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,
[esp+8]
.text:00401005 50                                   push    eax
.text:00401006 8B F1                                   mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                            call
??0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * c
onst &)
.text:0040100D C7 06 08    BB 42 00                          mov     dw
ord 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 1
0h
.text:00401020 C7 01 08    BB 42 00                          mov     dw
ord ptr [ecx],    offset off_42BB08
.text:00401026 E9 26 1C    00 00                             jmp     s
ub_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     dw
ord ptr [esi],    offset off_42BB08
.text:00401039 E8 13 1C    00 00                             call    s
ub_402C51
.text:0040103E F6 44 24    08 01                             test    b
yte ptr    [esp+8], 1
.text:00401043 74 09                                   jz      short loc
_40104E
.text:00401045 56                                   push    esi
.text:00401046 E8 6C 1E    00 00                             call
??3@YAXPAX@Z    ; operator 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 clas
sify 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 Learing Objectives and Constraints

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

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

In [ ]:
```
# !pip --version
# import matplotlib
# matplotlib.__version__
```

3. Exploratory Data Analysis

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
# import matplotlib
# matplotlib.use('nbAgg')
%matplotlib notebook
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize
from nltk.util import ngrams
# import h5py
import copy
```

In [ ]:

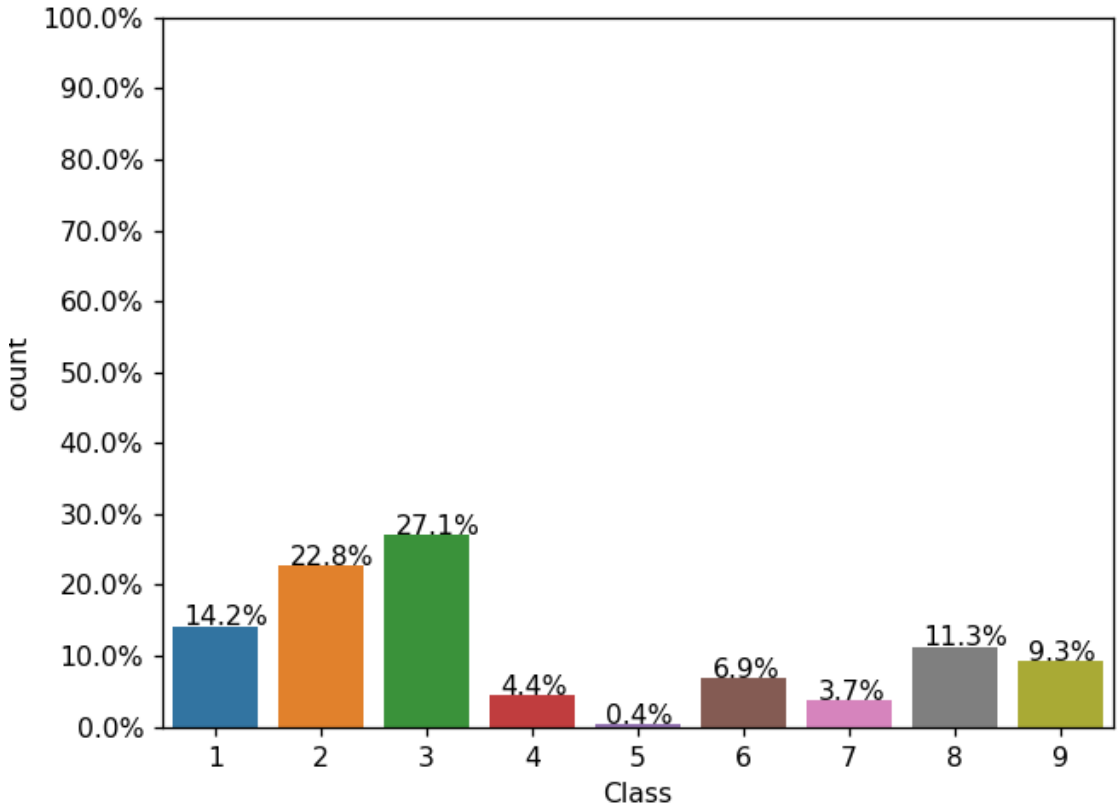
```
#separating byte files and asm files

source = 'train'
destination_1 = 'byteFiles'
destination_2 = 'asmFiles'

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


# os.makedirs('abc')
# shutil.move('a/abc.txt', 'abc/abc.txt')

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

# so by the end of this snippet we will separate all the .byte files and .as
if os.path.isdir(source):
    data_files = os.listdir(source)
    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 [ ]:

```
# close previous plot to reder new one
plt.close()
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

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

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [ ]:
```
# 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=35615717
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=15
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file
    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_byte
print (data_size_byte.head())
```

```
                     ID      size  Class
0  01azqd4InC7m9JpocGv5  4.234863      9
1  01IsoiSMh5gxyDYTl4CB  5.538818      2
2  01jsnpXSAlgw6aPeDxrU  3.887939      9
3  01kcPWA9K2BOxQeS5Rju  0.574219      1
4  01SuzwMJEIXsK7A8dQbl  0.370850      8
```

In [ ]:
```
# write size data in csv file
data_size_byte.to_csv('data_size_byte.csv')
```

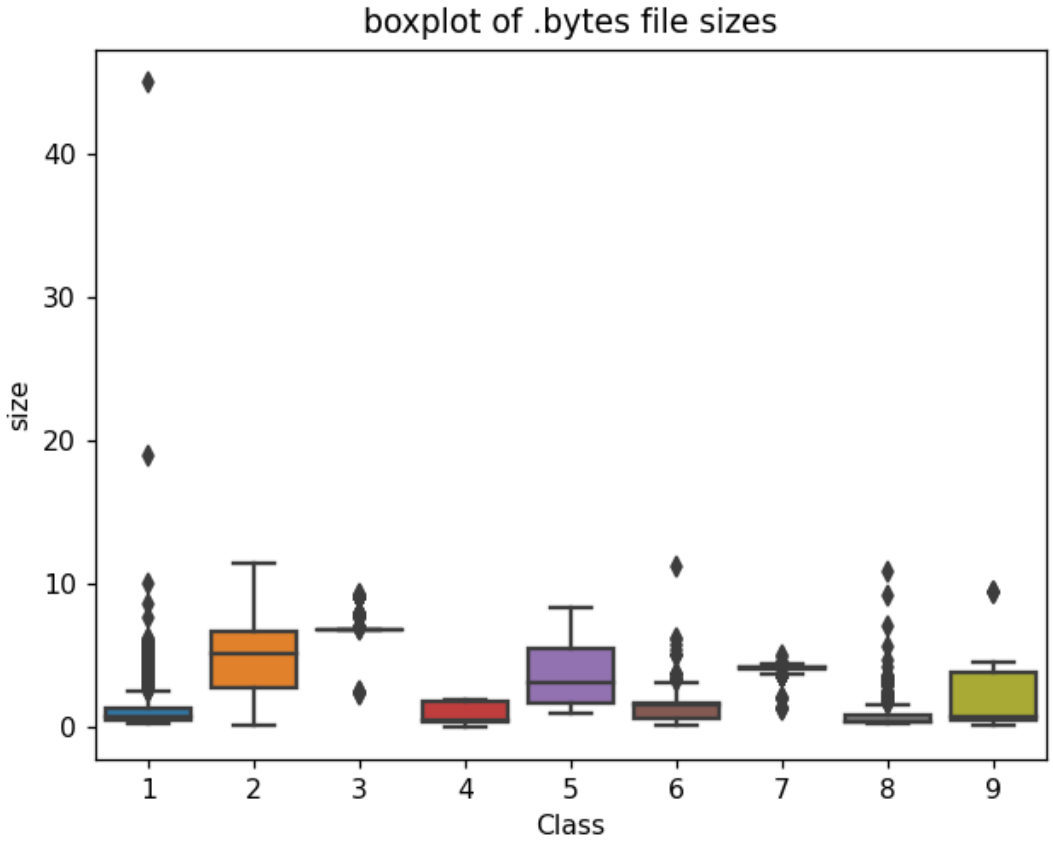
In [ ]:
```
data_size_byte = pd.read_csv('data_size_byte.csv')
print (data_size_byte.head())
```

```
   Unnamed: 0                    ID      size  Class
0           0  01azqd4InC7m9JpocGv5  4.234863      9
1           1  01IsoiSMh5gxyDYTl4CB  5.538818      2
2           2  01jsnpXSAlgw6aPeDxrU  3.887939      9
3           3  01kcPWA9K2BOxQeS5Rju  0.574219      1
4           4  01SuzwMJEIXsK7A8dQbl  0.370850      8
```

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

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

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



3.2.3 feature extraction from byte files

In [ ]:

```
# time taking task, take approx. 30-45 mins
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000

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

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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,1
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()
```

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

Out[6]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	368
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	653

2 rows × 258 columns

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

Out[7]:

	Unnamed: 0	ID	size	Class
0	0	01azqd4InC7m9JpocGv5	4.234863	9
1	1	01IsoiSMh5gxyDYTl4CB	5.538818	2

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

Out[8]:

	ID	0	1	2	3	4	5	6	7	8	...	fa	f
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3211	309
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	281	30

2 rows × 261 columns

In [ ]:
```
# 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_va
    return result1
result = normalize(byte_features_with_size)
```

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

Out[10]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747

2 rows × 261 columns

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

Out[11]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148

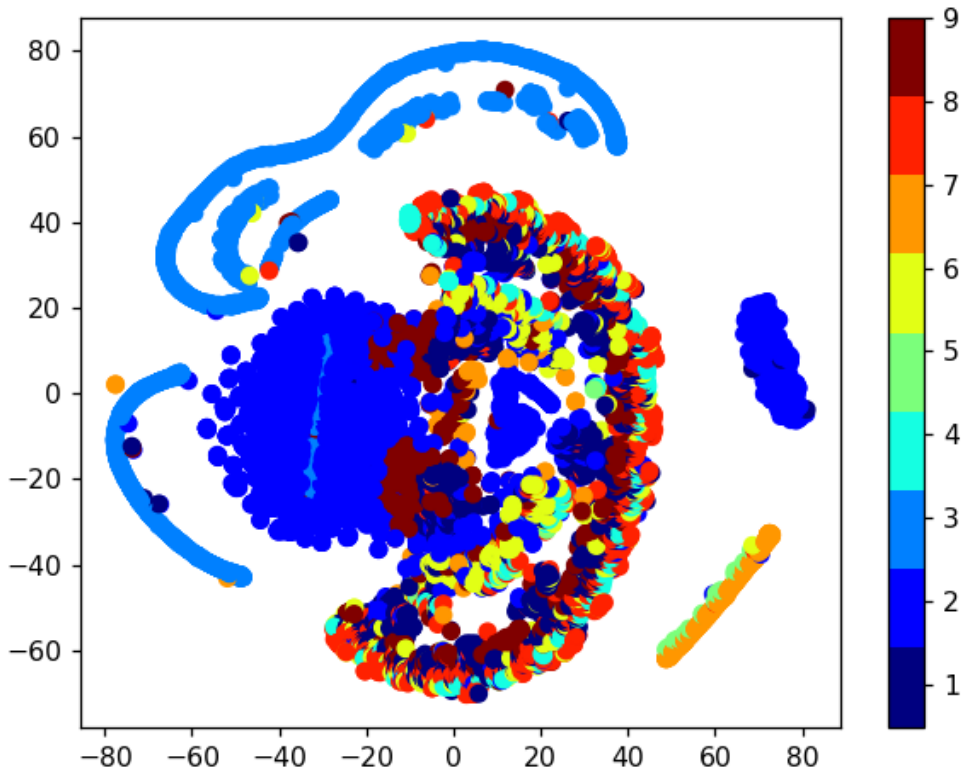
5 rows × 261 columns

3.2.4 Multivariate Analysis

In [ ]:

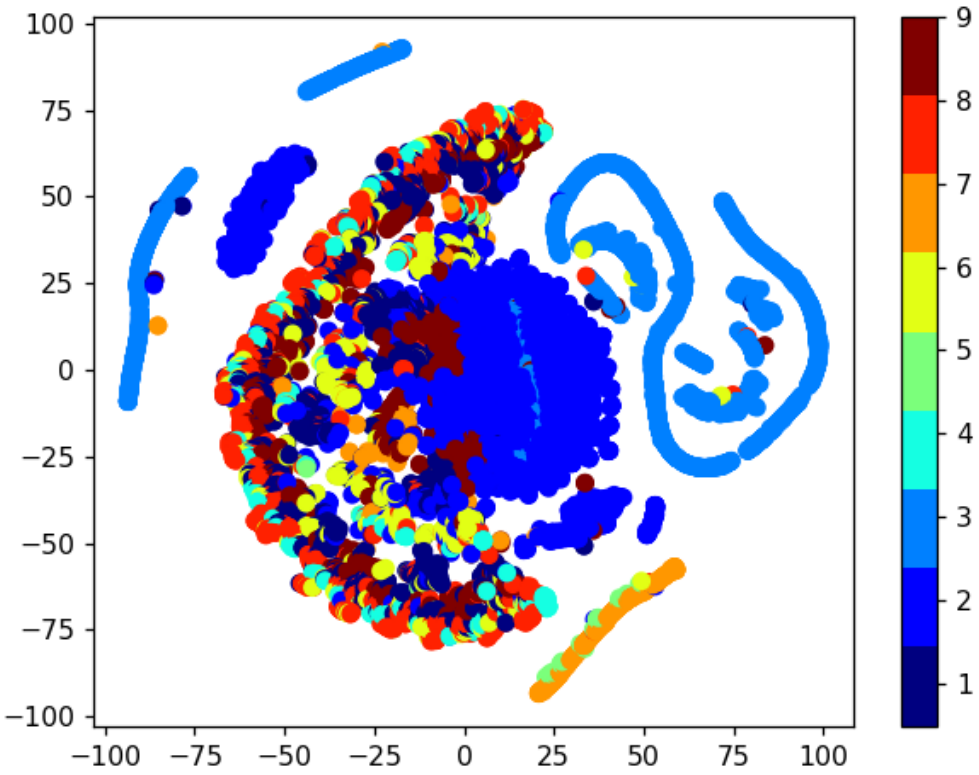
```
plt.close()
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))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

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



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

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



Train Test split

In [ ]:
```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of out
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class
# split the train data into train and cross validation by maintaining same d
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_
```

In [ ]:

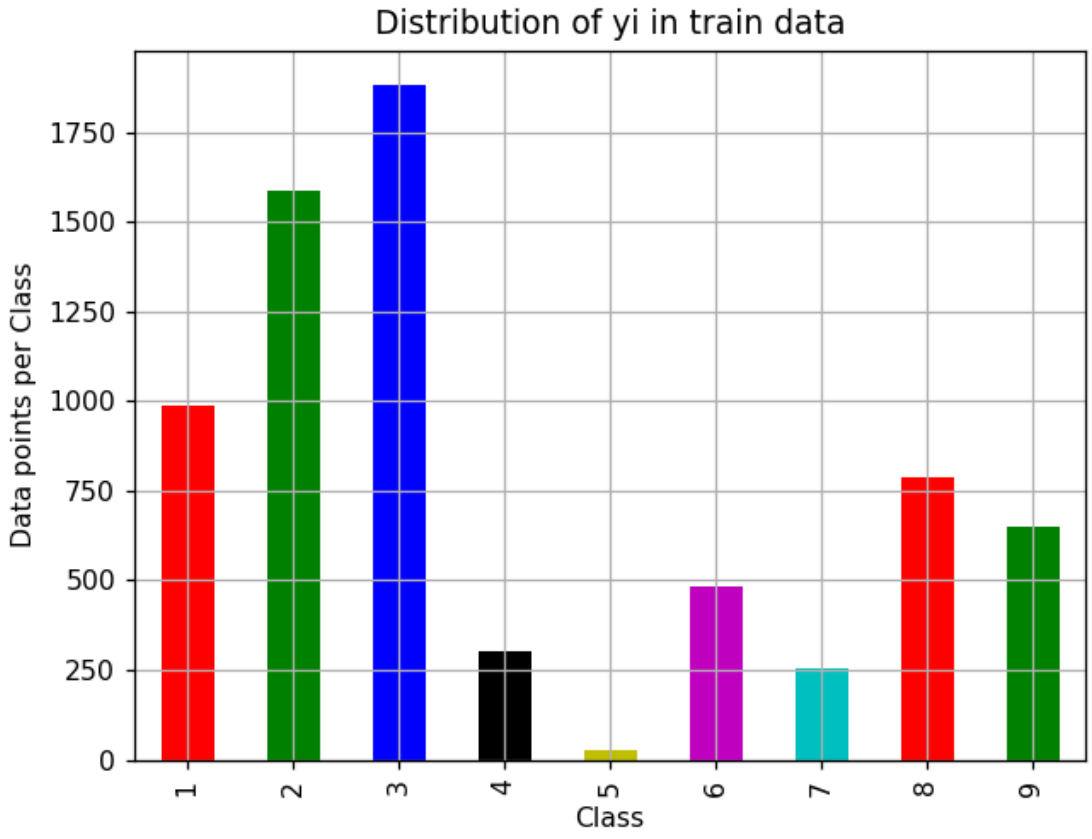
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
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 [ ]:

```
plt.close()
# it returns a dict, keys as class labels and values as the number of data p
train_class_distribution = y_train.value_counts().sort_index()#sortlevel()
test_class_distribution = y_test.value_counts().sort_index()#.sortlevel()
cv_class_distribution = y_cv.value_counts().sort_index()#.sortlevel()

# my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black,
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.ar
# -(train_class_distribution.values): the minus sign will give us in decreas
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distributio
```

<IPython.core.display.Javascript object>



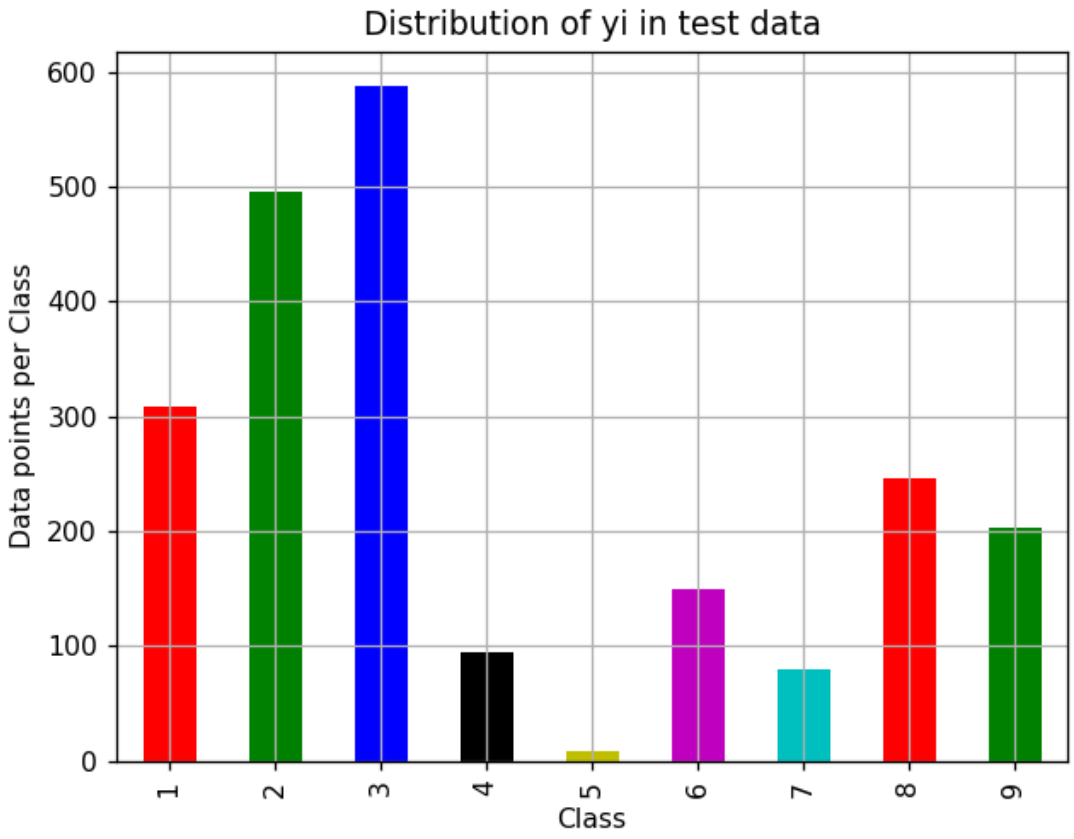

```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
```

```
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
```

In [ ]:

```
plt.close()
# my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black,
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()

print('-'*80)
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.ar
# -(train_class_distribution.values): the minus sign will give us in decreas
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
```

<IPython.core.display.Javascript object>



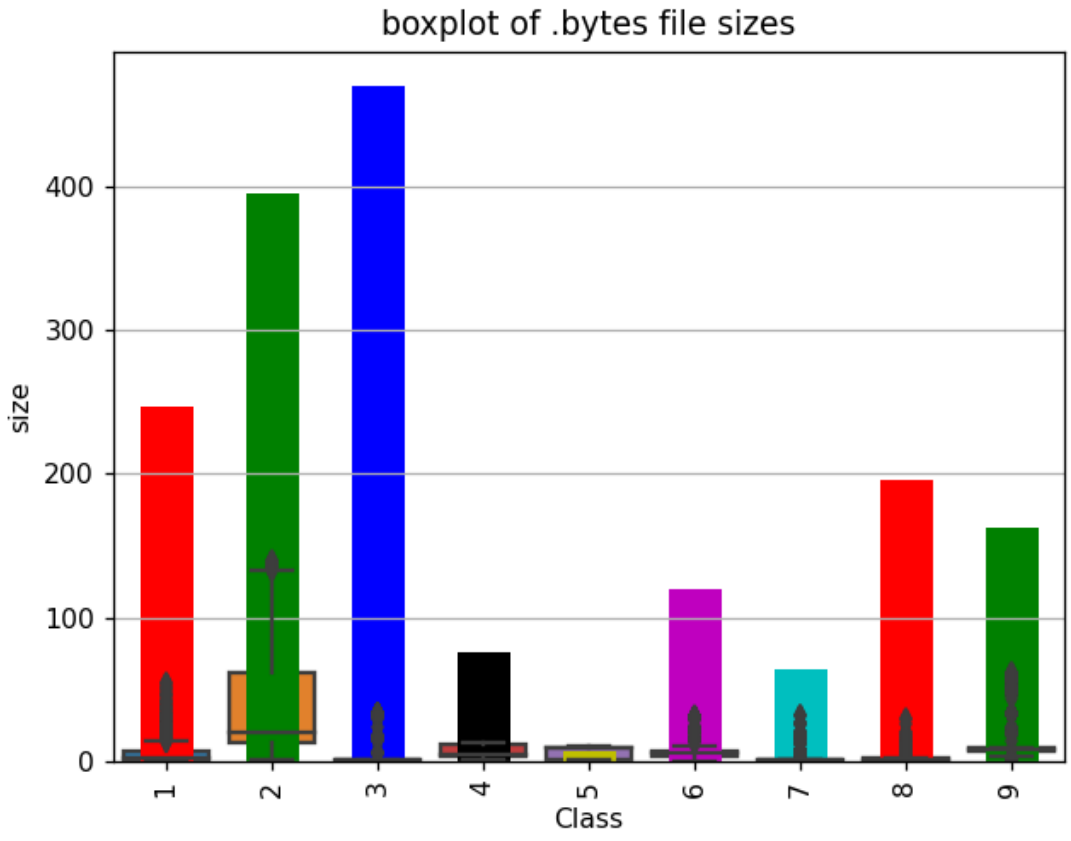

```
--------------------------------------------------------------------------------
-
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of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

