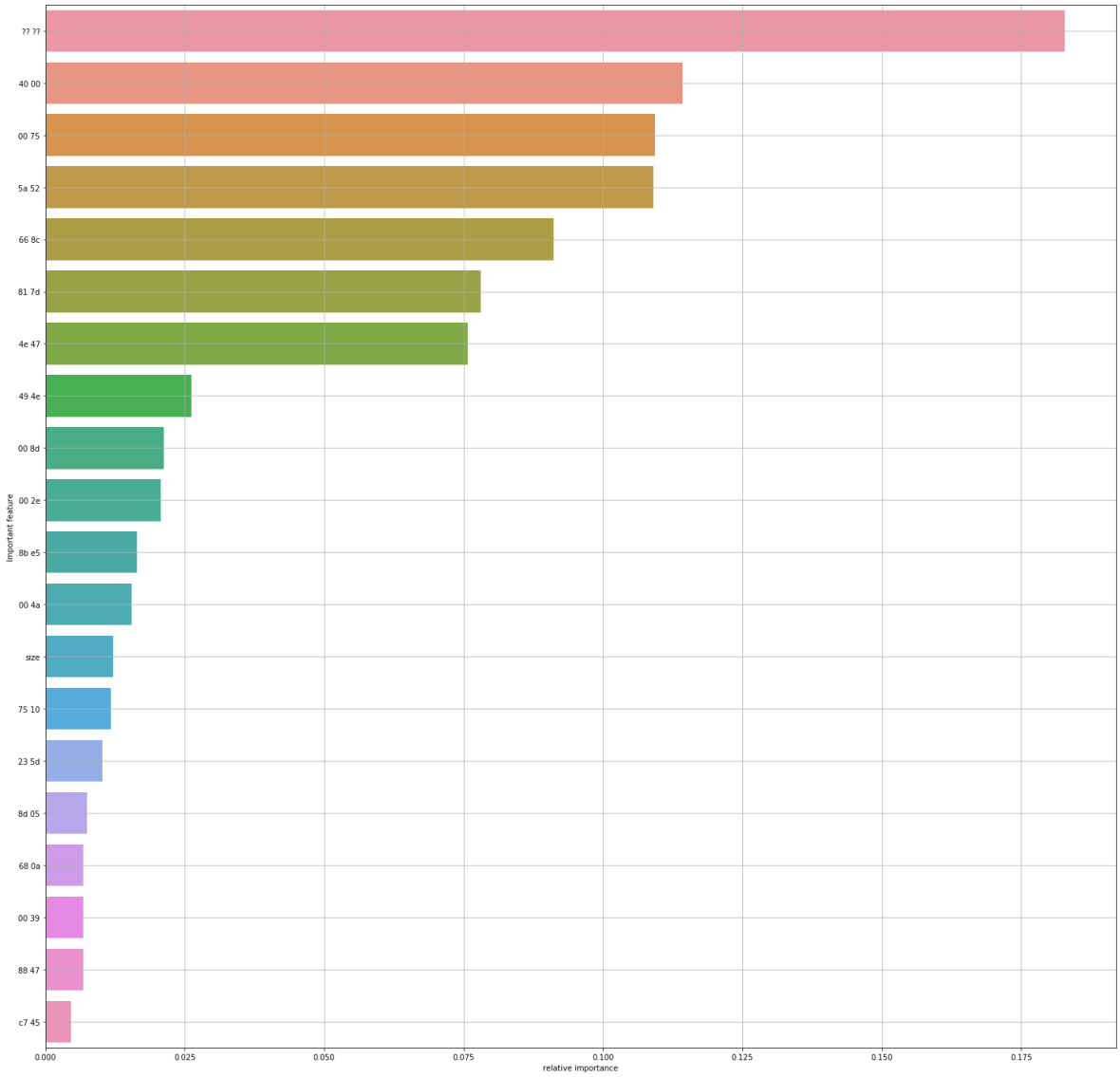
```
best_model=DecisionTreeClassifier(criterion='gini',max_depth=None,min_samples_split
best_model.fit(byte_bigram_w_size_x.values,byte_bigram_w_size_y.values)
feature_importance=best_model.feature_importances_
print("feature shape:",feature_importance.shape)
non_feature_index=np.array(np.where(feature_importance>0))
print("non zero feature shape:",non_feature_index.shape)
```

```
feature shape: (65538,)
non zero feature shape: (1, 65)
```

Top 20 important features

In [12]:

```
feature_im_sort=np.sort(feature_importance)[::-1][:20]
x_=np.argsort(feature_importance)[::-1]
feature_x=byte_bigram_w_size_x.columns[x_][:20]
plt.figure(figsize=(25,25))
sns.barplot(x=feature_im_sort,y=feature_x)
plt.ylabel('Important feature')
plt.xlabel('relative importance')
plt.grid()
plt.show()
```