```
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
```

In [ ]:

```
plt.close()
# my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black,
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()

print('-'*80)
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.ar
# -(train_class_distribution.values): the minus sign will give us in decreas
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.v
```

<IPython.core.display.Javascript object>




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

```
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

In [ ]:

```
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(te
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds t
    # 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
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds t
    # 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, yti
    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, yti
    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, yti
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

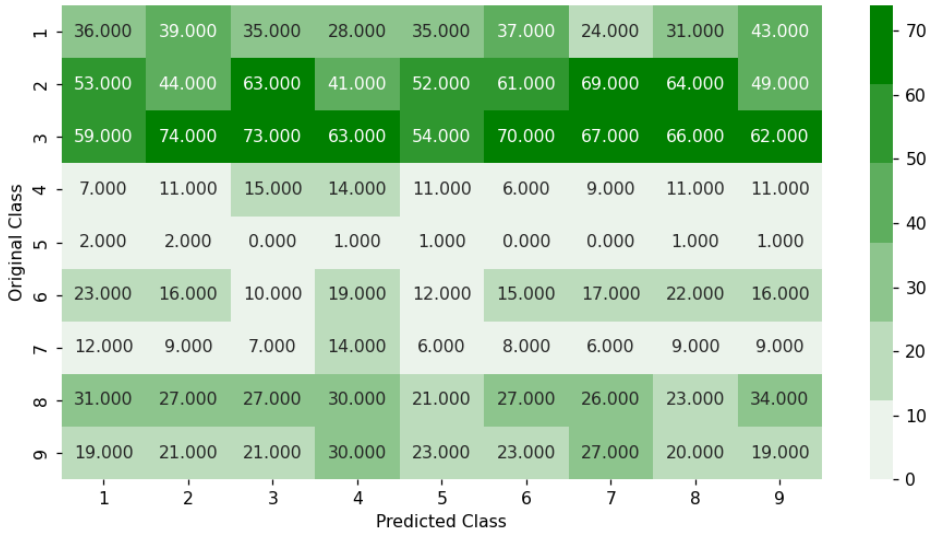
4.1.1. Random Model

In [ ]:

```
# 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 th
# 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,c


# 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_predic

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

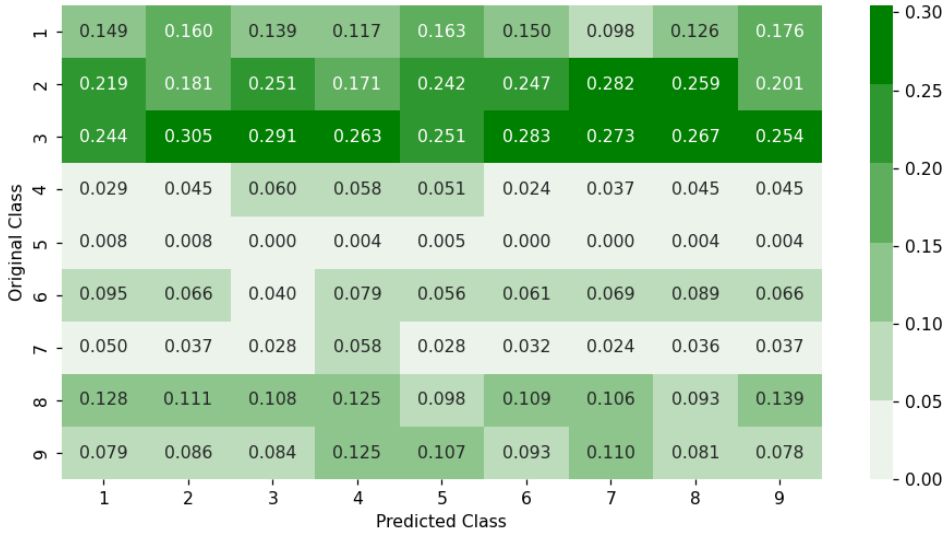
```
Log loss on Cross Validation Data using Random Model 2.4661664622685913
Log loss on Test Data using Random Model 2.4989638402300267
Number of misclassified points  89.37442502299908
-------------------------------------------------- Confusion matrix -----------
----------------------------------------
```

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

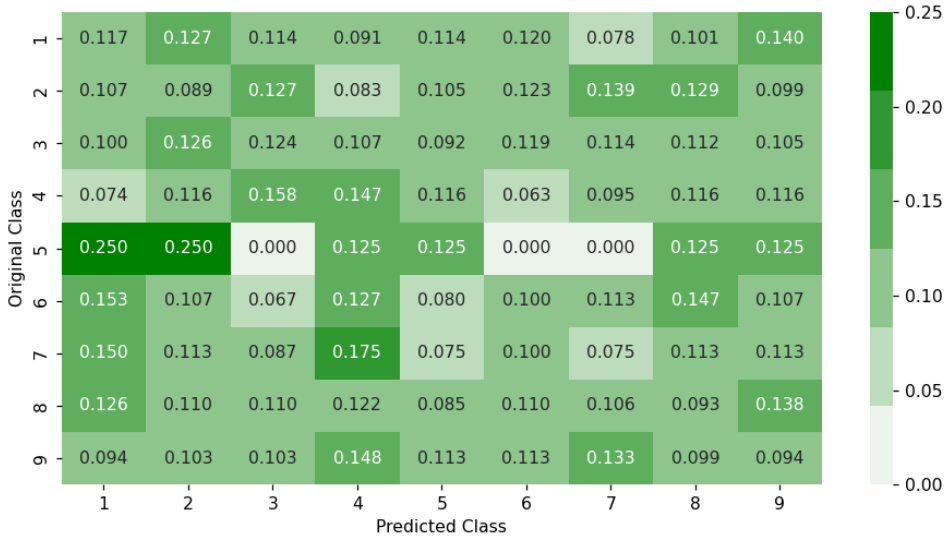



```
-------------------------------------------------- Precision matrix -----------
----------------------------------------
```

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




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

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



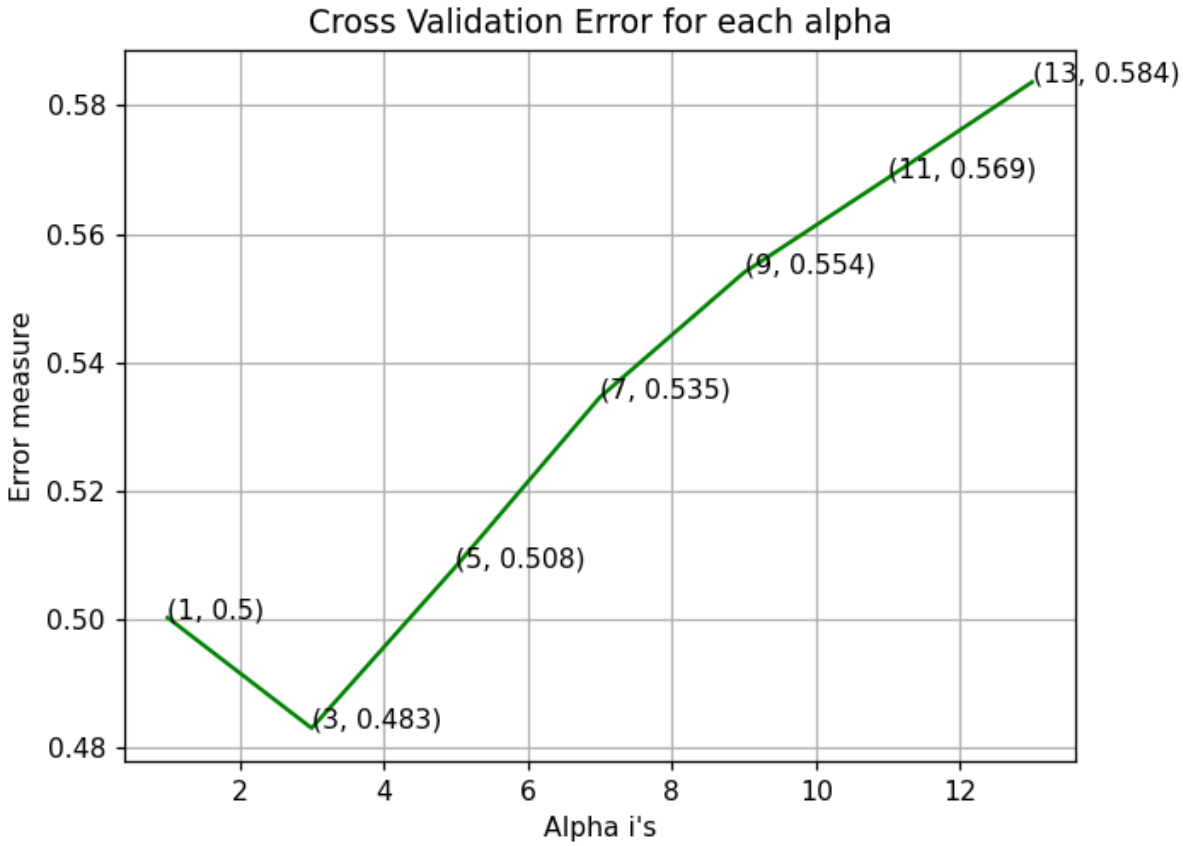

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.2. K Nearest Neighbour Classification

In [ ]:

```
%%time
plt.close()
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', l
# 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-onlin
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/sta
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='si
#
# 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,n_jobs=-1)
    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

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()
```

```
log_loss for k =  1 is 0.5002143528792724
log_loss for k =  3 is 0.48298514961697087
log_loss for k =  5 is 0.5081660747246044
log_loss for k =  7 is 0.5345306597477447
log_loss for k =  9 is 0.5539628167115949
log_loss for k =  11 is 0.5687076842810526
log_loss for k =  13 is 0.583647730251337

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

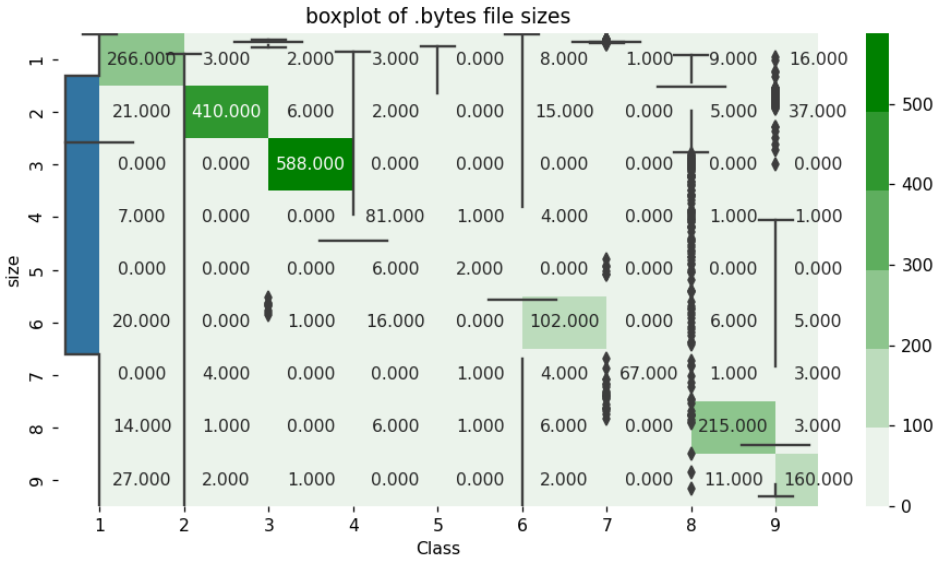



```
Wall time: 17.3 s
```

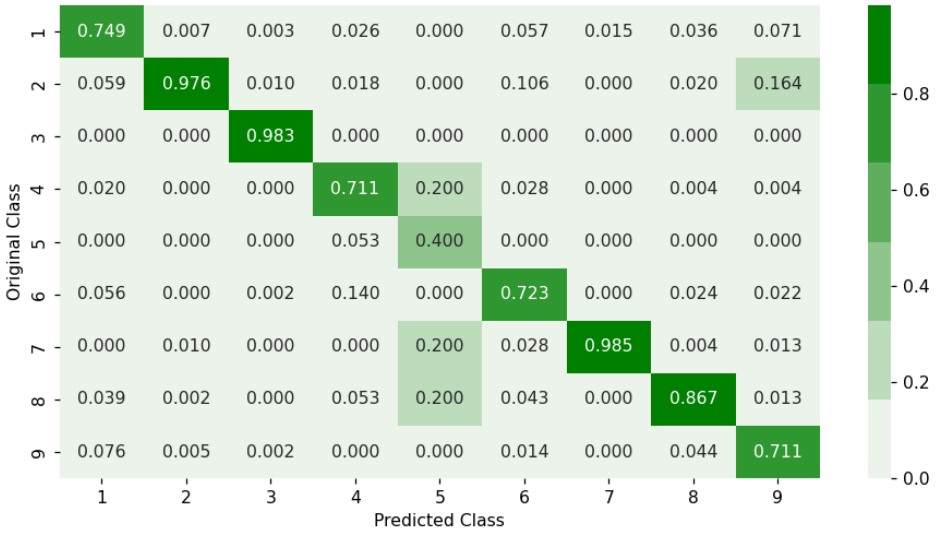
In [ ]:

```
# %%time
plt.close()
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
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
For values of best alpha =  3 The train log loss is: 0.2537021671455636
For values of best alpha =  3 The cross validation log loss is: 0.4829851496169
7087
For values of best alpha =  3 The test log loss is: 0.47508072294405124
Number of misclassified points  13.017479300827967
---------------------------------------------------- Confusion matrix -----------
--------------------------------------

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

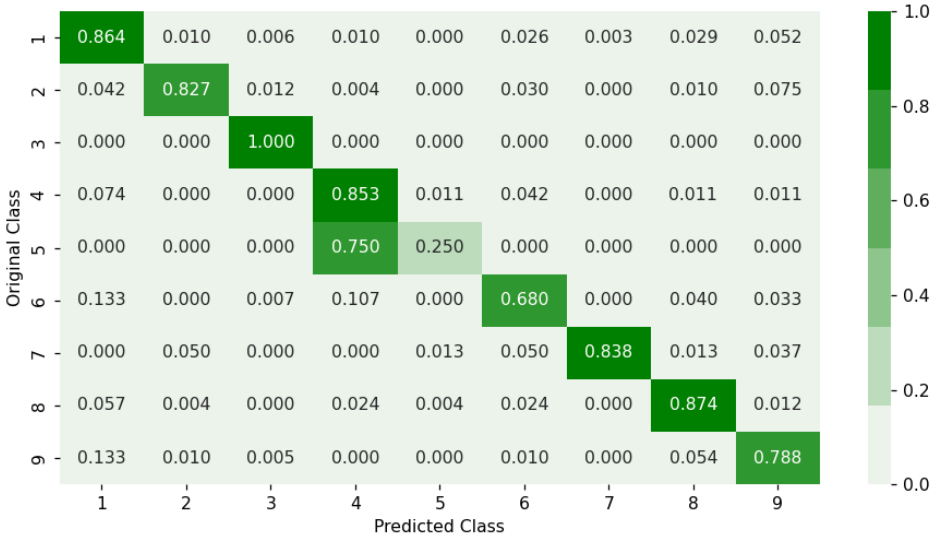



```
---------------------------------------------------- Precision matrix -----------
--------------------------------------

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




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

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



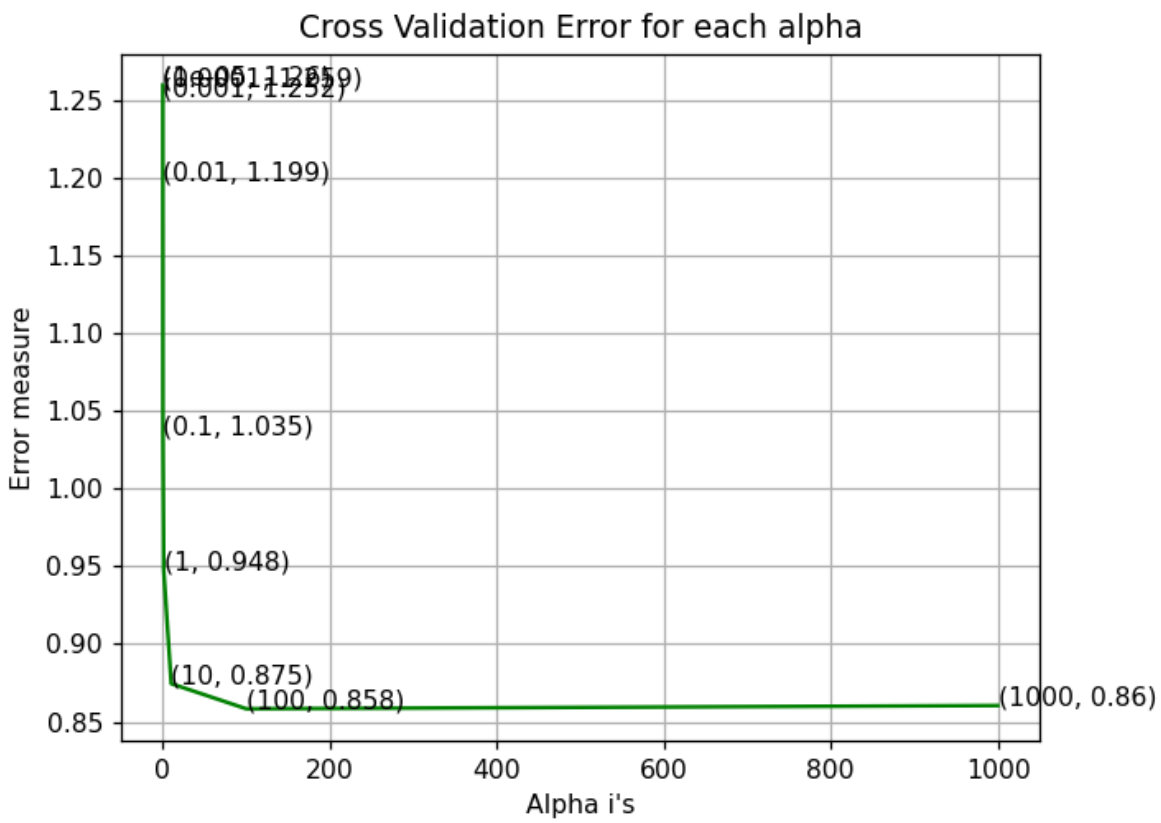

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.3. Logistic Regression

In [ ]:

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

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

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.cla

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()
```

```
log_loss for c =  1e-05 is 1.2596945203803422
log_loss for c =  0.0001 is 1.259300480278056
log_loss for c =  0.001 is 1.2523850219184203
log_loss for c =  0.01 is 1.1985635000231347
log_loss for c =  0.1 is 1.0349755546187265
log_loss for c =  1 is 0.947823196552623
log_loss for c =  10 is 0.8745191192859392
log_loss for c =  100 is 0.8583275102454776
log_loss for c =  1000 is 0.8604214263833574

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



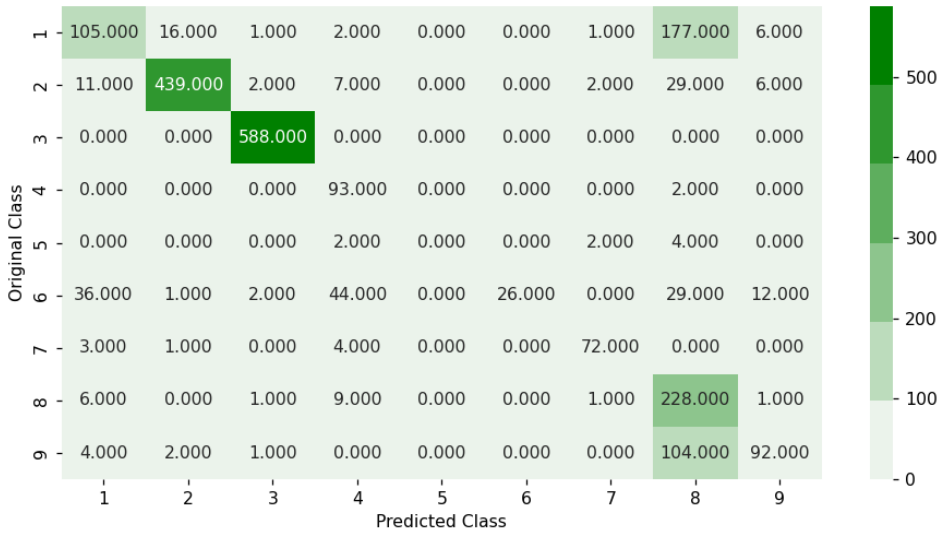

```
Wall time: 20.1 s
```

In [ ]:

```
%%time
plt.close()
logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='
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=logisti
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.cla
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