In [13]:

```
im_bi_fea=byte_bigram_w_size_x.values[:,non_feature_index[0]]
print(im_bi_fea.shape)
im_bi_feature=byte_bigram_w_size_x.columns[non_feature_index[0]]
bigram_impo=pd.DataFrame(im_bi_fea,columns=im_bi_feature)
bigram_impo['ID']=byte_bigram_w_size["ID"].values
bigram_impo.head()
```

```
(10868, 65)
```

Out[13]:

	00 00	00 2e	00 39	00 4a	00 75	00 8d	00 a3	00 d8	00 ff	03 5c	...	cc fb	e2 00	f e
0	66052.0	21.0	21.0	20.0	93.0	13.0	7.0	3.0	42.0	6.0	...	6.0	8.0	5.(
1	6189.0	16.0	16.0	30.0	30.0	24.0	18.0	18.0	54.0	24.0	...	14.0	12.0	20.(
2	16937.0	26.0	348.0	11.0	456.0	413.0	30.0	12.0	3844.0	0.0	...	0.0	10.0	6.(
3	29335.0	79.0	129.0	99.0	234.0	281.0	91.0	69.0	1696.0	67.0	...	3.0	54.0	0.(
4	107822.0	367.0	396.0	504.0	578.0	682.0	351.0	367.0	1691.0	6.0	...	11.0	473.0	13.(

5 rows × 66 columns

In [14]:

```
bigram_impo.to_csv('imp_bigram_w_size.csv')
```

ASM with pixel data

In [11]:

```
result_asm_pixel_y=result_asm_pixel.Class
result_asm_pixel_x=result_asm_pixel.drop(['ID','Class'],axis=1)
```

In [12]:

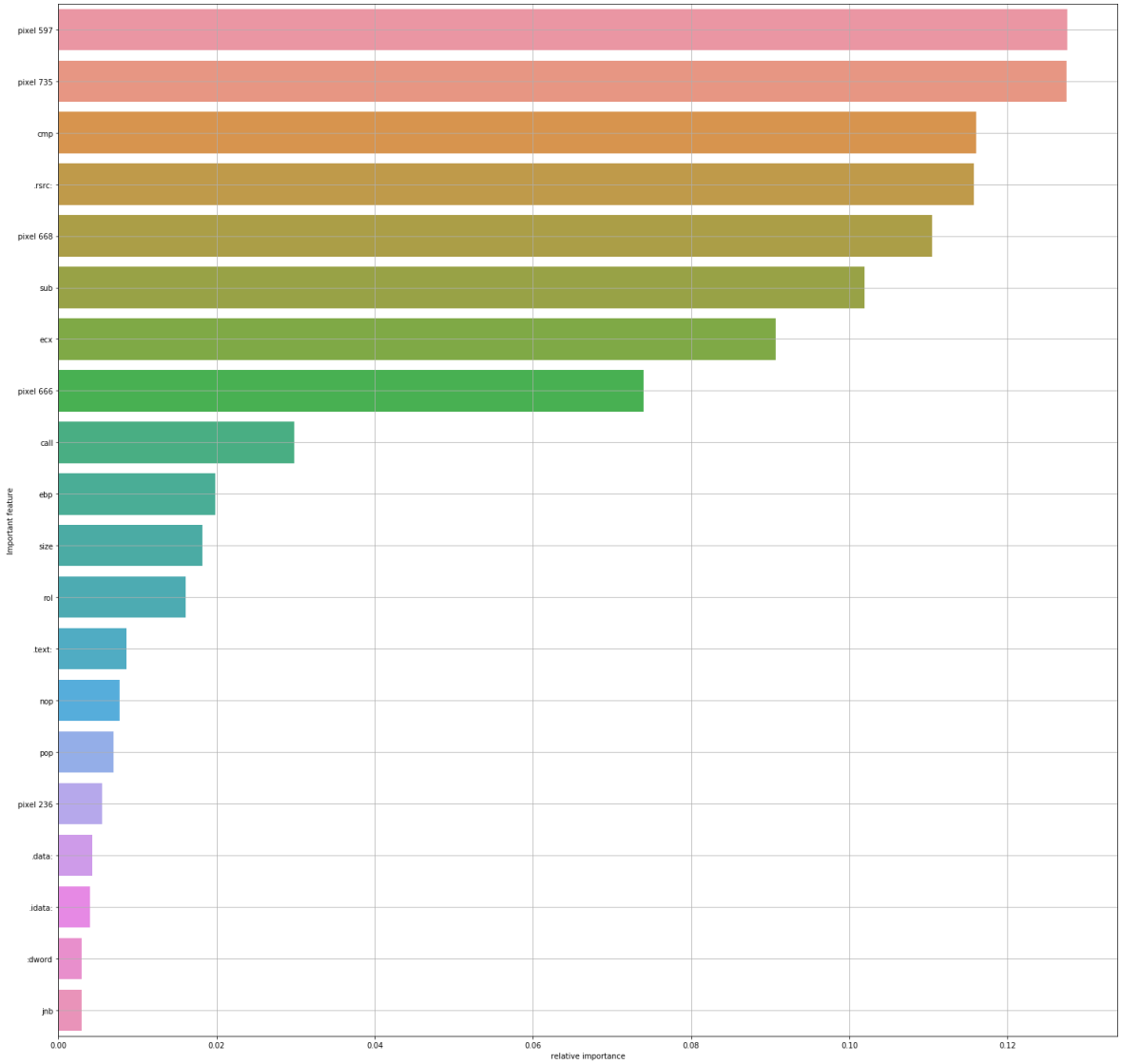
```
best_model=DecisionTreeClassifier(criterion='gini',max_depth=None,min_samples_split
best_model.fit(result_asm_pixel_x.values,result_asm_pixel_y.values)
feature_importance=best_model.feature_importances_
print("feature shape:",feature_importance.shape)
non_feature_index=np.array(np.where(feature_importance>0))
print("non zero feature shape:",non_feature_index.shape)
```

```
feature shape: (852,)
non zero feature shape: (1, 32)
```

Top 20 features

In [13]:

```
feature_im_sort=np.sort(feature_importance)[::-1][:20]
x_=np.argsort(feature_importance)[::-1]
feature_x=result_asm_pixel_x.columns[x_][:20]
plt.figure(figsize=(25,25))
sns.barplot(x=feature_im_sort,y=feature_x)
plt.ylabel('Important feature')
plt.xlabel('relative importance')
plt.grid()
plt.show()
```



In [14]:

```
im_as_pix=result_asm_pixel_x.values[:,non_feature_index[0]]
im_as_pixfeature=result_asm_pixel_x.columns[non_feature_index[0]]
asm_impo=pd.DataFrame(im_as_pix,columns=im_as_pixfeature)
asm_impo['Class']=result_asm_pixel.Class.values
asm_impo['ID']=result_asm_pixel["ID"].values
asm_impo.head()
```

Out[14]:

	.text:	.idata:	.data:	.rsrc:	pop	xor	nop	sub	imul	or	...	pixel 236	pixel 589	pixel 597	pixel 606	
0	744.0	127.0	57.0	3.0	19.0	18.0	0.0	5.0	0.0	6.0	...	48.0	48.0	10.0	48.0	
1	838.0	103.0	49.0	3.0	26.0	19.0	0.0	8.0	1.0	0.0	...	53.0	67.0	10.0	67.0	
2	427.0	50.0	43.0	3.0	18.0	9.0	0.0	5.0	1.0	4.0	...	48.0	48.0	10.0	48.0	
3	227.0	43.0	19.0	3.0	6.0	8.0	0.0	3.0	0.0	1.0	...	48.0	49.0	10.0	49.0	
4	402.0	59.0	170.0	3.0	5.0	3.0	0.0	2.0	2.0	0.0	...	51.0	54.0	10.0	54.0	

5 rows × 34 columns

In [15]:

```
asm_impo.to_csv("imp_asm_w_pix.csv",)
```

Merging the selected feature

In [1]:

```
asm_impo=pd.read_csv("imp_asm_w_pix.csv")
#print(asm_impo.head())
asm_impo=asm_impo.drop(['Unnamed: 0'],axis=1)
bigram_impo=pd.read_csv('imp_bigram_w_size.csv')
bigram_impo=bigram_impo.drop(['Unnamed: 0'],axis=1)
imp_byte_asm=pd.merge(asm_impo,bigram_impo,on='ID',how='left')
imp_byte_asm.head()
```

Out[1]:

	.text:	.idata:	.data:	.rsrc:	pop	xor	nop	sub	imul	or	...	c9 02	cc fb	e2 00	fe ef	ff 15	f
0	744.0	127.0	57.0	3.0	19.0	18.0	0.0	5.0	0.0	6.0	...	4.0	1.0	3.0	2.0	34.0	1(
1	838.0	103.0	49.0	3.0	26.0	19.0	0.0	8.0	1.0	0.0	...	1.0	4.0	6.0	0.0	21.0	
2	427.0	50.0	43.0	3.0	18.0	9.0	0.0	5.0	1.0	4.0	...	3.0	0.0	4.0	3.0	10.0	
3	227.0	43.0	19.0	3.0	6.0	8.0	0.0	3.0	0.0	1.0	...	1.0	3.0	3.0	2.0	6.0	
4	402.0	59.0	170.0	3.0	5.0	3.0	0.0	2.0	2.0	0.0	...	1.0	0.0	8.0	3.0	8.0	