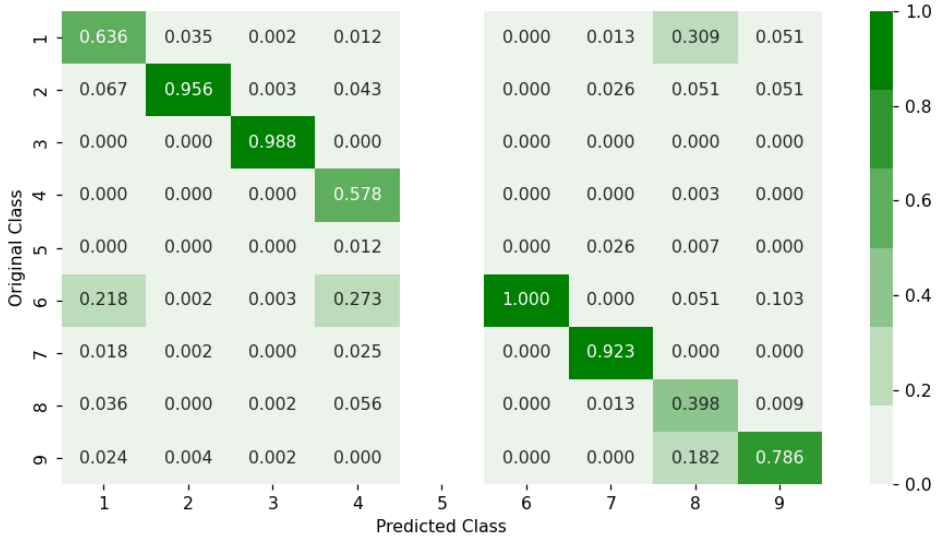
```
log loss for train data 0.8615268625525035
log loss for cv data 0.8583275102454776
log loss for test data 0.8563175051825581
Number of misclassified points  24.42502299908004
---------------------------------------------------- Confusion matrix -----------
--------------------------------------
```

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



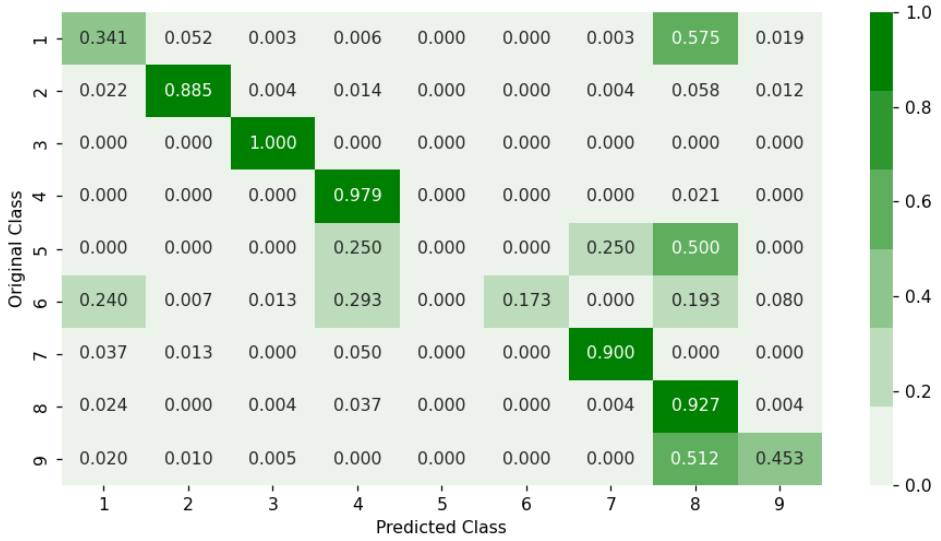

```
---------------------------------------------------- Precision matrix -----------
--------------------------------------
```

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




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

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




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Wall time: 4.28 s
```

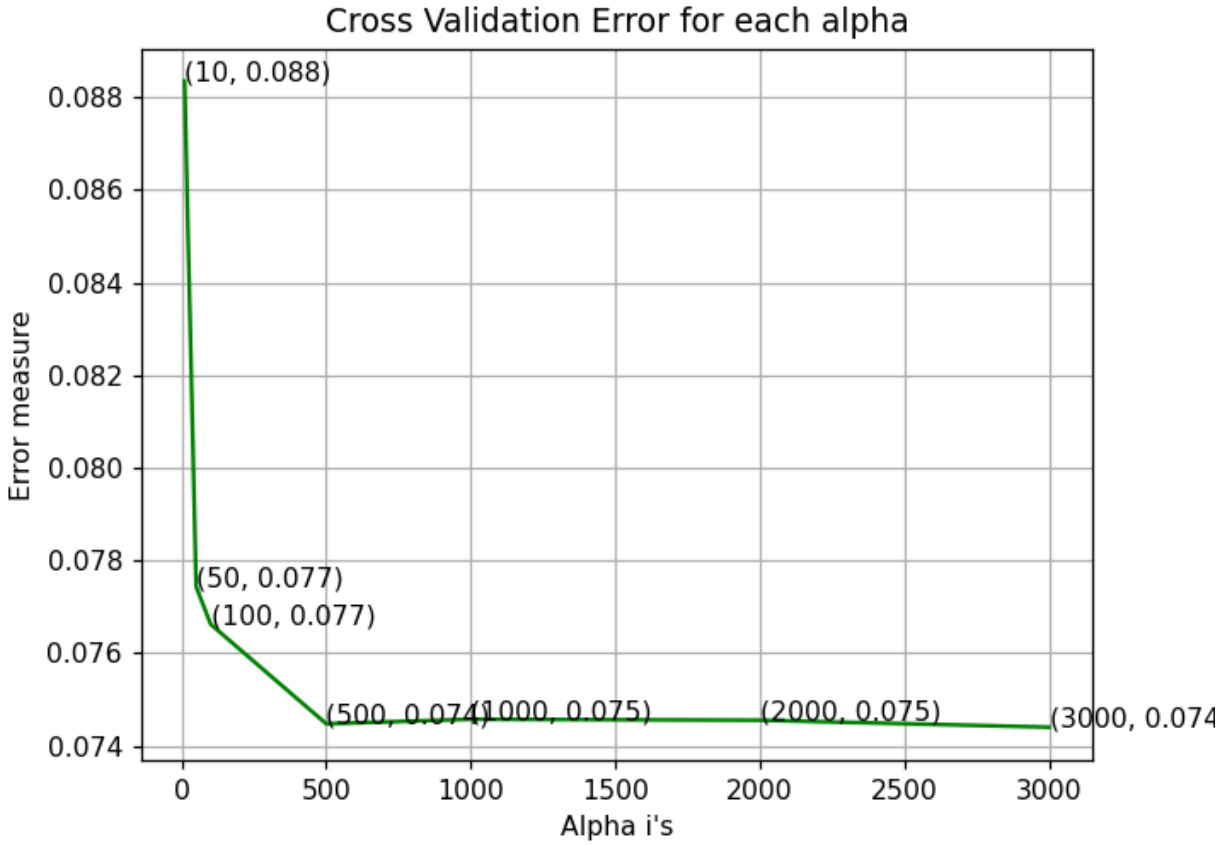
4.1.4. Random Forest Classifier

In [ ]:

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# --------------------------------

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

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()
```

```
log_loss for c =  10 is 0.08835471979796837
log_loss for c =  50 is 0.07743620199725376
log_loss for c =  100 is 0.07662447196390508
log_loss for c =  500 is 0.07447383968731333
```

```
log_loss for c =  1000 is 0.07457217724631218
log_loss for c =  2000 is 0.07455069132208691
log_loss for c =  3000 is 0.07440057924615005

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

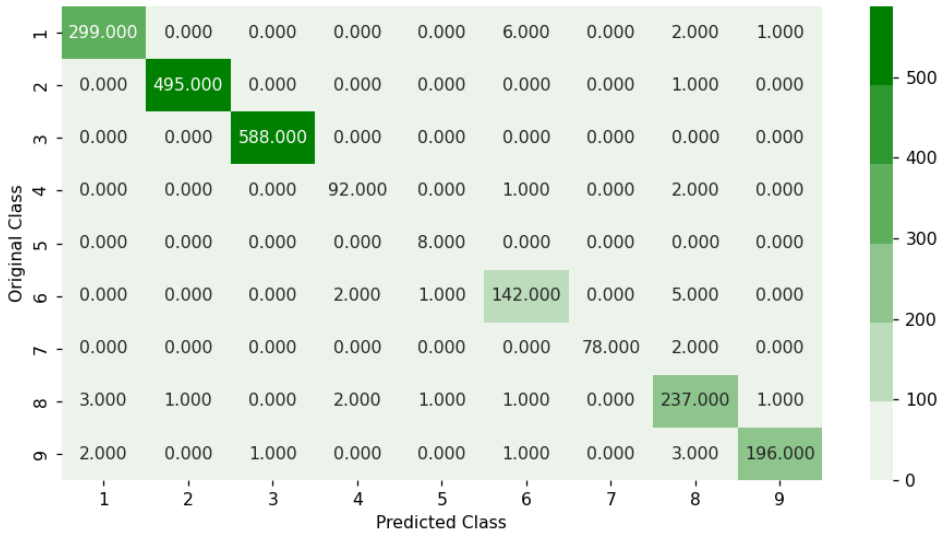



```
Wall time: 6min 21s
```

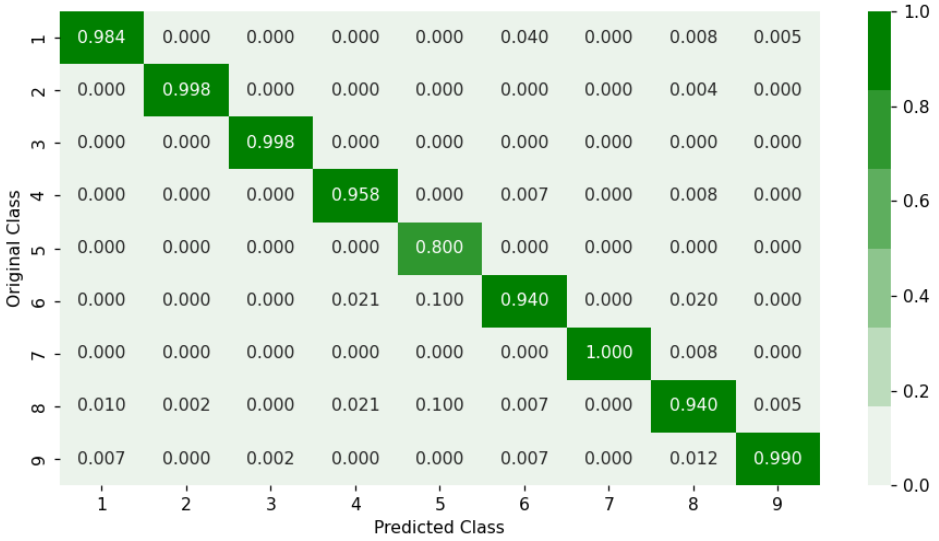
In [ ]:

```
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
For values of best alpha =  3000 The train log loss is: 0.025936967738833847
For values of best alpha =  3000 The cross validation log loss is: 0.0744005792
4615005
For values of best alpha =  3000 The test log loss is: 0.07556461381685227
Number of misclassified points  1.7939282428702852
-------------------------------------------------- Confusion matrix -----------
---------------------------------------

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

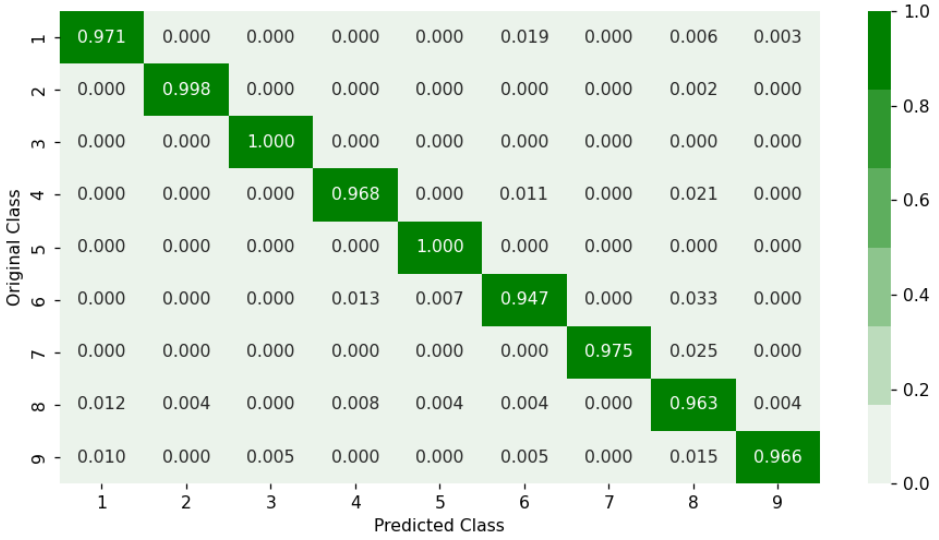



```
-------------------------------------------------- Precision matrix -----------
---------------------------------------

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




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

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




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

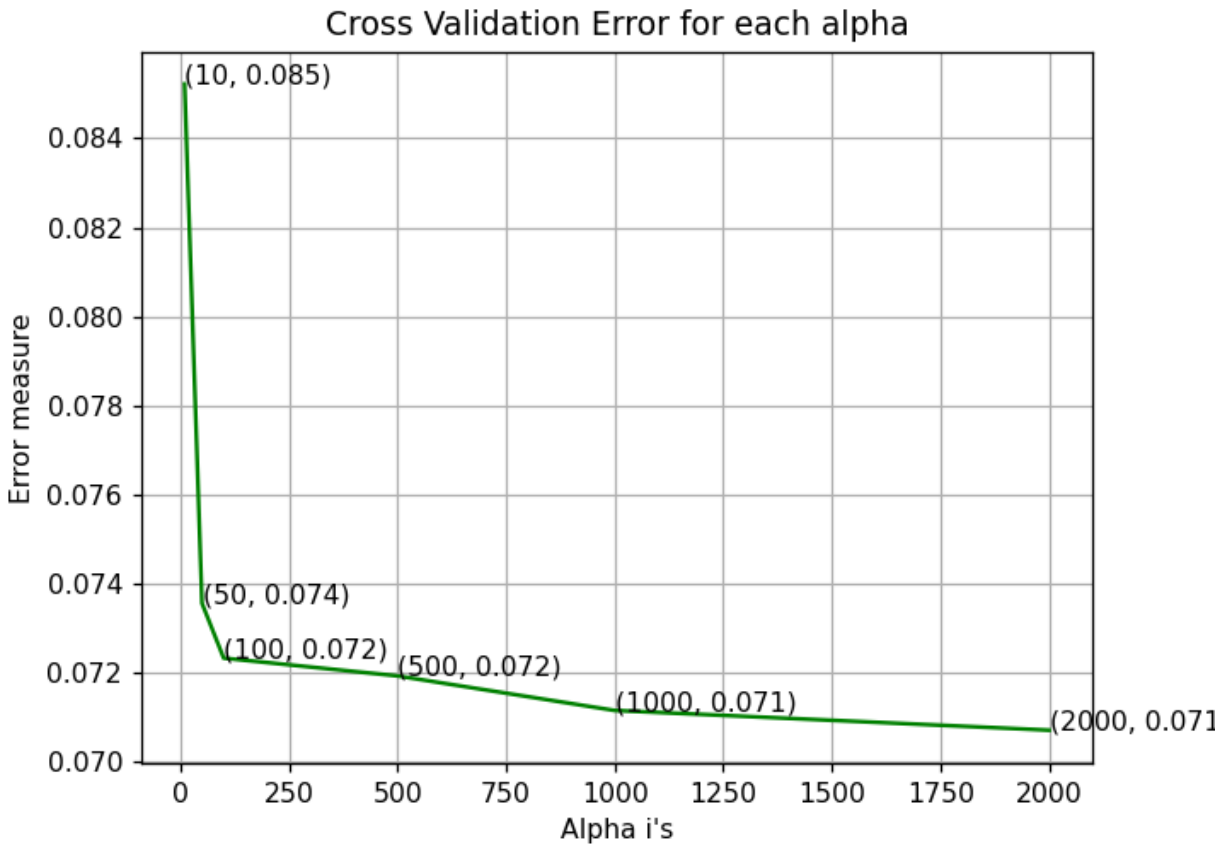
4.1.5. XgBoost Classification

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-onli
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# ----------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1,eval_metric='merror')
    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

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()
```

```
log_loss for c =  10 is 0.08521935703009392
log_loss for c =  50 is 0.07356860014225576
log_loss for c =  100 is 0.0723360105588515
log_loss for c =  500 is 0.071936595464993
log_loss for c =  1000 is 0.07116122467854873
log_loss for c =  2000 is 0.07071886205835459
```

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




```
Wall time: 18min 58s
```

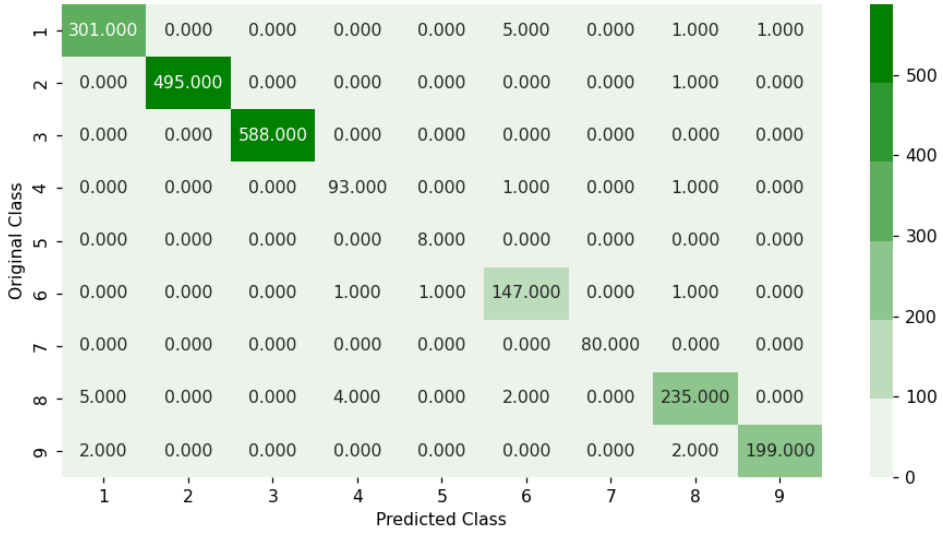
In [ ]:

```
%%time
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
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
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

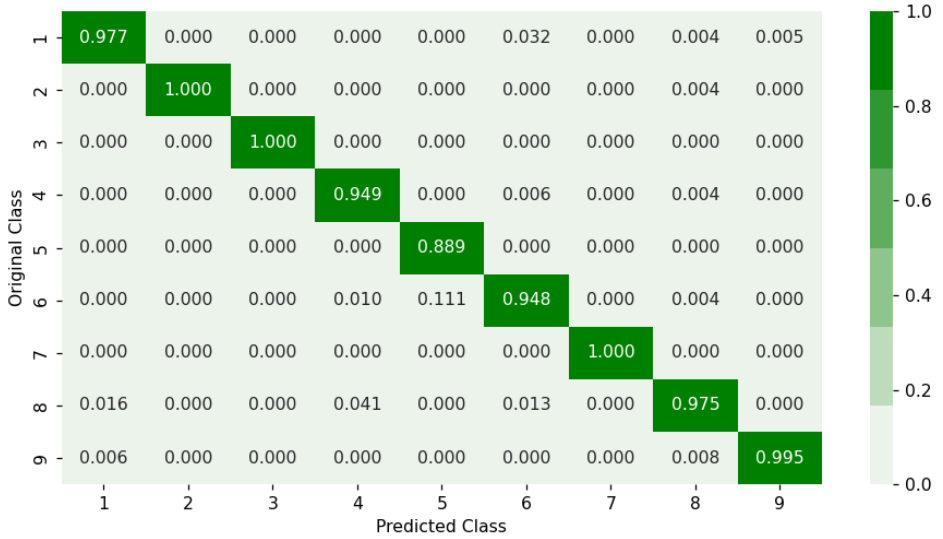
```
For values of best alpha =  2000 The train log loss is: 0.02149461592509416
For values of best alpha =  2000 The cross validation log loss is: 0.0707188620
5835459
For values of best alpha =  2000 The test log loss is: 0.06757843531946935
Wall time: 8min 31s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

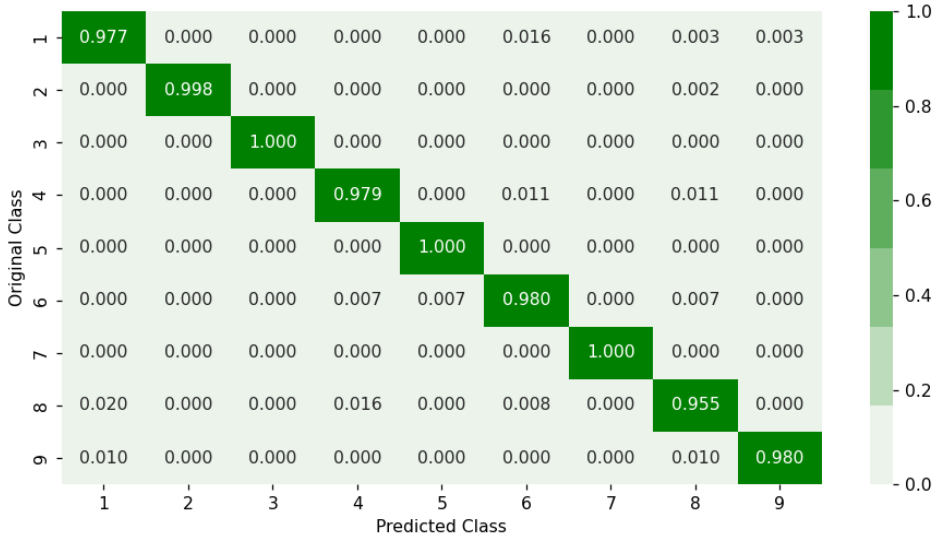
```
Number of misclassified points  1.2879484820607177
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [ ]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuni
x_cfl=XGBClassifier(n_threads=-1,eval_metric='merror')

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_
random_cfl1.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[13:23:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:576:
Parameters: { "n_threads" } might not be used.

  This could be a false alarm, with some parameters getting used by language bi
ndings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such c
ases.
```

Out[33]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=10)
```

In [ ]:
```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.2, 'co
lsample_bytree': 0.3}
```

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

x_cfl=XGBClassifier(n_estimators=random_cfl1.best_params_['n_estimators'], l
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

```
train loss 0.020861953249744387
cv loss 0.06650402634081588
test loss 0.06588410810995225
```

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls

6. APIs
With the help of parallel processing we extracted all the features.In parallel we can use all the cores that are present in our computer.

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

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

4.2.1 Feature extraction from asm files

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

In [ ]:

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

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

In [ ]:

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

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

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdat
    #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', 's
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registe
    #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
            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' se
                        if registers[i] in li and ('text' in l or 'CODE' in
                            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:','.rdat
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 's
    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
            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
                            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:','.rdat
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 's
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('thrid/'+f,encoding='cp1252',errors ='replace') as
            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
                            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:','.rdat
```

```
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 's
    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
            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
                            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:','.rdat
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 's
    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
            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
                            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 [ ]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain
# 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[37]:

	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 [ ]:

```
#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=35615717
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=15
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file
    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())
```

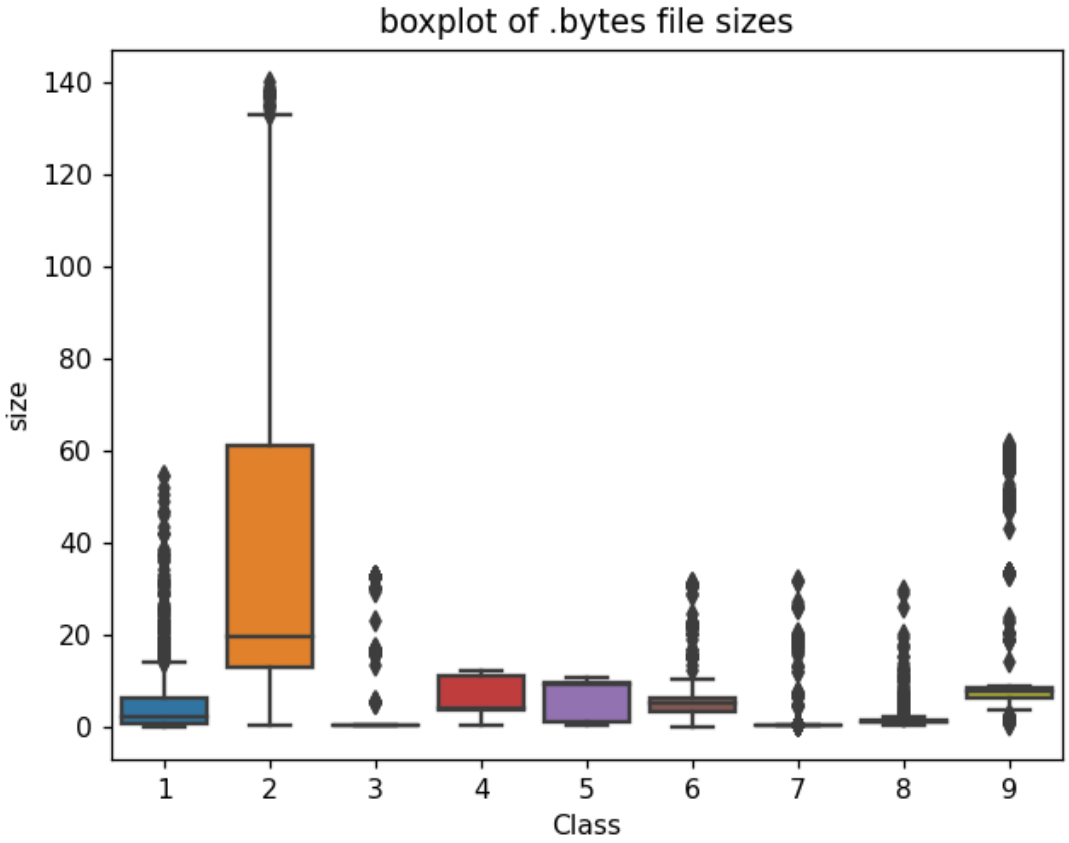
```
                     ID       size  Class
0  01azqd4InC7m9JpocGv5  56.229886      9
1  01IsoiSMh5gxyDYTl4CB  13.999378      2
2  01jsnpXSAlgw6aPeDxrU   8.507785      9
3  01kcPWA9K2BOxQeS5Rju   0.078190      1
4  01SuzwMJEIXsK7A8dQbl   0.996723      8
```

4.2.1.2 Distribution of .asm file sizes

In [ ]:

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

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



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

```
(10868, 54)
(10868, 3)
```

Out[44]:

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

5 rows × 55 columns

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

Out[45]:

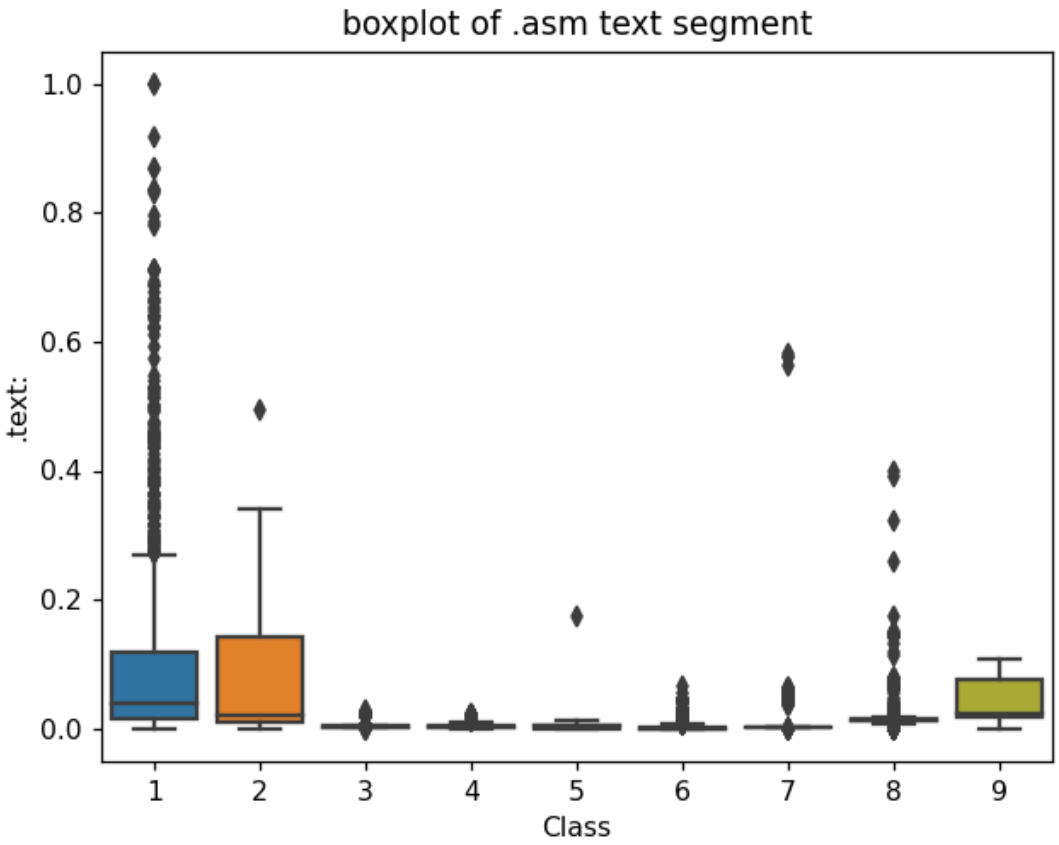
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.(
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.(
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.(
3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

5 rows × 55 columns

4.2.2 Univariate analysis on asm file features

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

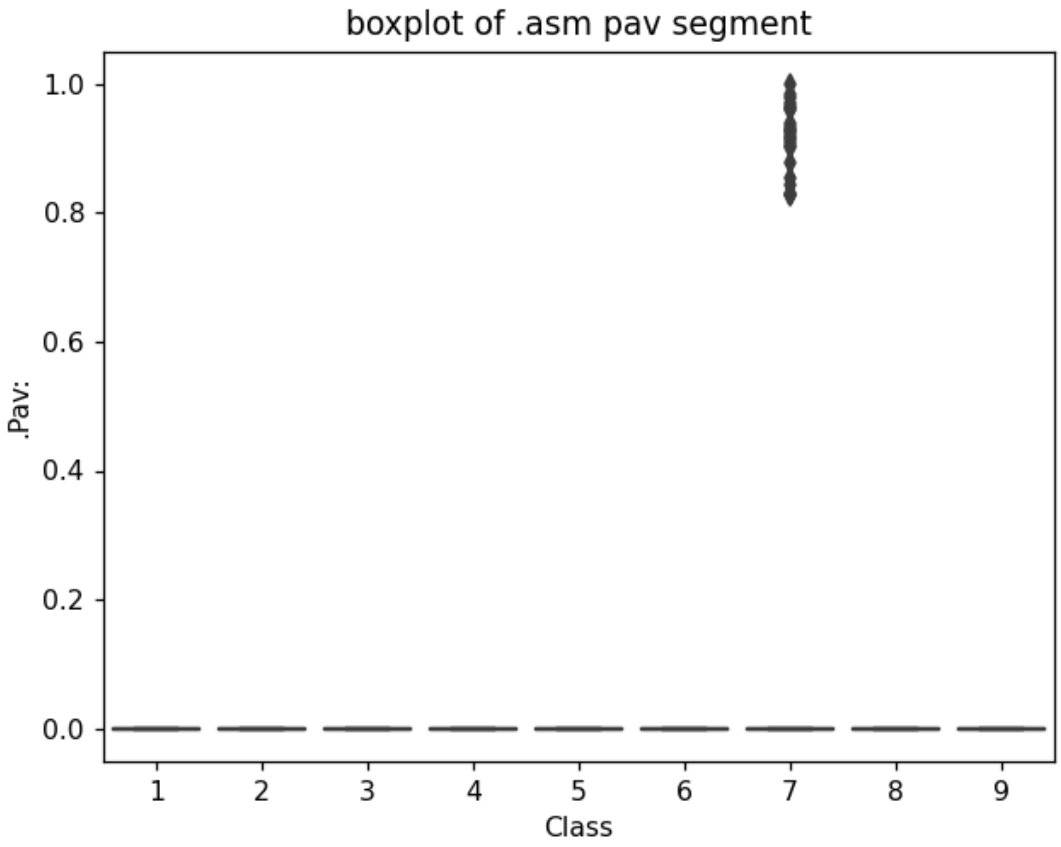
```
<IPython.core.display.Javascript object>
```



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

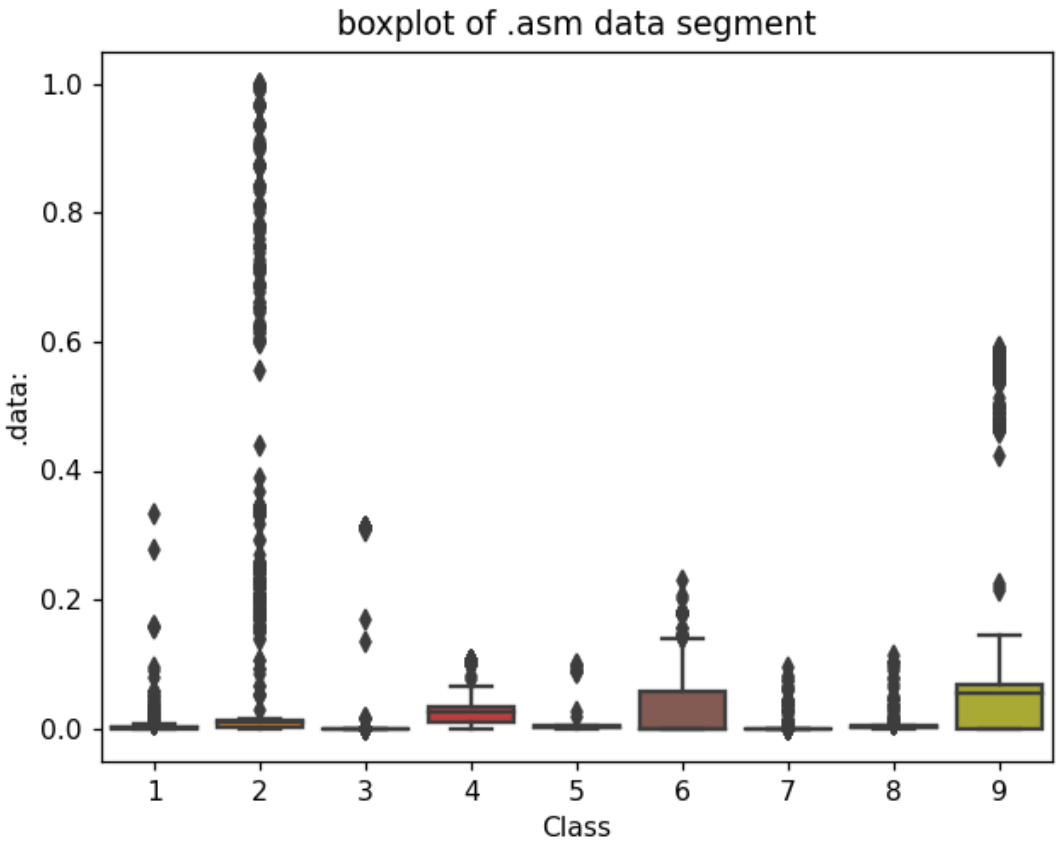
In [ ]:
```
plt.close()
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```

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



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

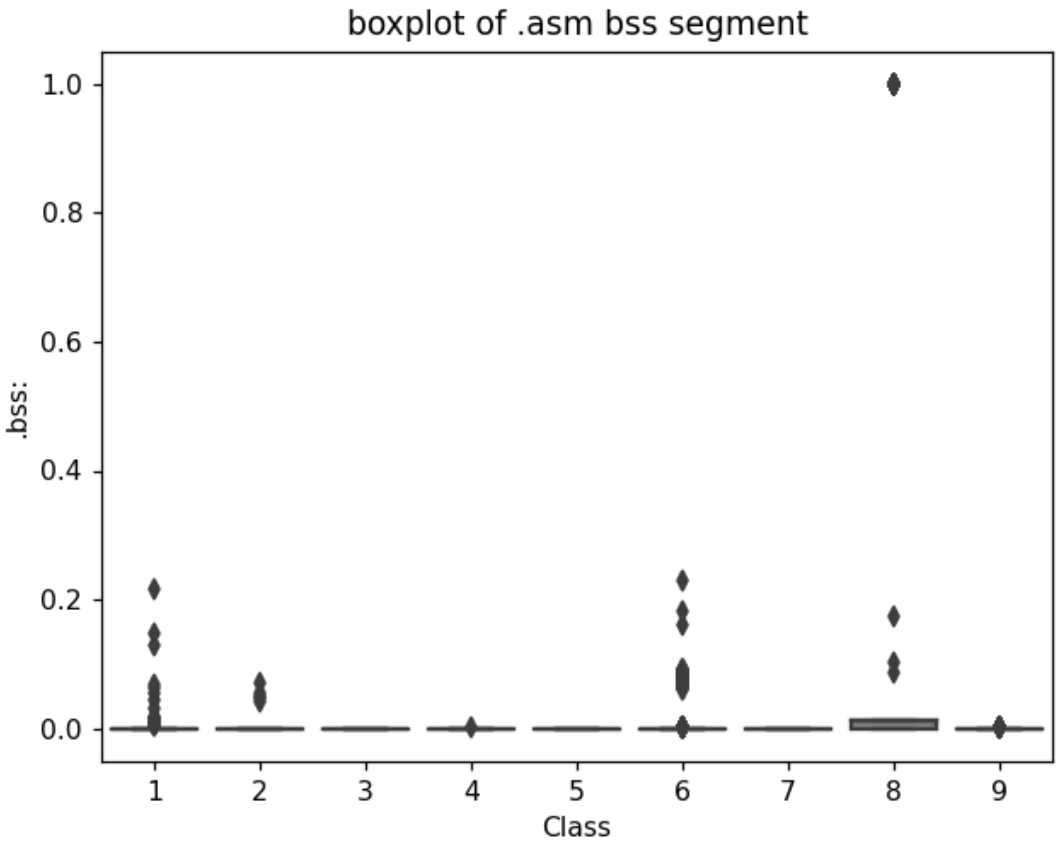
<IPython.core.display.Javascript object>



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

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

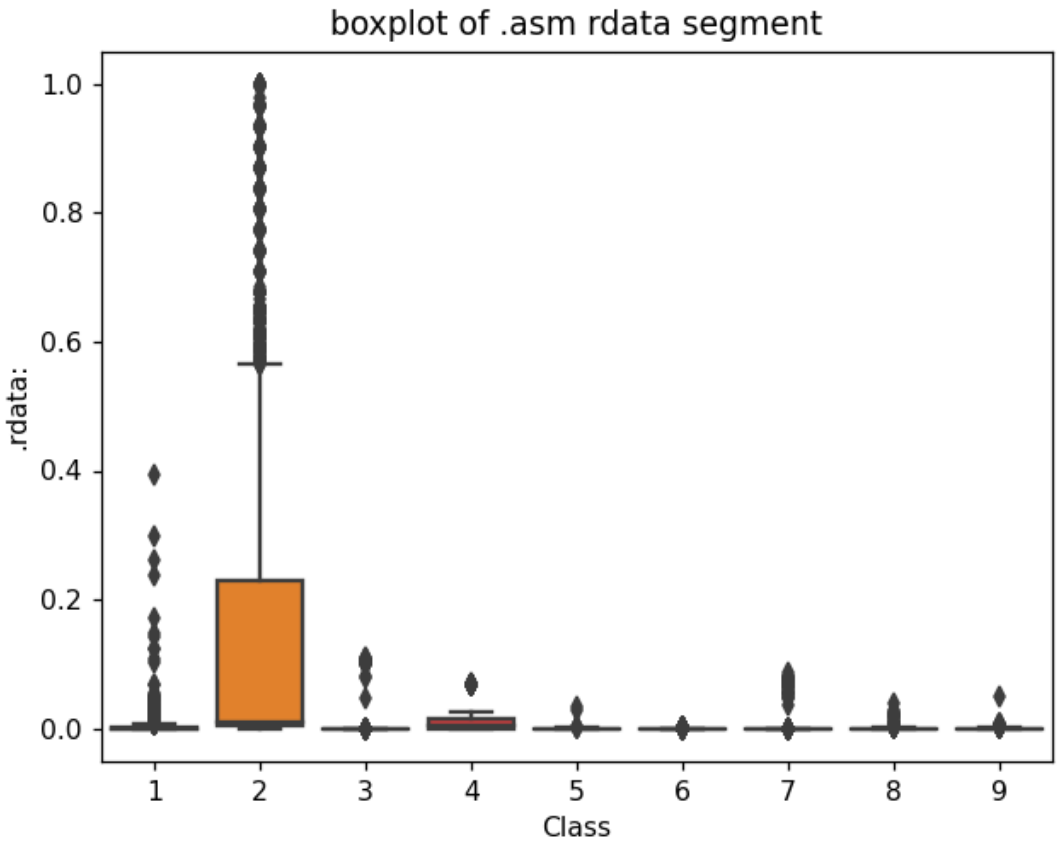
```
<IPython.core.display.Javascript object>
```



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

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

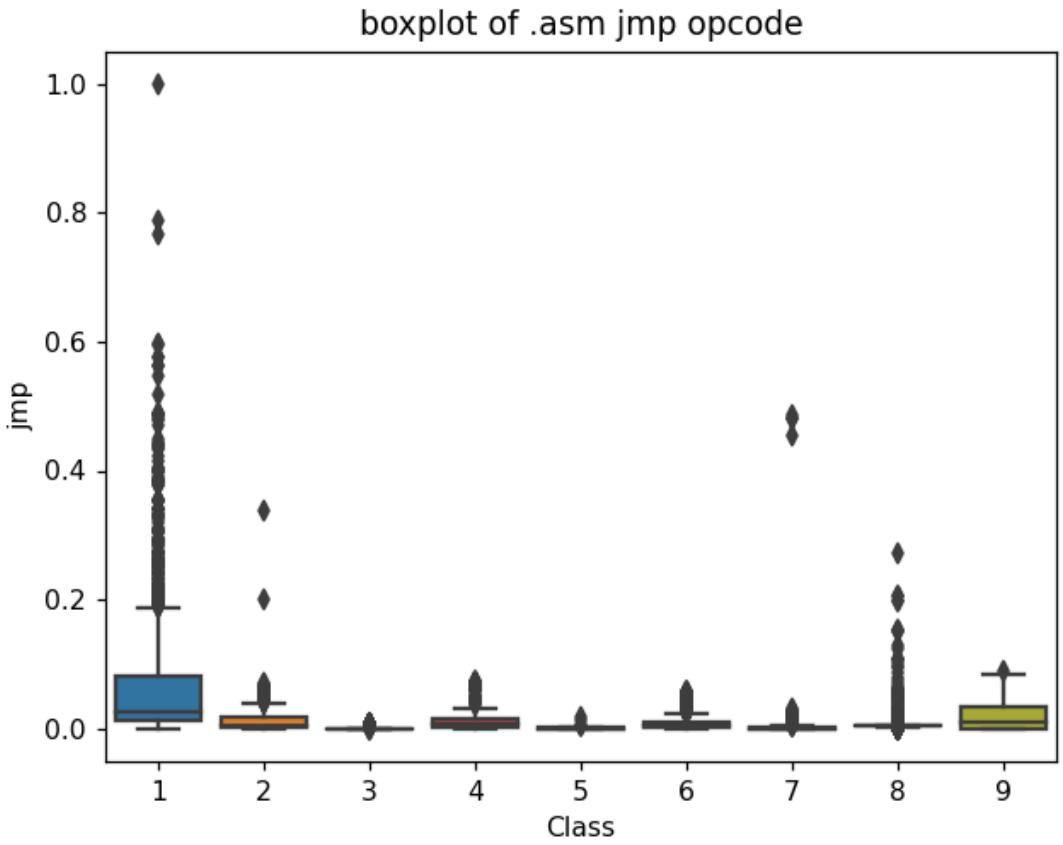
<IPython.core.display.Javascript object>