5 rows × 99 columns

In [4]:

```
imp_byte_asm=normalize(imp_byte_asm)
imp_byte_asm.columns
```

Out[4]:

```
Index(['.text:', '.idata:', '.data:', '.rsrc:', 'pop', 'xor', 'nop',
'sub',
       'imul', 'or', 'shr', 'cmp', 'call', 'rol', 'jnb', '.dll', 'st
d::',
       ':dword', 'eax', 'ecx', 'ebp', 'esp', 'eip', 'size_x', 'pixel 2
36',
       'pixel 589', 'pixel 597', 'pixel 606', 'pixel 633', 'pixel 66
6',
       'pixel 668', 'pixel 735', 'Class', 'ID', '00 00', '00 2e', '00
39',
       '00 4a', '00 75', '00 8d', '00 a3', '00 d8', '00 ff', '03 5c',
'04 85',
       '0d 2f', '0f af', '10 68', '17 25', '1b b7', '23 5d', '24 08',
'24 70',
       '2f e7', '30 55', '35 60', '35 78', '38 8d', '3e f1', '40 00',
'41 44',
       '46 69', '49 4e', '4d 5d', '4e 47', '55 d5', '5a 52', '66 8c',
'67 6c',
       '68 0a', '6c 6c', '6c 73', '70 77', '75 10', '77 70', '80 3e',
'81 7d',
       '88 47', '89 48', '8b 44', '8b 55', '8b e5', '8d 00', '8d 05',
'c0 05',
       'c3 90', 'c4 04', 'c5 89', 'c7 45', 'c9 02', 'cc fb', 'e2 00',
'fe ef',
       'ff 15', 'ff 55', 'ff 89', 'ff d0', '?? ??', 'size_y'],
      dtype='object')
```

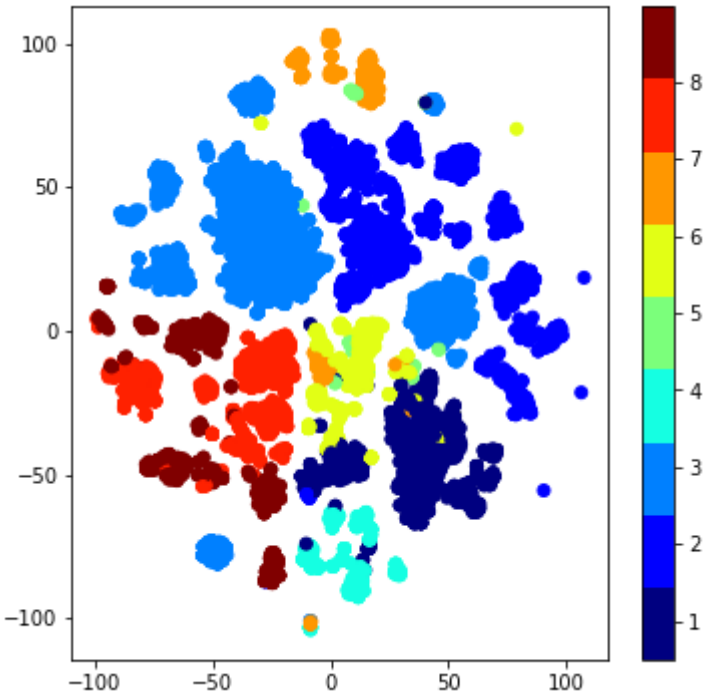
In [5]:

```
result_y=imp_byte_asm.Class
result_x=imp_byte_asm.drop(['Class','ID'],axis=1)
```

TSNE visualization

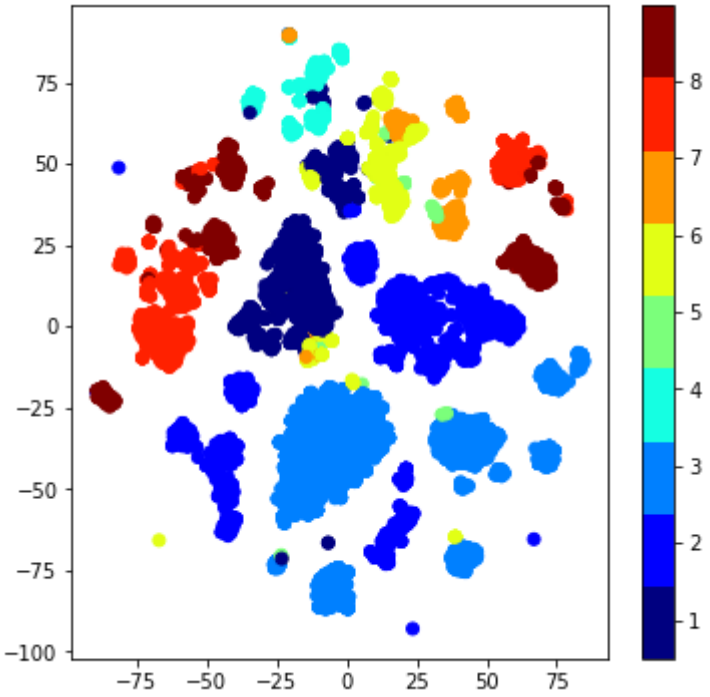
In [8]:

```
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_x,)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.figure(figsize=(6,6))
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()
```



In [22]:

```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x,)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.figure(figsize=(6,6))
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()
```



Modelling with selected feature

In [10]:

```
x_train,test_x,y_train,test_y=train_test_split(result_x,result_y,stratify=result_y,
train_x,cv_x,train_y,cv_y=train_test_split(x_train,y_train,stratify=y_train,test_si
print(train_x.shape,train_y.shape)
print(cv_x.shape,cv_y.shape)
print(test_x.shape,test_y.shape)
```

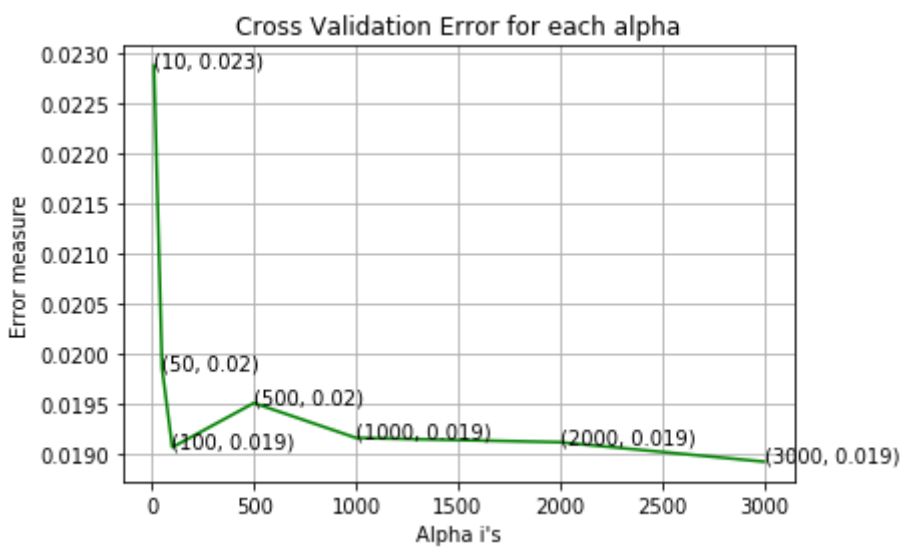
```
(6955, 97) (6955,)
(1739, 97) (1739,)
(2174, 97) (2174,)
```

Random Forest on selected feature

In [24]:

```
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(train_x, train_y)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    predict_y = sig_clf.predict_proba(cv_x)
    cv_log_error_array.append(log_loss(cv_y, predict_y, labels=r_cfl.classes_, eps=

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=
r_cfl.fit(train_x, train_y)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(train_x, train_y)

predict_y = sig_clf.predict_proba(train_x)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",l
predict_y = sig_clf.predict_proba(cv_x)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
log_loss for c =  10 is 0.022883483030221495
log_loss for c =  50 is 0.019851207466211402
log_loss for c =  100 is 0.01906105684881852
log_loss for c =  500 is 0.019504731002560387
log_loss for c =  1000 is 0.01915597074641316
log_loss for c =  2000 is 0.019112357191470364
log_loss for c =  3000 is 0.018917519871888853
```