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

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

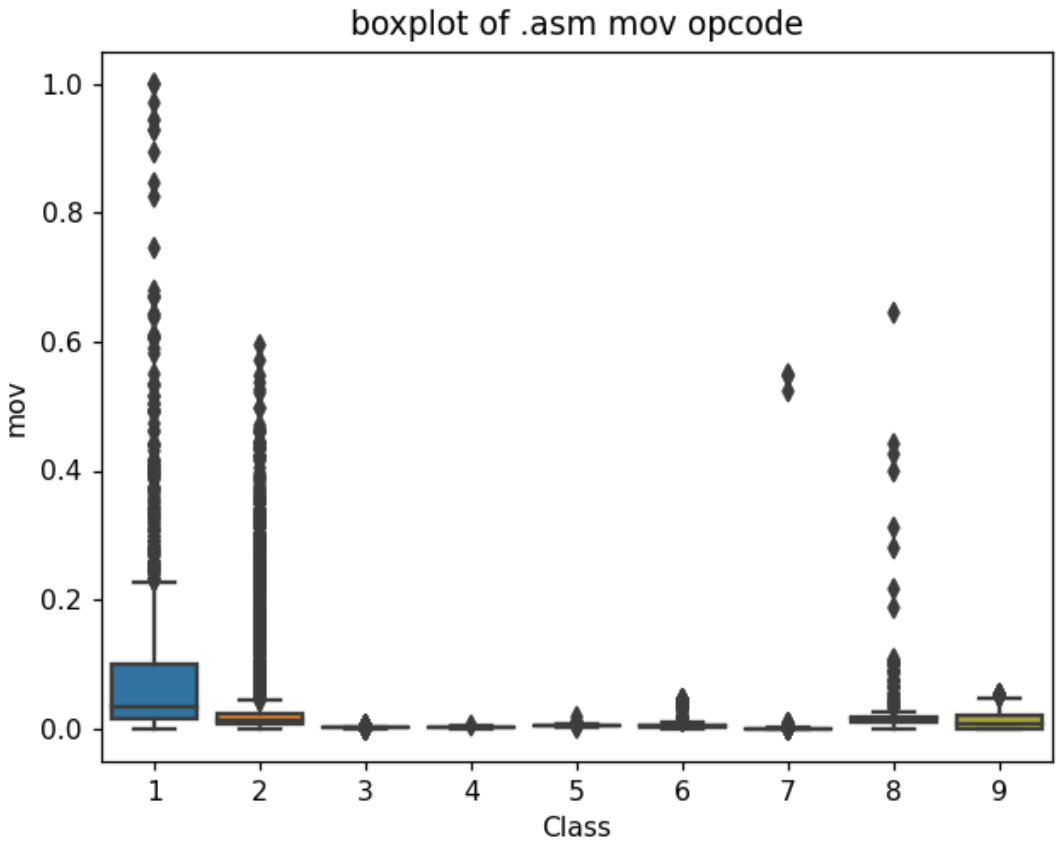
```
<IPython.core.display.Javascript object>
```



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

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

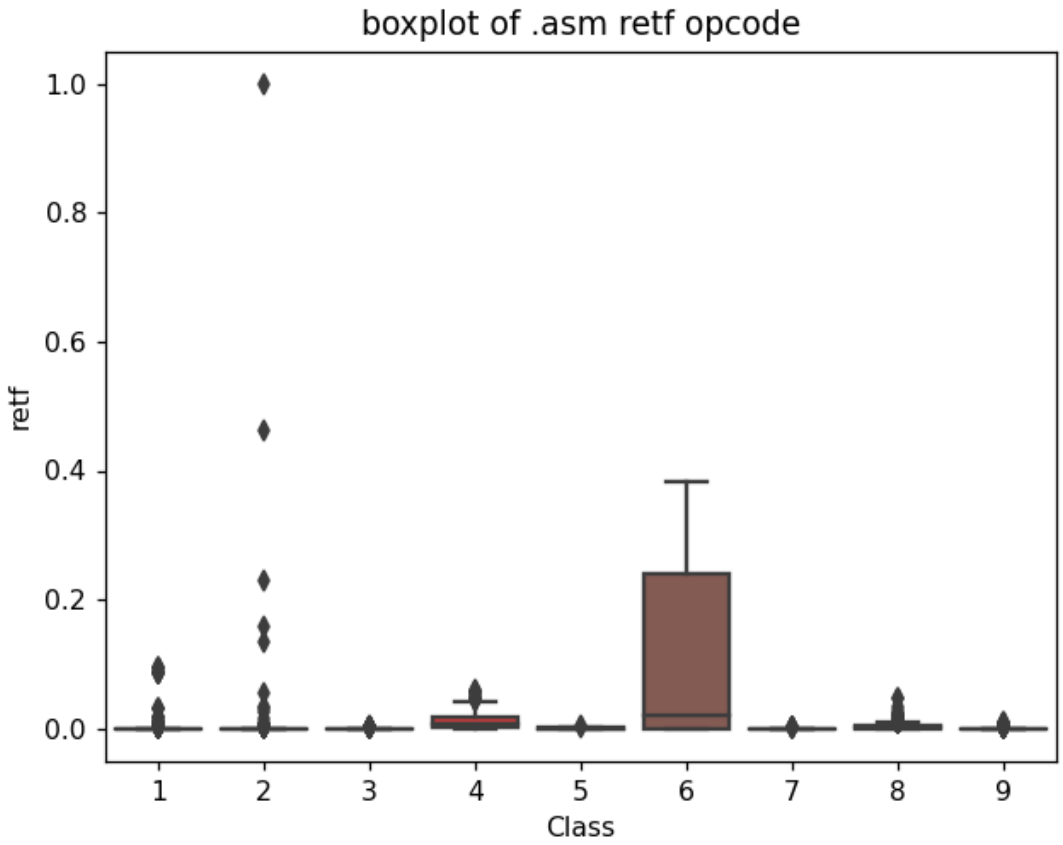
```
<IPython.core.display.Javascript object>
```



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

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

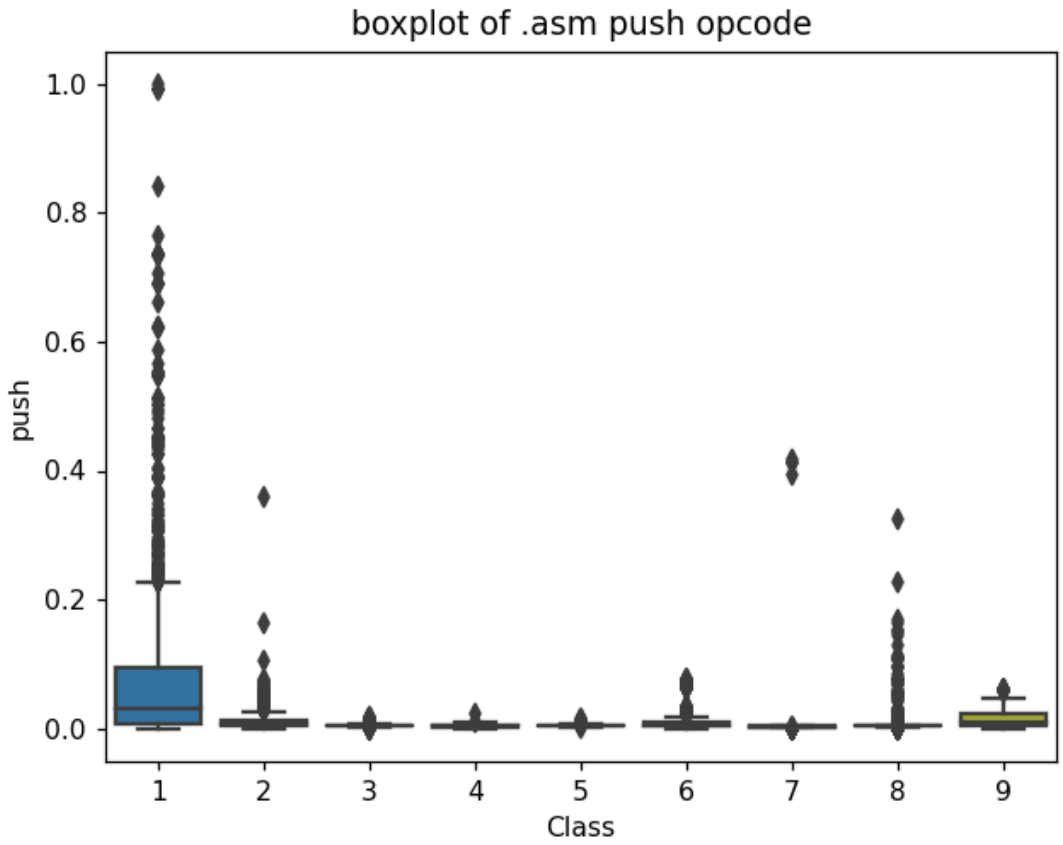
```
<IPython.core.display.Javascript object>
```



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

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

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



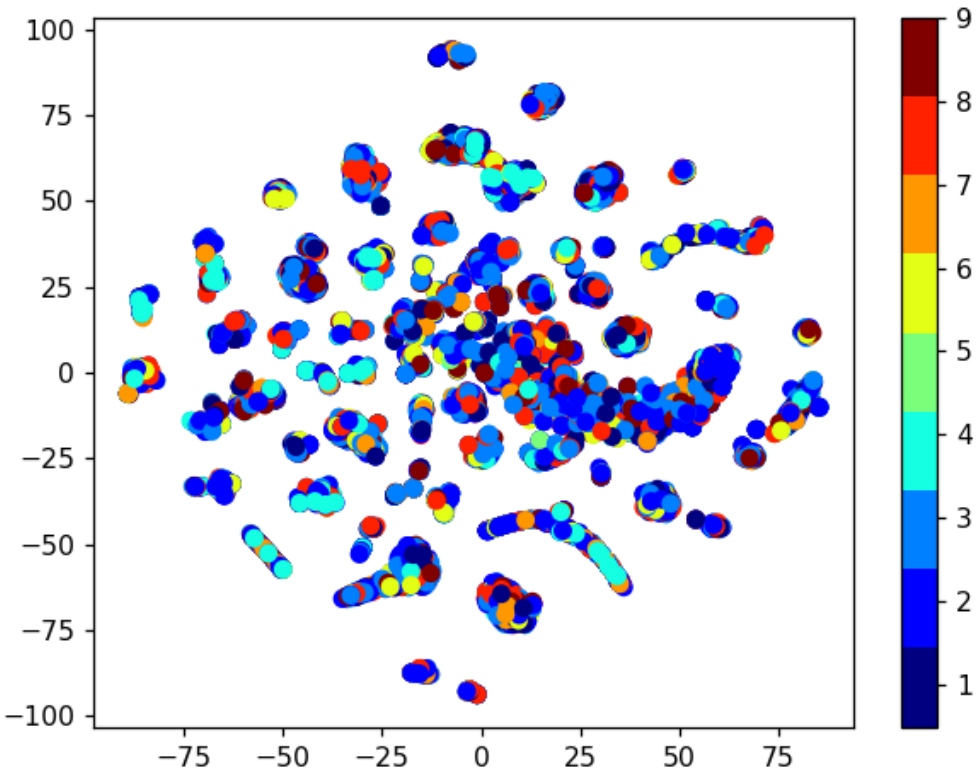

```
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 100
0
```

4.2.2 Multivariate Analysis on .asm file features

In [ ]:

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

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

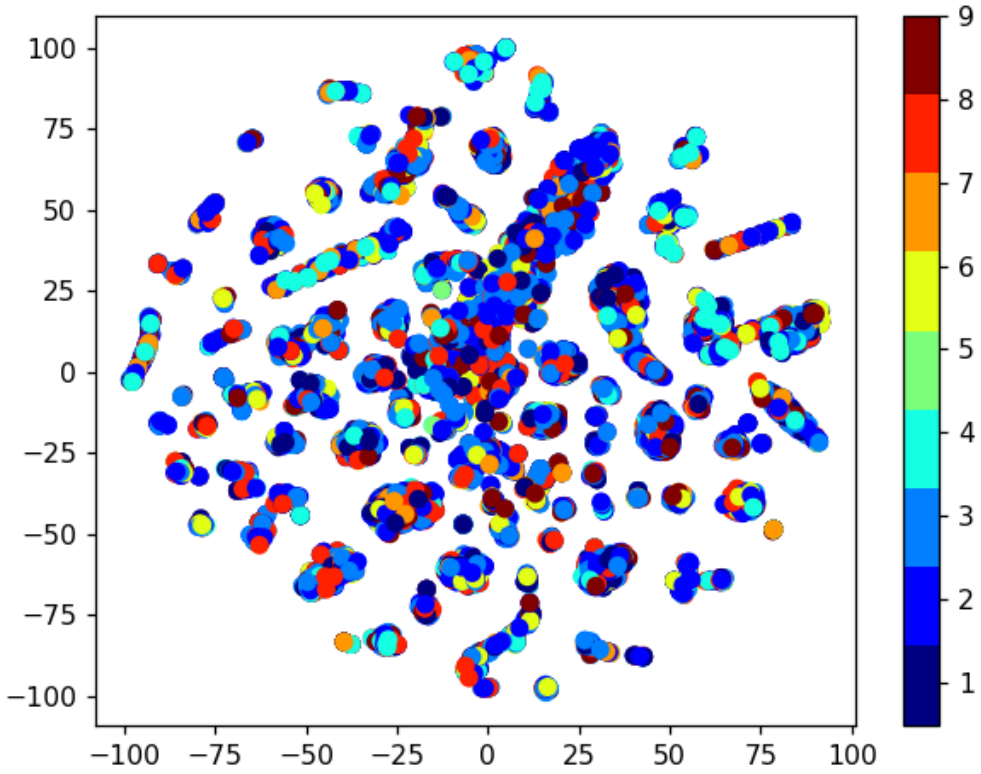
```
<IPython.core.display.Javascript object>
```



In [ ]:

```
plt.close()
# by univariate analysis on the .asm file features we are getting very negli
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analy
# the plot looks very messy

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

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



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

- We have taken only 52 features from asm files (after reading through many blogs and research papers)
- The univariate analysis was done only on few important features.
- Take-aways

- 1. Class 3 can be easily separated because of the frequency of segments,opcodes and keywords being less
- 2. Each feature has its unique importance in separating the Class labels.

4.3 Train and test split

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

In [ ]:
```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,as
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm,
```

In [ ]:
```
1 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_x     False
size_y     False
dtype: bool
```

4.4. Machine Learning models on features of .asm

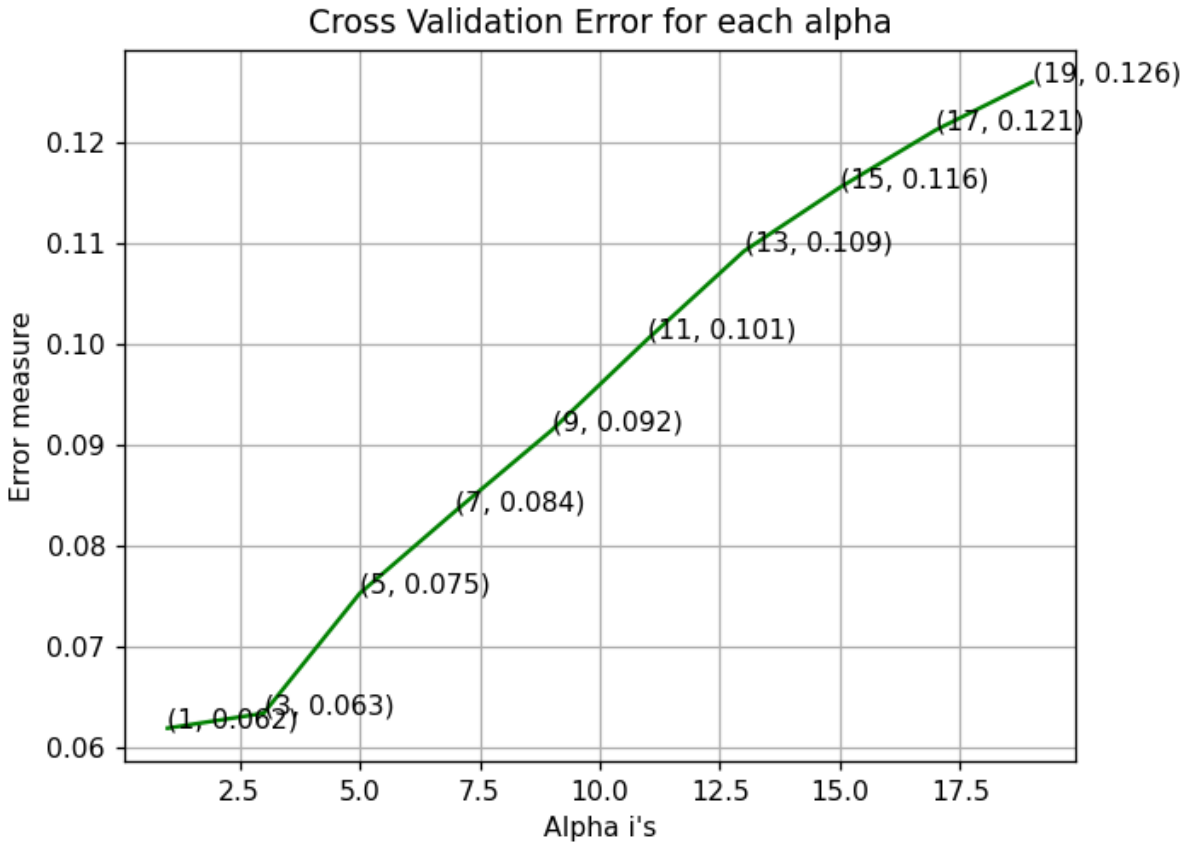
files

4.4.1 K-Nearest Neigbors

In [ ]:

```
%%time
plt.close()
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', l
# 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-onlin
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/sta
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='si
#
# 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,n_jobs=-1)
    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.cla

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()
```

```
log_loss for k =  1 is 0.06196307446768273
log_loss for k =  3 is 0.0634370558192665
log_loss for k =  5 is 0.07532813696091004
log_loss for k =  7 is 0.08352849459591188
log_loss for k =  9 is 0.09151963799438398
log_loss for k =  11 is 0.10057694623548089
log_loss for k =  13 is 0.10927147427247862
log_loss for k =  15 is 0.11559634126175941
log_loss for k =  17 is 0.12130447147461845
log_loss for k =  19 is 0.12604700619828274

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

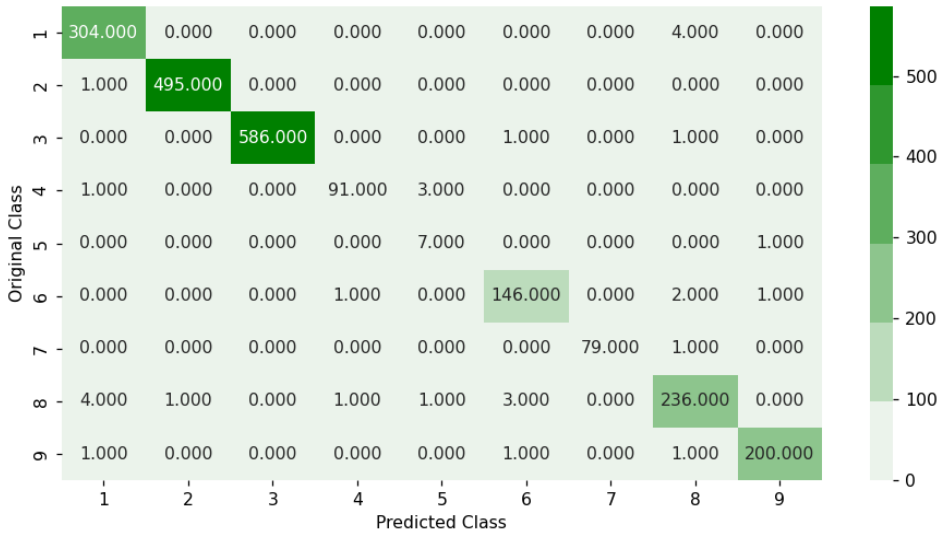



```
Wall time: 23.5 s
```

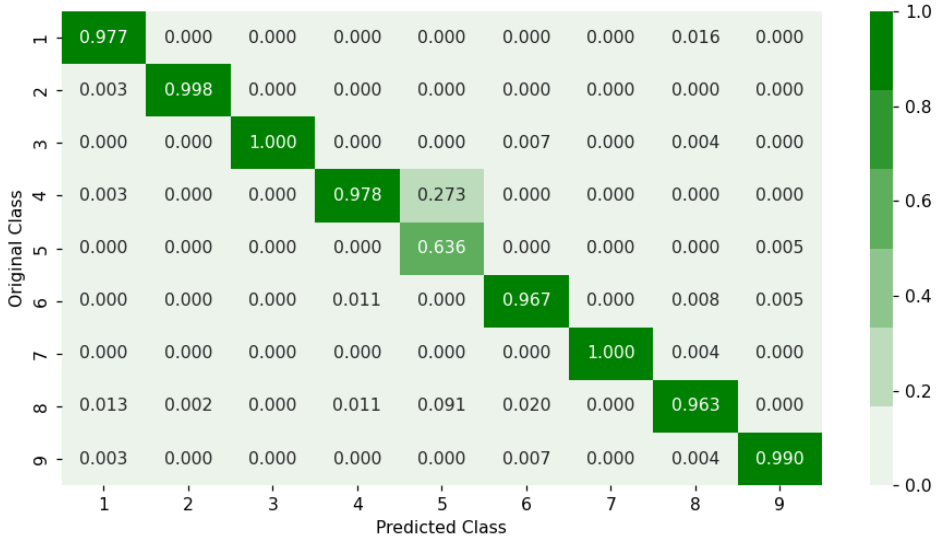
In [ ]:

```
%%time
plt.close()
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))
```