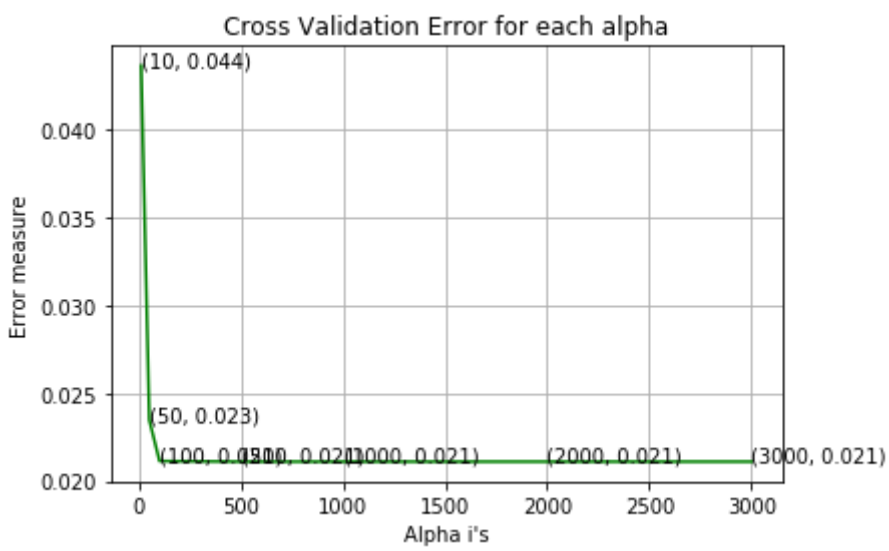

```
For values of best alpha =  3000 The train log loss is: 0.008942560472
770651
For values of best alpha =  3000 The cross validation log loss is: 0.0
18917519871888853
For values of best alpha =  3000 The test log loss is: 0.0204828242149
43126
```

LGBM on selected feature

In [25]:

```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=LGBMClassifier(n_estimators=i)
    x_cfl.fit(train_x.values, train_y.values)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(train_x.values, train_y.values)
    predict_y = sig_clf.predict_proba(cv_x.values)
    cv_log_error_array.append(log_loss(cv_y.values, predict_y, labels=x_cfl.classes

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=LGBMClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(train_x.values, train_y.values,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(train_x.values, train_y.values)

predict_y = sig_clf.predict_proba(train_x.values)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",l
predict_y = sig_clf.predict_proba(cv_x.values)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x.values)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
log_loss for c =  10 is 0.043662606212730444
log_loss for c =  50 is 0.02346355217185988
log_loss for c =  100 is 0.021132676718924784
log_loss for c =  500 is 0.021099124114480335
log_loss for c =  1000 is 0.021099121015988185
log_loss for c =  2000 is 0.021099121863816222
log_loss for c =  3000 is 0.02109912410412019
```




```
[LightGBM] [Warning] num_threads is set with nthread=-1, will be overr
idden by n_jobs=-1. Current value: num_threads=-1
[LightGBM] [Warning] num_threads is set with nthread=-1, will be overr
idden by n_jobs=-1. Current value: num_threads=-1
[LightGBM] [Warning] num_threads is set with nthread=-1, will be overr
idden by n_jobs=-1. Current value: num_threads=-1
For values of best alpha =  1000 The train log loss is: 0.009700514577
067897
For values of best alpha =  1000 The cross validation log loss is: 0.0
21099121863815625
For values of best alpha =  1000 The test log loss is: 0.0416225317433
00384
```

Hyperparameter for lightgbm using gridsearch

In [11]:

```
from sklearn.model_selection import GridSearchCV
x_cfl=LGBMClassifier()

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:    3.1s
[Parallel(n jobs=-1)]: Done  10 tasks      | elapsed:    4.0s
```

In [12]:

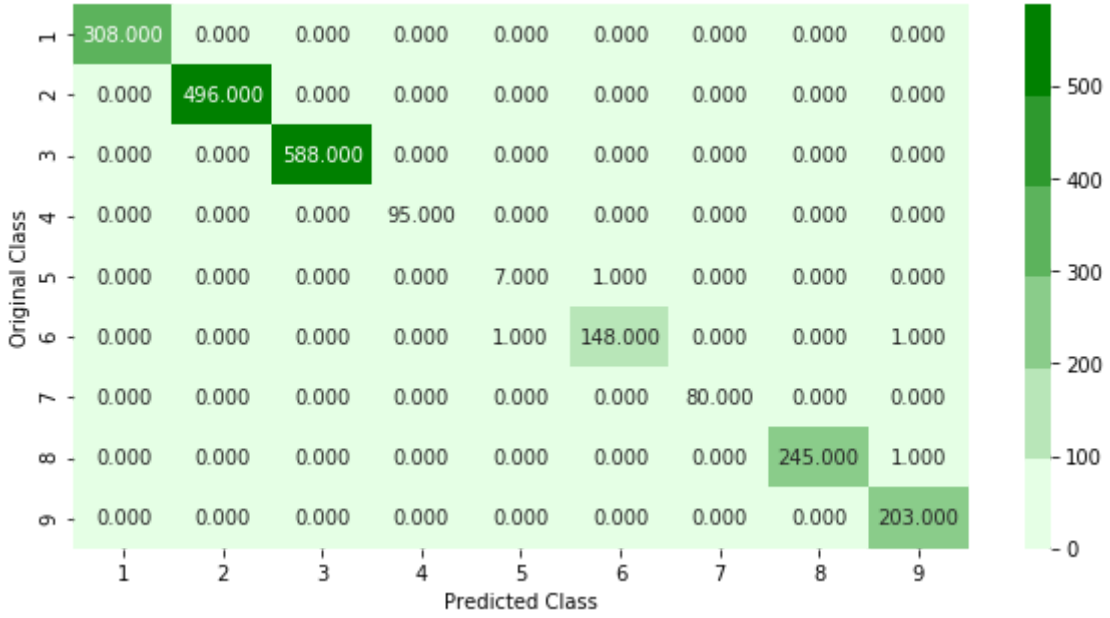
```
print (random_cfl.best_params_)
```

```
{'colsample_bytree': 0.1, 'learning_rate': 0.03, 'max_depth': 3, 'n_es
timators': 2000, 'subsample': 0.1}
```

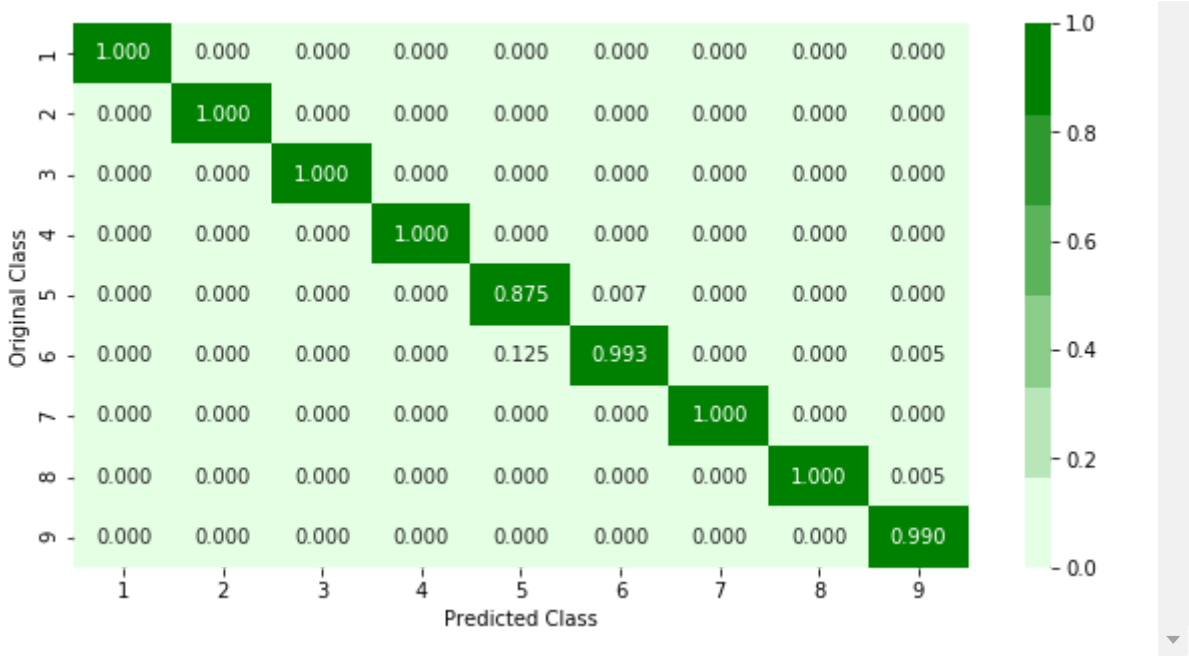
In [15]:

```
x_cfl=LGBMClassifier(n_estimators=2000,max_depth=3,learning_rate=0.03,colsample_byt
x_cfl.fit(train_x.values,train_y,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(train_x.values, train_y.values)

predict_y = sig_clf.predict_proba(train_x.values)
print ("The train log loss is:",log_loss(train_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x.values)
print("The cv log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x.values)
print("The test log loss is:",log_loss(test_y, predict_y))
plot_confusion_matrix(test_y,sig_clf.predict(test_x.values))
```