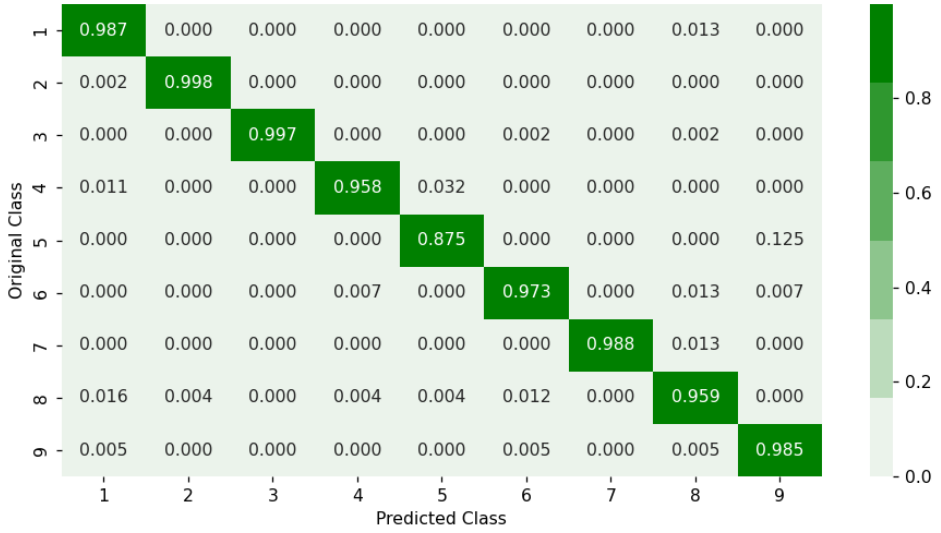
```
log loss for train data 0.028080753461076014
log loss for cv data 0.06196307446768273
log loss for test data 0.09035187549912337
Number of misclassified points  1.3799448022079117
---------------------------------------------------- Confusion matrix -----------
--------------------------------------

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




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

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




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

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



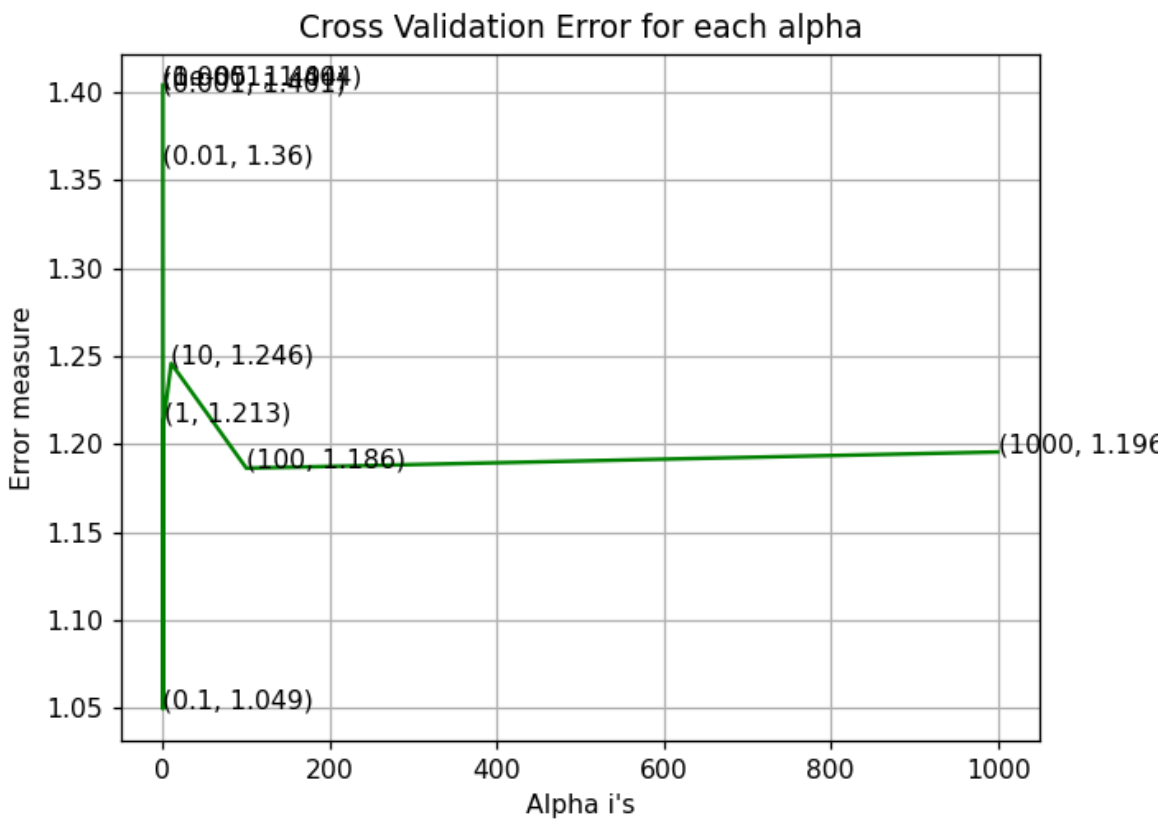

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Wall time: 6.88 s
```

4.4.2 Logistic Regression

In [ ]:

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

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


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

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()
```

```
log_loss for c =  1e-05 is 1.4042018382359298
log_loss for c =  0.0001 is 1.4041160332020937
log_loss for c =  0.001 is 1.4010346346222686
log_loss for c =  0.01 is 1.3597556190135878
log_loss for c =  0.1 is 1.0494097878786035
log_loss for c =  1 is 1.2131815126874865
log_loss for c =  10 is 1.2458785619041555
log_loss for c =  100 is 1.1862923569419228
log_loss for c =  1000 is 1.1955136795366272

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

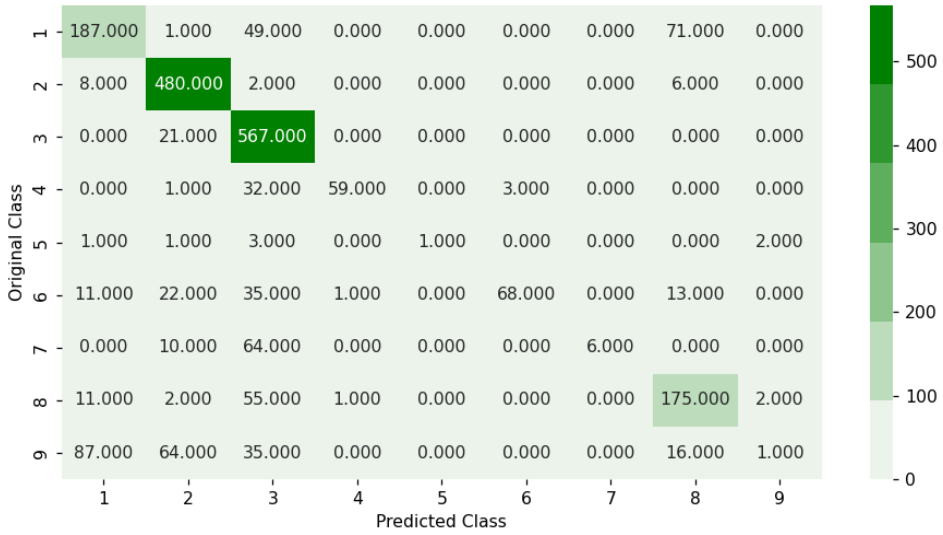



```
Wall time: 12.3 s
```

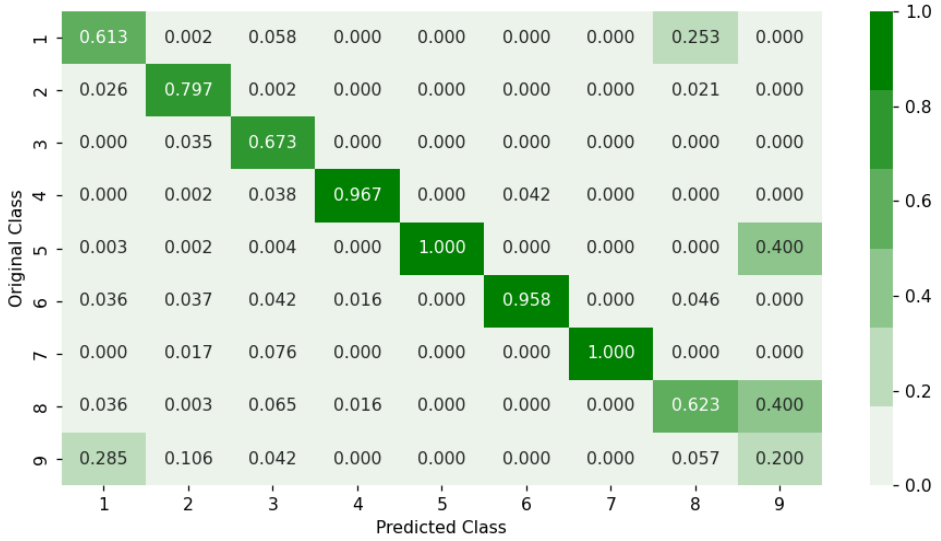
In [ ]:

```
%%time
plt.close()
logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='
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=lo
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logistic
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logi
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

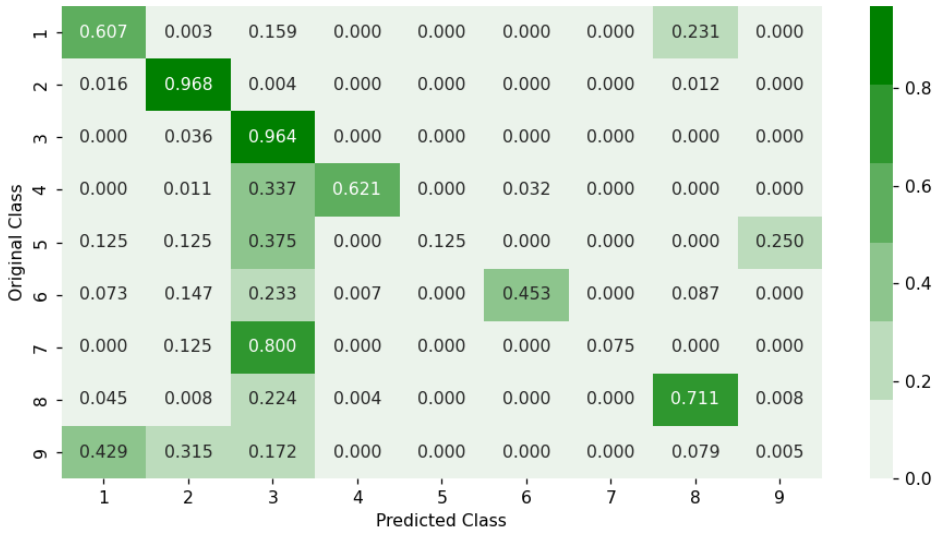
```
log loss for train data 1.03202773035354
log loss for cv data 1.0494097878786035
log loss for test data 1.048671333428269
Number of misclassified points  28.978840846366143
---------------------------------------------------- Confusion matrix -----------
--------------------------------------

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




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

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




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

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




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Wall time: 1.4 s
```

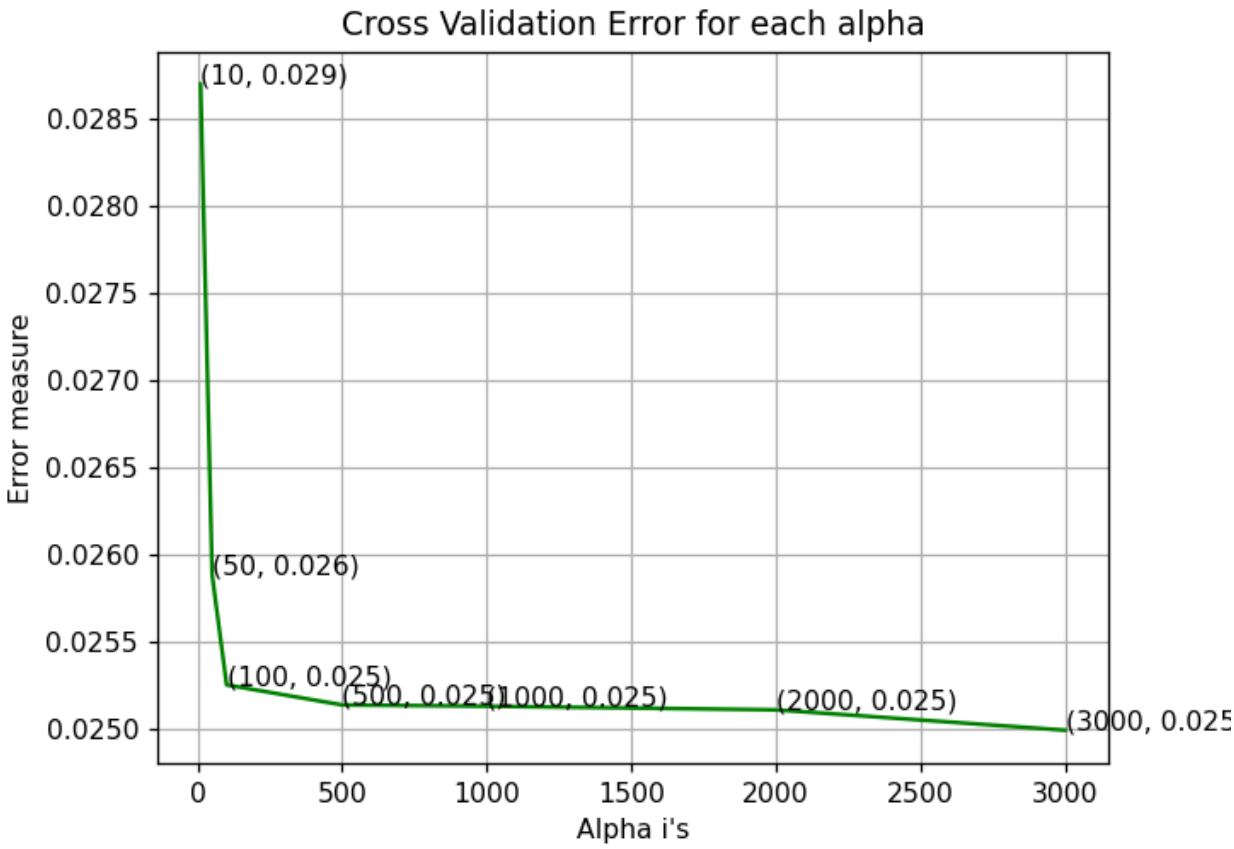
4.4.3 Random Forest Classifier

In [ ]:

```
%%time
plt.close()
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# ---------------------------------

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

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()
```

```
log_loss for c =  10 is 0.02870105266243957
log_loss for c =  50 is 0.02588595952016721
log_loss for c =  100 is 0.025255589487768727
log_loss for c =  500 is 0.025140567951287794
log_loss for c =  1000 is 0.025133463255381086
log_loss for c =  2000 is 0.02511241263920872
log_loss for c =  3000 is 0.024995953922172888
```

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

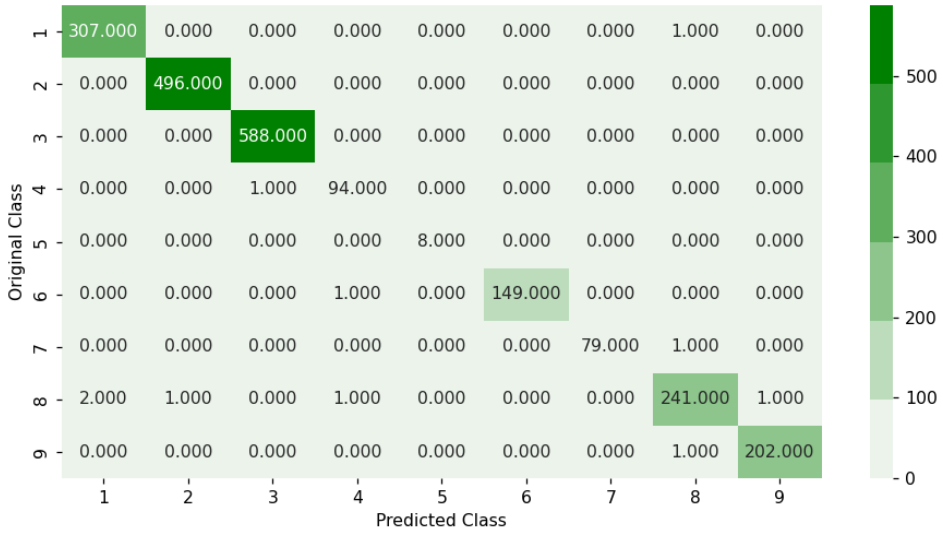



```
Wall time: 2min 5s
```

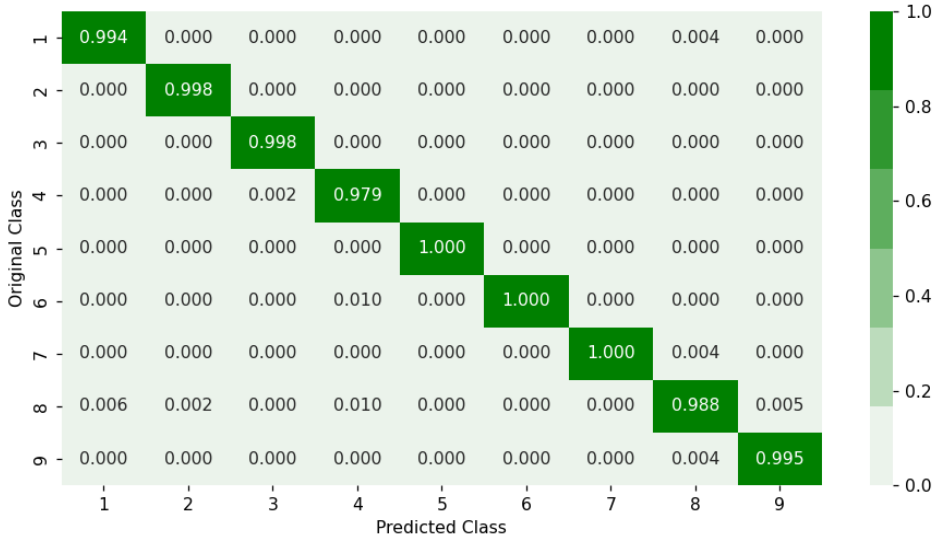
In [ ]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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=si
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.
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

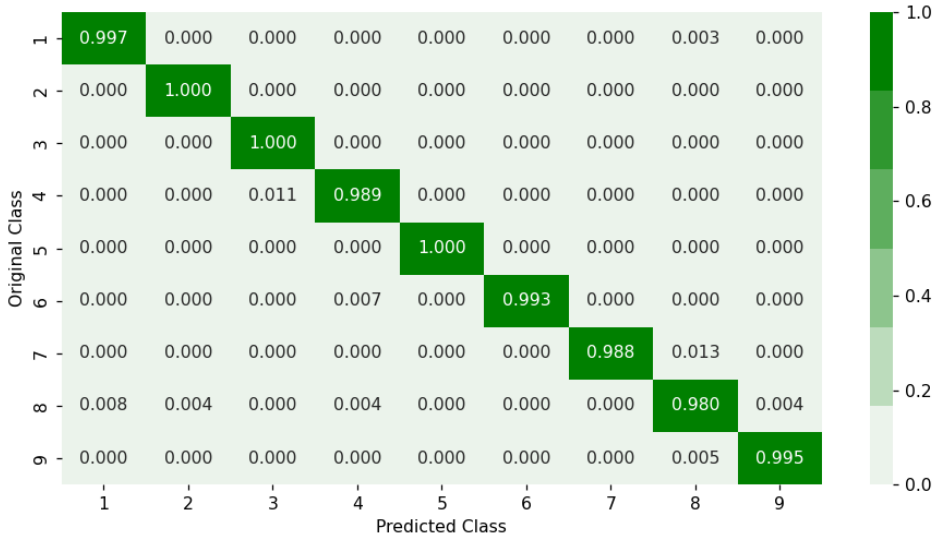
```
log loss for train data 0.016543571714631746
log loss for cv data 0.024995953922172888
log loss for test data 0.03002483994079553
Number of misclassified points  0.45998160073597055
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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



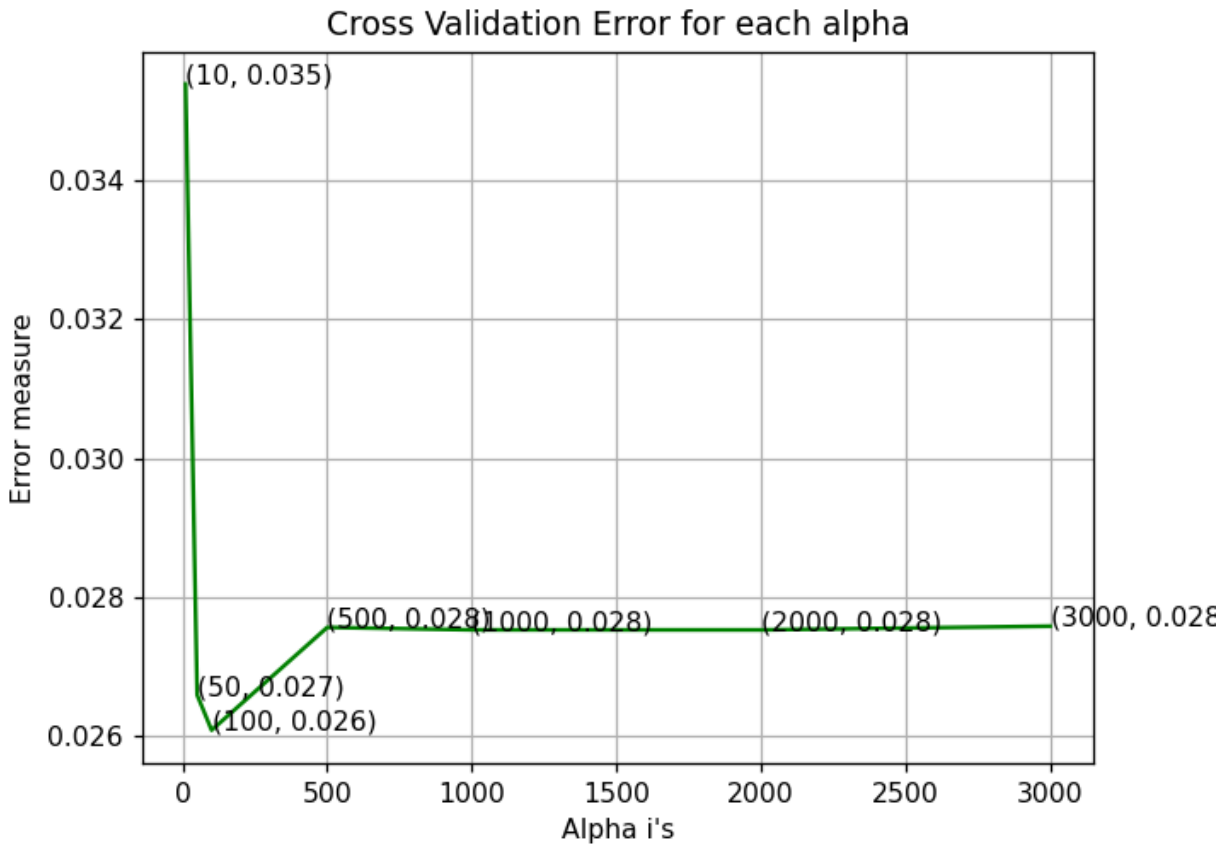

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Wall time: 1min 6s
```

4.4.4 XgBoost Classifier

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

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

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()
```

```
log_loss for c =  10 is 0.03538140487639453
log_loss for c =  50 is 0.026599269749628056
log_loss for c =  100 is 0.02609290651710922
log_loss for c =  500 is 0.027570471356489223
log_loss for c =  1000 is 0.02753506258354255
log_loss for c =  2000 is 0.027534291935260823
log_loss for c =  3000 is 0.027587652844795293

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