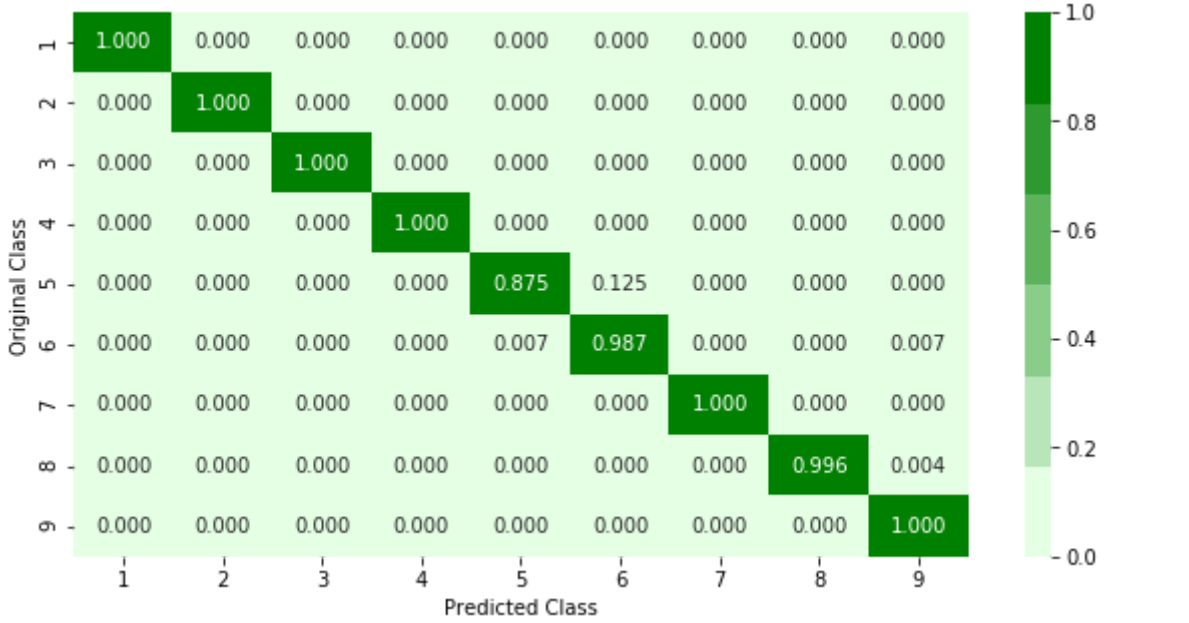
```
The train log loss is: 0.006475078103043351
The cv log loss is: 0.007259920493756501
The test log loss is: 0.011970586774216321
Number of misclassified points  0.18399264029438822
---------------------------------------------------- Confusion matrix --
-----------------------------------------------------
```




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




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




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

In [16]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names =['FEATURE','MODEL','LOG-LOSS']
x.add_row(['byte_file','random model',2.48981])
x.add_row(["byte file", 'K-nn', 0.204055])
x.add_row(['byte file','Logistic Regression',0.536438])
x.add_row(['byte file','Random forest',0.0993])
x.add_row(['byte file','lightgbm',0.06952])
x.add_row(["asm", 'K-nn', 0.08328 ])
x.add_row(['asm','Logistic Regression',0.368890])
x.add_row(['asm','Random forest',0.029091])
x.add_row(['asm','lightgbm',0.0251248])
x.add_row(['byte file + asm','Random forest',0.029091])
x.add_row(['byte file + asm','lightgbm',0.025582])
x.add_row(['byte file + asm + advance feature','Random forest',0.020482])
x.add_row(['byte file + asm + advance feature','lightgbm',0.01197])
print(x)
```

```
+-----------------------------------+---------------------+-----------
+
|              FEATURE              |        MODEL        |  LOG-LOSS
|
+-----------------------------------+---------------------+-----------
+
|             byte_file             |     random model    |  2.48981
|
|             byte file             |         K-nn        |  0.204055
|
|             byte file             | Logistic Regression |  0.536438
|
|             byte file             |    Random forest    |   0.0993
|
|             byte file             |       lightgbm      |  0.06952
|
|                asm                |         K-nn        |  0.08328
|
|                asm                | Logistic Regression |  0.36889
|
|                asm                |    Random forest    |  0.029091
|
|                asm                |       lightgbm      | 0.0251248
|
|          byte file + asm          |    Random forest    |  0.029091
|
|          byte file + asm          |       lightgbm      |  0.025582
|
| byte file + asm + advance feature |    Random forest    |  0.020482
|
| byte file + asm + advance feature |       lightgbm      |  0.01197
|
+-----------------------------------+---------------------+-----------
+
```

Conclusion

1. Dataset given have two files, byte and asm. Data have 9 clasess and is imbalanced in nature.

2. The byte file contains hexadecimal code with address of registors.
3. Asm file contains the code written in assembly language also called opcode.
4. Given two files, our task here is to predict that these files belongs to one of these 9 classes.
5. To featurize the two files,used NLP technique .
6. fisrt, we have used 1-gram on the byte and asm files. Using this featurization we have reduce the logloss to 0.025
7. secondly, we have used bigram on the byte files as from domain knowledge we know that assembly has format like instruction with address of resistors where data is stored. These bigram feature have more information embedded.