```
Wall time: 9min 23s
```

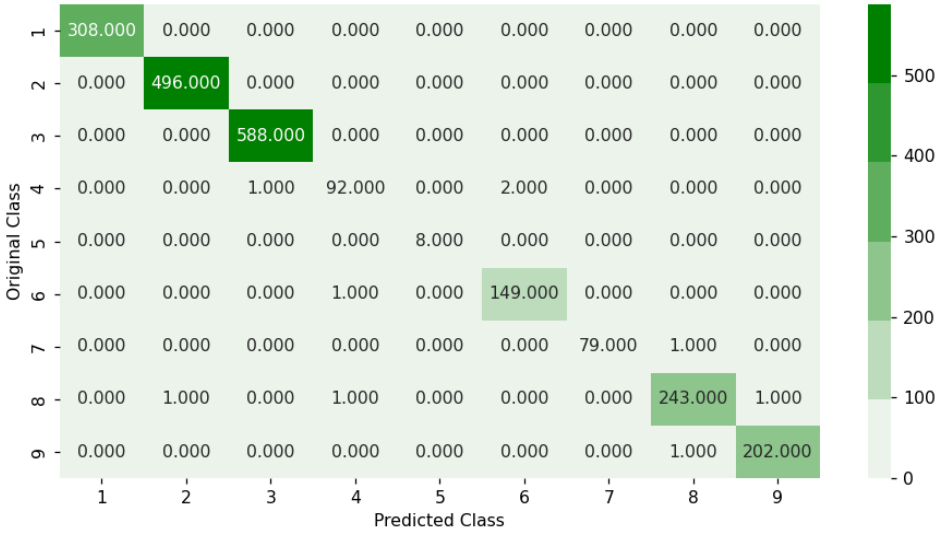
In [ ]:

```
%%time
# plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
x_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

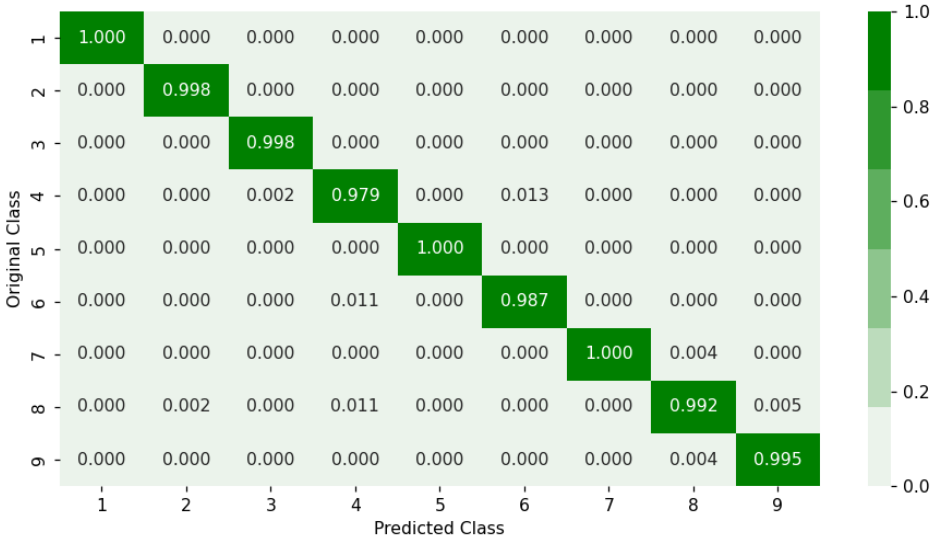
```
For values of best alpha =  100 The train log loss is: 0.014533560672759438
For values of best alpha =  100 The cross validation log loss is: 0.02609290651
710922
For values of best alpha =  100 The test log loss is: 0.033966683446424034
Wall time: 17.4 s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

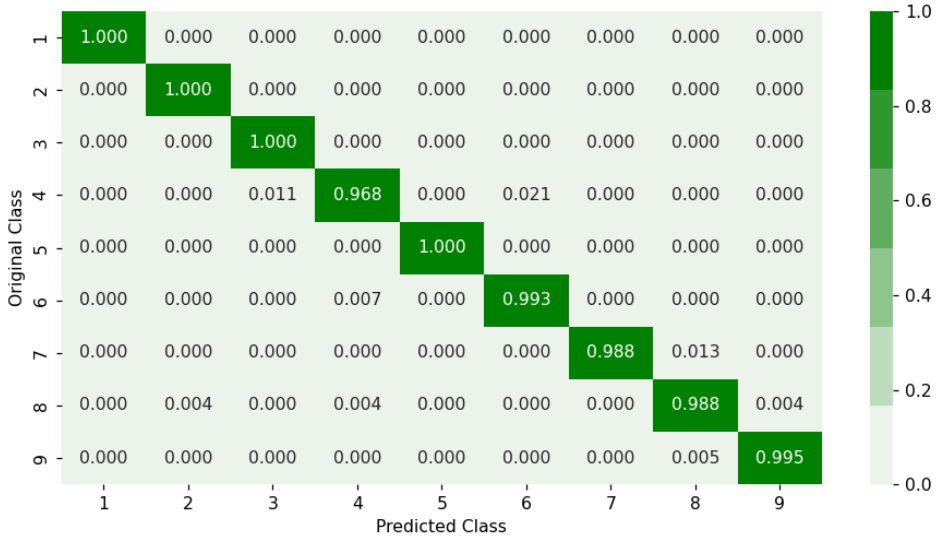
```
Number of misclassified points  0.41398344066237347
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

4.4.5 Xgboost Classifier with best hyperparameters

In [ ]:
```
%%time
x_cfl=XGBClassifier(eval_metric='merror')

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_j
random_cfl.fit(X_train_asm,y_train_asm)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Wall time: 5min 46s
```

Out[59]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=10)
```

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

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.03,
'colsample_bytree': 1}
```

In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

x_cfl=XGBClassifier(n_estimators=random_cfl.best_params_['n_estimators'],sub
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

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

```
train loss 0.013947345854695163
cv loss 0.0266339887970144
test loss 0.033521527573825725
```

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

4.5.1. Merging both asm and byte file features

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

Out[102]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148

5 rows × 261 columns

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

Out[63]:

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

5 rows × 55 columns

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

```
(10868, 261)
(10868, 55)
```

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

Out[65]:

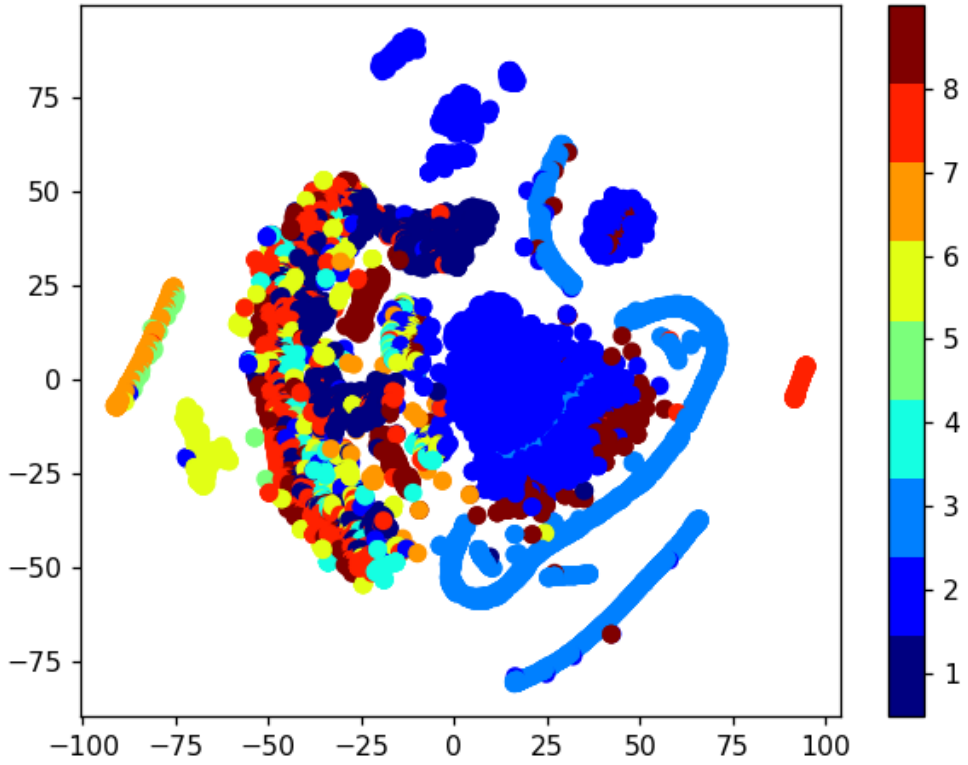
	0	1	2	3	4	5	6	7	8	
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000

5 rows × 309 columns

4.5.2. Multivariate Analysis on final features

In [ ]:
```
plt.close()

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()
```

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



4.5.3. Train and Test split

In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, re
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tr
```

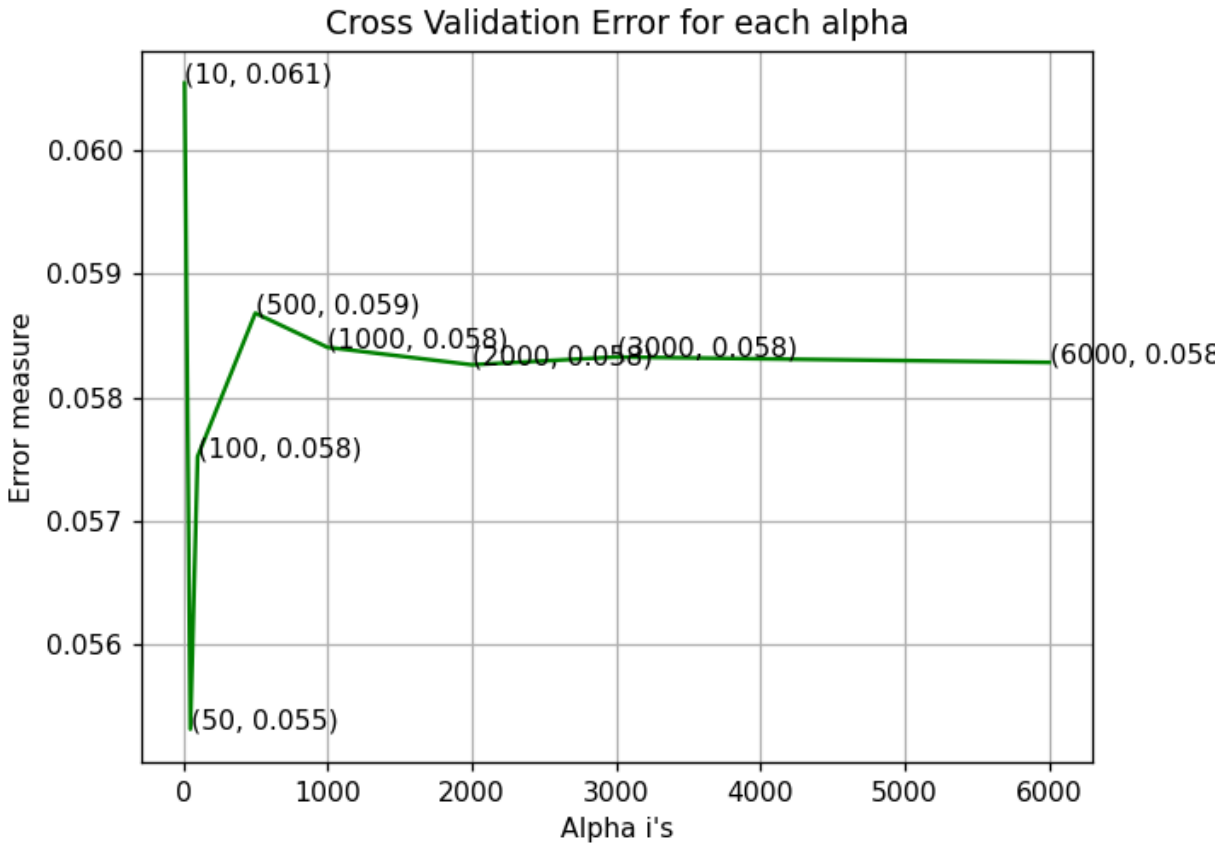
4.5.4. Random Forest Classifier on final features

In [ ]:

```
%%time
plt.close()
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# -------------------------------

alpha=[10,50,100,500,1000,2000,3000,6000]
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.c

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()
```

```
log_loss for c =  10 is 0.060549679958931804
log_loss for c =  50 is 0.055309964407794746
log_loss for c =  100 is 0.057520746644079135
log_loss for c =  500 is 0.05868398461810052
log_loss for c =  1000 is 0.058406703332937945
log_loss for c =  2000 is 0.058265162494557404
```

```
log_loss for c =  3000 is 0.05832805890136454
log_loss for c =  6000 is 0.058284390246964767

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




```
Wall time: 11min 29s
```

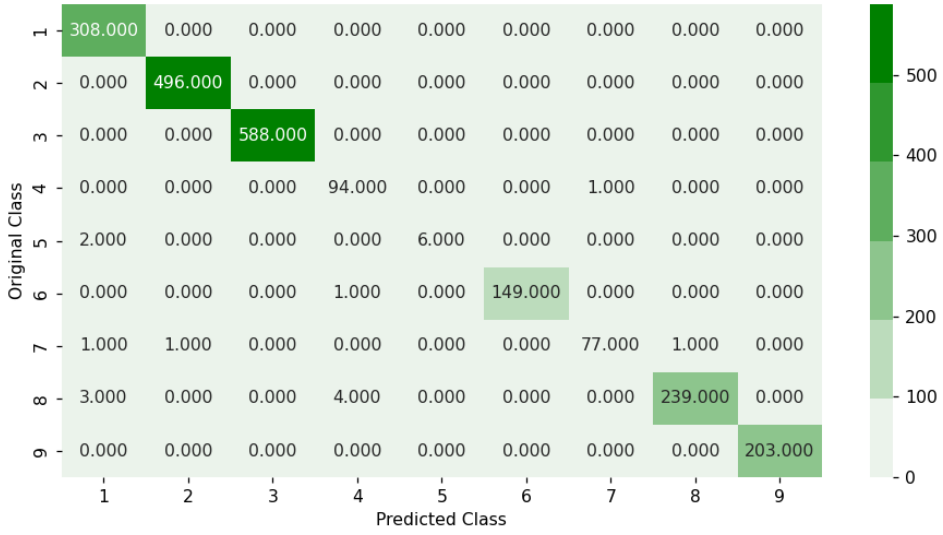
In [ ]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

```
For values of best alpha =  50 The train log loss is: 0.016434093303097957
For values of best alpha =  50 The cross validation log loss is: 0.055309964407
794746
For values of best alpha =  50 The test log loss is: 0.037498216951384854
Wall time: 4.66 s
```

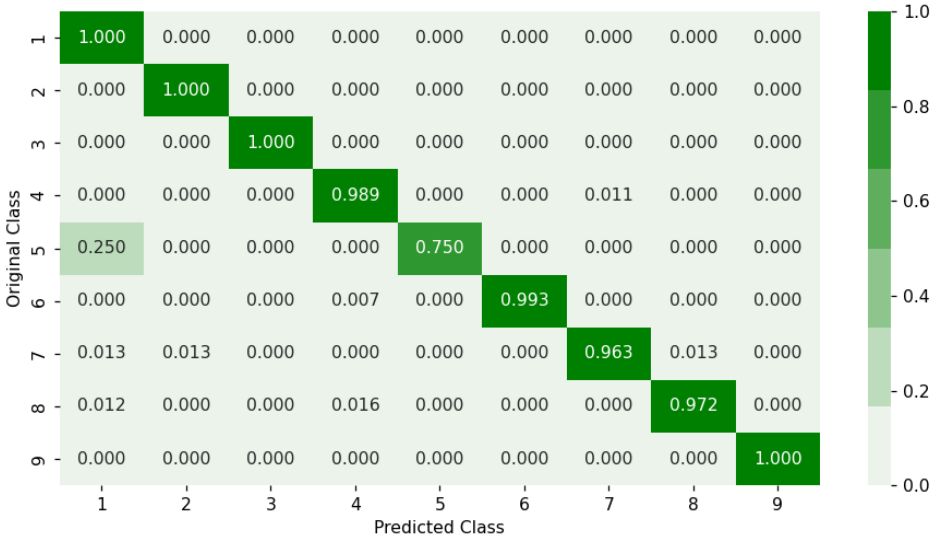
In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
Number of misclassified points  0.6439742410303588
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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


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

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




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

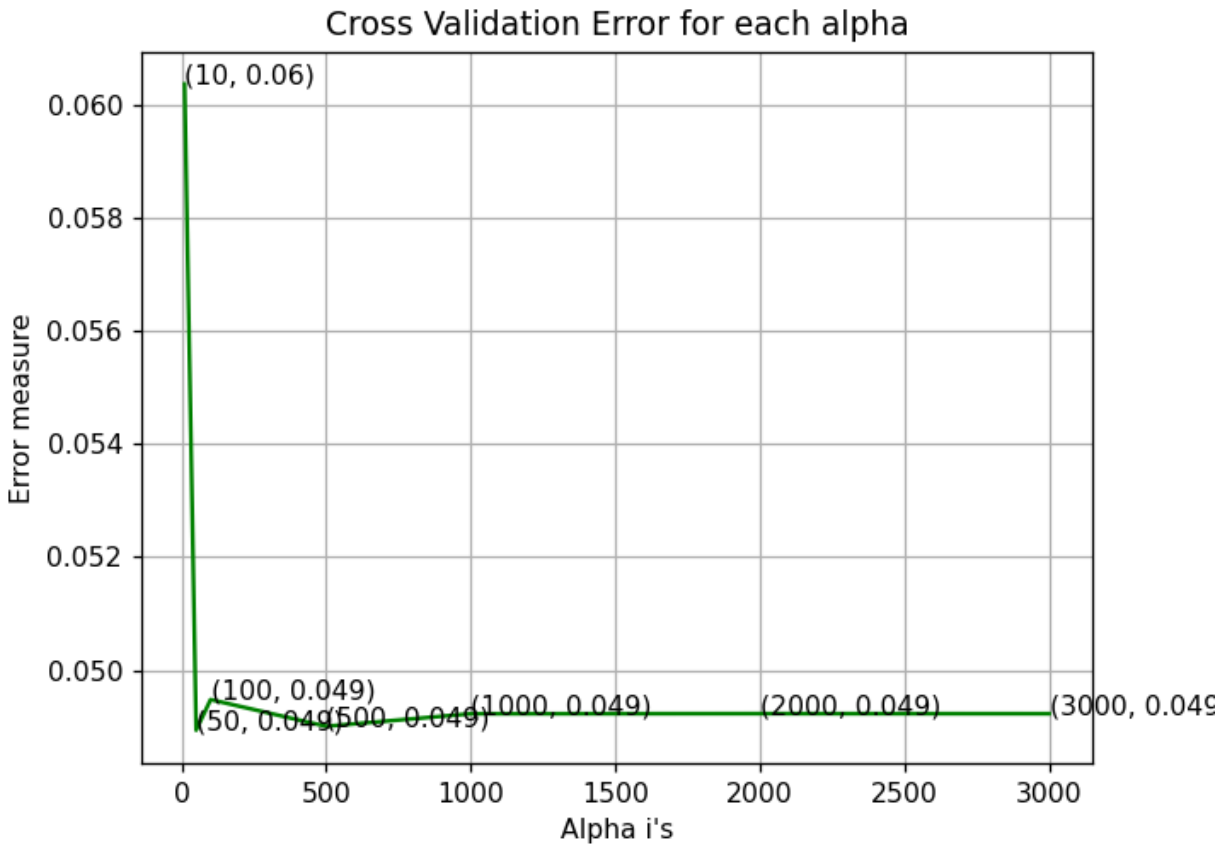
4.5.5. XgBoost Classifier on final features

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
# merror: Multiclass classification error rate. It is calculated as #(wrong
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=-1,eval_metric='merror')
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.c

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()
```

```
log_loss for c =  10 is 0.06036163528403693
log_loss for c =  50 is 0.04894097962092203
log_loss for c =  100 is 0.04948120187913875
log_loss for c =  500 is 0.049009150440359237
log_loss for c =  1000 is 0.0492309050652048
log_loss for c =  2000 is 0.049233220880327395
log_loss for c =  3000 is 0.049232047814634555
```

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




```
Wall time: 29min 38s
```

In [ ]:

```
%%time
plt.close()
# merror: Multiclass classification error rate. It is calculated as #(wrong
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  50 The train log loss is: 0.01243119470828622
For values of best alpha =  50 The cross validation log loss is: 0.048940979620
92203
For values of best alpha =  50 The test log loss is: 0.03604404772148142
Wall time: 59.3 s
```

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

In [ ]:
```
x_cfl=XGBClassifier(nthread=-1,eval_metric='merror')
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_j
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[72]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=10)
```

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

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 10, 'learning_rate': 0.1, 'c
olsample_bytree': 1}
```

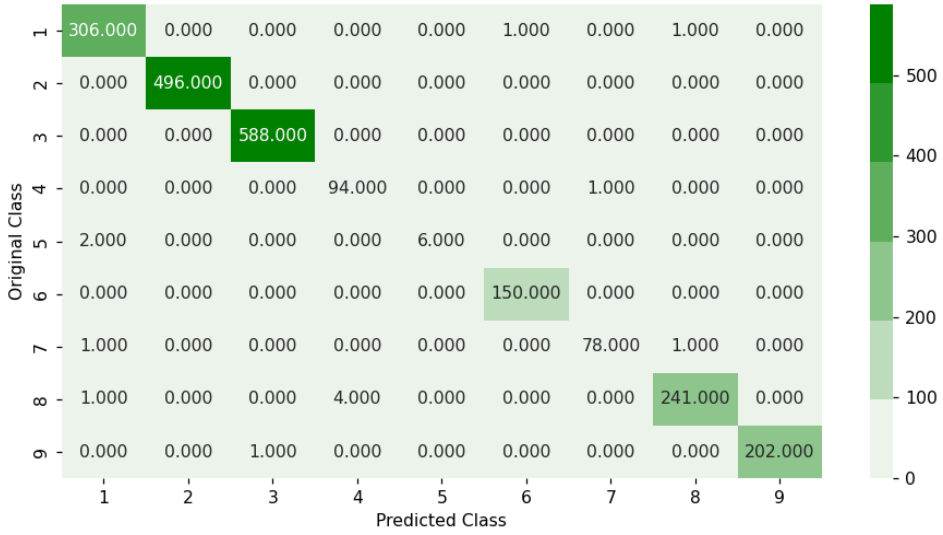
In [ ]:

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# ----------------------

x_cfl=XGBClassifier(n_estimators=random_cfl.best_params_['n_estimators'],max
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

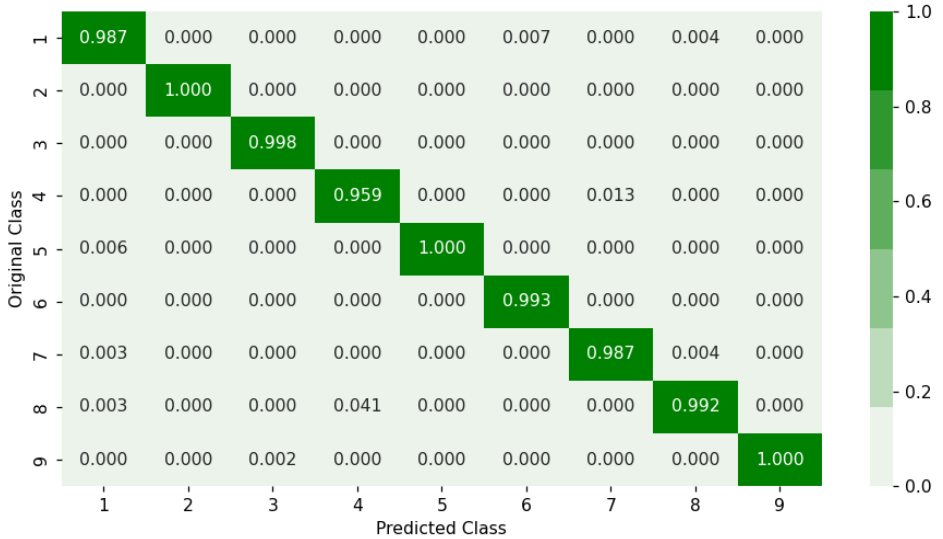
```
For values of best alpha =  50 The train log loss is: 0.012562886864752694
For values of best alpha =  50 The cross validation log loss is: 0.051314911398
118
For values of best alpha =  50 The test log loss is: 0.03588369895820438
Wall time: 3min 17s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

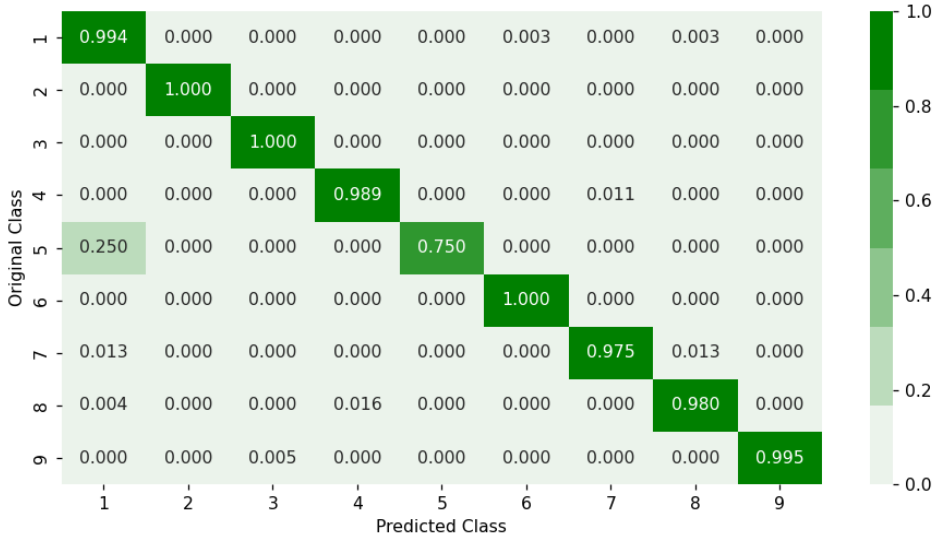
```
Number of misclassified points  0.5979760809567618
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

Bytes Files Model Comparision

In [1]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = (['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Lo

table.add_row(['Random Model', '-', 2.4661664622685913, 2.4989638402300267,
table.add_row(['K Nearest Neighbour',0.2537021671455636, 0.48298514961697087
table.add_row(['Logistic Regression', 0.8615268625525035, 0.8583275102454776
table.add_row(['Random Forest Classifier', 0.025936967738833847, 0.074400579
table.add_row(['XgBoost Classification', 0.02149461592509416, 0.070718862058
print(table)
```

```
+--------------------------+----------------------+---------------------+------
---------------+--------------------------------+
|          Model           |    Train Log Loss    |     CV Log Loss     |    Te
st Log Loss    | Number of Misclassified Points |
+--------------------------+----------------------+---------------------+------
---------------+--------------------------------+
|       Random Model       |          -           |  2.4661664622685913 |  2.49
89638402300267 |       89.37442502299908        |
|   K Nearest Neighbour    |  0.2537021671455636  | 0.48298514961697087 | 0.475
08072294405124 |       13.017479300827967       |
|   Logistic Regression    |  0.8615268625525035  |  0.8583275102454776 |  0.85
63175051825581 |       24.42502299908004        |
| Random Forest Classifier | 0.025936967738833847 | 0.07440057924615005 | 0.075
56461381685227 |       1.7939282428702852       |
|  XgBoost Classification  | 0.02149461592509416  | 0.07071886205835459 | 0.067
57843531946935 |       1.2879484820607177       |
+--------------------------+----------------------+---------------------+------
---------------+--------------------------------+
```

ASM Files Model Comparision

In [2]:
```
table = PrettyTable()
table.field_names  = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Lo

table.add_row(['K Nearest Neighbour', 0.028080753461076014, 0.06196307446768
table.add_row(['Logistic Regression', 1.03202773035354, 1.0494097878786035,1
table.add_row(['Random Forest Classifier', 0.016543571714631746,0.0260929065
table.add_row(['XgBoost Classification', 0.014533560672759438, 0.03159265585
print(table)
```

```
+--------------------------+----------------------+---------------------+------
----------------+---------------------------------+
|          Model           |    Train Log Loss    |     CV Log Loss     |    Te
st Log Loss     | Number of Misclassified Points |
+--------------------------+----------------------+---------------------+------
----------------+---------------------------------+
|   K Nearest Neighbour    | 0.028080753461076014 | 0.06196307446768273 | 0.090
35187549912337  |       1.3799448022079117       |
|   Logistic Regression    |   1.03202773035354   |  1.0494097878786035 |  1.04
8671333428269   |       28.978840846366143       |
| Random Forest Classifier | 0.016543571714631746 | 0.02609290651710922 | 0.030
02483994079553  |      0.45998160073597055       |
|  XgBoost Classification  | 0.014533560672759438 | 0.03159265585460439 | 0.033
966683446424034 |      0.41398344066237347       |
+--------------------------+----------------------+---------------------+------
----------------+---------------------------------+
```

Bytes + ASM Files Model Comparision

In [ ]:
```
table = PrettyTable()
table.field_names  = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Lo

table.add_row(['Random Forest Classifier', 0.016434093303097957, 0.055309964
table.add_row(['XgBoost Classification', 0.012562886864752694, 0.05131491139
print(table)
```

```
+--------------------------+----------------------+---------------------+------
----------------+---------------------------------+
|          Model           |    Train Log Loss    |     CV Log Loss     |    T
est Log Loss    | Number of Misclassified Points |
+--------------------------+----------------------+---------------------+------
----------------+---------------------------------+
| Random Forest Classifier | 0.014762155486539353 | 0.042874337981406434 | 0.04
192004241532452 |       0.9199632014719411       |
|  XgBoost Classification  | 0.010227838888593169 | 0.046105797578046656 | 0.03
857589259645944 |        0.78196872125115        |
+--------------------------+----------------------+---------------------+------
----------------+---------------------------------+
```

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=VLQTRILGz5Y#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, a
fter extracting it will occupy ~128GB data your dirve

2. if you are having computation power limitations, you can try using go
ogle colab, with GPU option enabled (you can search for how to enable GP
U in colab) 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 suggest you to use GCP over Colab)

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

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

Assignment Part 1

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

1.1 Process Bigram on ByteFiles

In [ ]:
```
# list byteFiles directory
files = os.listdir('byteFiles')
```

In [ ]:
```
len(files)
```

Out[89]:
```
10868
```

In [ ]:
```
all_keys = []

def calc_bi_grams(file):
    '''
    Calculate unigram and bigram
    '''
    temp_list = []
    with open('byteFiles/'+file,"r") as byte_flie:
        all_lines = []
        for lines in byte_flie:
            line=lines.rstrip().split(" ")
            all_lines.extend(line)
            # unigrams
            for hex_code in line:
                temp_list.append(hex_code.lower())
        temp_list = list(set(temp_list))
        # bigrams
        bi_g = [' '.join(x) for x in list(ngrams(all_lines, 2))]
        for hex_code in bi_g:
            temp_list.append(hex_code.lower())
        temp_list = list(set(temp_list))
    return temp_list
```

In [ ]:
```
# Process and collect bigram of bytes file
from tqdm import tqdm
for file in tqdm(files):
    all_keys.extend(calc_bi_grams(file))
    all_keys = list(set(all_keys))

len(all_keys)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 10868/10868 [4:39:31<00:00,  1.54s/it]
```

Out[93]: 66306

In [ ]:
```
all_keys
```

Out[94]:
```
['60 94',
 '67 33',
 '9e 44',
 'ce 22',
 'ee ae',
 '5e 85',
 'a8 fe',
 '29 07',
 '00 b5',
 '1c 03',
 'f2 d9',
 '12 01',
 '15 15',
 'df ae',
 '3f 8e',
 'ea c1',
 '2e 00',
 '75',
 'ff 80',
```

In [ ]:
```
with open('uni_bigram_keys.pkl', 'wb') as big:
    pickle.dump(all_keys, big)
```

In [ ]:
```
with open('uni_bigram_keys.pkl', 'rb') as big:
    all_keys = pickle.load(big)
```

In [ ]:
```
len(all_keys)
```

Out[78]:
```
66306
```

In [ ]:
```
a="00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15\
,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29\
,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,\
3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,\
53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,\
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,90,91,92,93,94,95,96,\
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,c0,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,ed,ee,ef,f0,f1,f2,f3,f4,f5,\
f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
a = a.replace(",", " ")
a_uni = a.split(" ")
len(a_uni)
```

Out[4]:
```
257
```

In [ ]:
```
# all_keys.extend(a_uni)
```

In [ ]:
```
1 len(all_keys)
```

Out[6]: 66306

In [ ]:
```
1 all_keys
```

Out[7]:
```
['60 94',
 '67 33',
 '9e 44',
 'ce 22',
 'ee ae',
 '5e 85',
 'a8 fe',
 '29 07',
 '00 b5',
 '1c 03',
 'f2 d9',
 '12 01',
 '15 15',
 'df ae',
 '3f 8e',
 'ea c1',
 '2e 00',
 '75',
 'ff 80',
```

In [ ]:

```
import scipy
from scipy.sparse import csr_matrix
from tqdm import tqdm

# bytebigram_vect = csr_matrix((len(files), len(all_keys)))
# from tqdm import tqdm
# for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
#     f = open('byteFiles/' + file)
#     bytebigram_vect[i, :] += csr_matrix(vector.fit_transform([f.read().rep
#     f.close()

def extract_bigrams_byte_features(files):
    pid = os.getpid()
    print('Process started with id = ', pid)
    ftot = len(files)

    str_pid = str(pid)

    vector_str_pid = CountVectorizer(lowercase=False,ngram_range=(2,2), voca
    bytebigram_vect_str_pid = csr_matrix((len(files), len(all_keys)))
    print('vector variable created with name ', 'bytebigram_vect', '_', str_
    for i, file in tqdm(enumerate(files)):
        f = open('byteFiles/' + file)
        bytebigram_vect_str_pid[i, :] += csr_matrix(vector_str_pid.fit_trans
        f.close()
#         print(pid, i + 1, 'of', ftot, 'files processed.')

        # Print progress
#         if (i+1) % 100 == 0:
#             print(pid, i + 1, 'of', ftot, 'files processed.')

    filename = 'bytebigram_'+str_pid+'.npz'
    scipy.sparse.save_npz(filename, bytebigram_vect_str_pid)#'bytebigram.npz

# extract_bigrams_byte_features('abc')
```

In [ ]:

```
# Diving the entire file in 4 parts and process the files separartely
import time as tm
from multiprocessing import Pool
start_time = tm.time()
tfiles = os.listdir('byteFiles')
quart = int(len(tfiles)/4)
train1 = tfiles[:quart]
train2 = tfiles[quart:(2*quart)]
train3 = tfiles[(2*quart):(3*quart)]
train4 = tfiles[(3*quart):]
print(len(tfiles), quart, (len(train1)+len(train2)+len(train3)+len(train4)))
trains = [train1, train2, train3, train4]
print("Elapsed time: {:.2f} hours.".format((tm.time() - start_time)/3600.0))
```

```
10868 2717 10868
Elapsed time: 0.00 hours.
```

In [ ]:
```
1 extract_bigrams_byte_features(train1)
```

```
Process started with id =  1804
vector variable created with name  bytebigram_vect _ 1804

2717it [2:36:27,  3.46s/it]
```

In [ ]:
```
1 extract_bigrams_byte_features(train2)
```

```
Process started with id =  1804
vector variable created with name  bytebigram_vect _ 1804

2717it [2:43:48,  3.62s/it]
```

In [ ]:
```
1 extract_bigrams_byte_features(train3)
```

```
Process started with id =  1804
vector variable created with name  bytebigram_vect _ 1804

2717it [2:51:51,  3.80s/it]
```

In [ ]:
```
1 extract_bigrams_byte_features(train4)
```

In [ ]:
```
1  # Load all genereated npz files
2  import scipy
3  from scipy.sparse import csr_matrix
4  from sklearn.preprocessing import normalize
5  bytebigram_vect_1 = scipy.sparse.load_npz('train1_bigram.npz')#normalize(sci
6  bytebigram_vect_2 = scipy.sparse.load_npz('train2.npz')#normalize(scipy.spar
7  bytebigram_vect_3 = scipy.sparse.load_npz('train3.npz')#normalize(scipy.spar
8  bytebigram_vect_4 = scipy.sparse.load_npz('train4.npz')#normalize(scipy.spar
9  # bytebigram_vect_5 = normalize(scipy.sparse.load_npz('bytebigram_8767.npz')
10 # bytebigram_vect_6 = normalize(scipy.sparse.load_npz('bytebigram_8768.npz')
```

In [ ]:
```
1 # Merge all npz files
2 from scipy.sparse import vstack
3 bytebigram_vect_7 = vstack((bytebigram_vect_1, bytebigram_vect_2, bytebigram
4 # , bytebigram_vect_5, bytebigram_vect_6
5
6 bytebigram_vect_7
```

Out[4]:
```
<10868x66306 sparse matrix of type '<class 'numpy.float64'>'
        with 502109160 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
1 import scipy
2 scipy.sparse.save_npz("FULL_train_byte_bigram_matrix", bytebigram_vect_7)
```

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

In [ ]:
```
# Load all genereated npz files
import scipy
from scipy.sparse import csr_matrix
from sklearn.preprocessing import normalize
bytebigram_vect_7= scipy.sparse.load_npz('FULL_train_byte_bigram_matrix.npz'
```

In [ ]:
```
scipy.sparse.save_npz("Normalised_bigram_matrix", normalize(bytebigram_vect_
```

In [ ]:
```
import scipy
Normalized_bigram_matrix=scipy.sparse.load_npz('Normalised_bigram_matrix.npz
```

In [ ]:
```
# collect top 300 features from bigrams
byte_bi_indxes_1 = imp_features(Normalized_bigram_matrix, all_keys, 300)
```

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

In [ ]:
```
data_y.head()
result_y = data_y
```

In [ ]:
```
byte_bi_indxes_1
```

Out[71]:
```
array([48949, 38665, 43568,  5490, 25036,  2051, 28062, 42546, 19758,
       25080, 47662, 36889, 25563, 44401, 19935, 45256, 10732,   734,
       36283,  7968, 45660, 18881,  6666,  8394,  4632, 45989, 64156,
       53767, 55835, 61242,  3003, 53971, 49904, 55666, 25954, 64731,
       55199, 30333,  7818, 45772, 50942, 13580, 21811, 65345,  1936,
         395, 62692, 27478, 15946,  4938, 16380, 63495, 59181, 30661,
        9122, 19551, 18442, 26345, 48864,  3491, 28135, 12672, 50892,
       14398, 63869,  7062, 28408, 64219, 33748, 12560, 38590, 45990,
       65206, 47446, 17579, 59845, 26204, 60627, 36780, 38579, 10157,
        5793, 27705, 58437, 29971, 24099, 10250, 49726, 60261, 33985,
       22551, 23567,  5478, 50191, 42341, 38687, 31788,  2709, 47275,
       42466, 41739, 63100, 50702, 31500, 34913, 12806,  3180, 51493,
       25769, 14667, 11820, 31244, 22944, 13430,  6947, 55668, 43023,
       40300, 61437,  1867, 25634, 53568, 63581,   863,  5304, 65933,
       62158, 47240, 61965, 26538, 27334, 24808, 24596,  2247, 23195,
       61311, 23505, 46247, 20629, 47538, 62166, 28989, 23150, 52925,
        3514, 25829, 53099,  5839,  5745, 16694, 28559, 36739, 19131,
       43367, 56825, 54495, 26478, 45114, 20861, 19922, 50910, 42786,
       25554, 11818,   899, 44325, 30612, 39625,  3367, 39294, 40729,
       41160, 20229, 45435, 41632, 21969,  1409, 18726,  9110, 50856,
        1392, 37524, 32247, 62046, 52949, 12927, 42631, 17893, 26938,
       54087, 24977, 61406, 11247, 23169, 60847, 61890,  2479, 45587,
       28790, 47028, 60070, 42401, 30238, 30094, 31647, 53628, 42553,
       41626,  5616, 52366, 29255, 63408, 49721, 11762, 56010, 43669,
       41257, 59100, 51966,   739, 13310,  6637, 54936,  3763,  7979,
       57353,  4744, 26125, 46201, 42383, 15110,  9171,  6865,  5155,
       60504, 41202, 31319, 42986, 63492, 31174, 64280, 57534,  7471,
       37398,  5998, 52529,  1343, 65522, 24171,  1821,  8910,  6647,
       38961, 50635, 27556, 17800, 59166, 23663, 59364, 19232, 42268,
        1121, 24981, 54801, 41502, 24884,  4685, 20687, 20615, 52670,
        7232, 64420, 58578, 26627, 55755, 51843,  6784,  1560, 20043,
       33120, 18645,  7222, 55429, 50705, 37556, 57751, 12988, 20539,
       65137, 45041, 50152,  9458, 45173, 29057, 36927,  3196, 45679,
        1879, 64538, 63080], dtype=int64)
```

In [ ]:
```
# Save generated index file
np.save('byte_bi_indx', byte_bi_indxes_1)
```

In [ ]:
```
byte_bi_indxes = np.load('byte_bi_indx.npy')
```

In [ ]:
```
byte_bi_indxes
```

Out[81]:
```
array([48949, 38665, 43568,  5490, 25036,  2051, 28062, 42546, 19758,
       25080, 47662, 36889, 25563, 44401, 19935, 45256, 10732,   734,
       36283,  7968, 45660, 18881,  6666,  8394,  4632, 45989, 64156,
       53767, 55835, 61242,  3003, 53971, 49904, 55666, 25954, 64731,
       55199, 30333,  7818, 45772, 50942, 13580, 21811, 65345,  1936,
         395, 62692, 27478, 15946,  4938, 16380, 63495, 59181, 30661,
        9122, 19551, 18442, 26345, 48864,  3491, 28135, 12672, 50892,
       14398, 63869,  7062, 28408, 64219, 33748, 12560, 38590, 45990,
       65206, 47446, 17579, 59845, 26204, 60627, 36780, 38579, 10157,
        5793, 27705, 58437, 29971, 24099, 10250, 49726, 60261, 33985,
       22551, 23567,  5478, 50191, 42341, 38687, 31788,  2709, 47275,
       42466, 41739, 63100, 50702, 31500, 34913, 12806,  3180, 51493,
       25769, 14667, 11820, 31244, 22944, 13430,  6947, 55668, 43023,
       40300, 61437,  1867, 25634, 53568, 63581,   863,  5304, 65933,
       62158, 47240, 61965, 26538, 27334, 24808, 24596,  2247, 23195,
       61311, 23505, 46247, 20629, 47538, 62166, 28989, 23150, 52925,
        3514, 25829, 53099,  5839,  5745, 16694, 28559, 36739, 19131,
       43367, 56825, 54495, 26478, 45114, 20861, 19922, 50910, 42786,
       25554, 11818,   899, 44325, 30612, 39625,  3367, 39294, 40729,
       41160, 20229, 45435, 41632, 21969,  1409, 18726,  9110, 50856,
        1392, 37524, 32247, 62046, 52949, 12927, 42631, 17893, 26938,
       54087, 24977, 61406, 11247, 23169, 60847, 61890,  2479, 45587,
       28790, 47028, 60070, 42401, 30238, 30094, 31647, 53628, 42553,
       41626,  5616, 52366, 29255, 63408, 49721, 11762, 56010, 43669,
       41257, 59100, 51966,   739, 13310,  6637, 54936,  3763,  7979,
       57353,  4744, 26125, 46201, 42383, 15110,  9171,  6865,  5155,
       60504, 41202, 31319, 42986, 63492, 31174, 64280, 57534,  7471,
       37398,  5998, 52529,  1343, 65522, 24171,  1821,  8910,  6647,
       38961, 50635, 27556, 17800, 59166, 23663, 59364, 19232, 42268,
        1121, 24981, 54801, 41502, 24884,  4685, 20687, 20615, 52670,
        7232, 64420, 58578, 26627, 55755, 51843,  6784,  1560, 20043,
       33120, 18645,  7222, 55429, 50705, 37556, 57751, 12988, 20539,
       65137, 45041, 50152,  9458, 45173, 29057, 36927,  3196, 45679,
        1879, 64538, 63080], dtype=int64)
```

In [ ]:
```
# Pick top 300 features
from tqdm import tqdm
top_byte_bi = np.zeros((10868, 0))
for i in tqdm(byte_bi_indxes):
    sliced = bytebigram_vect_7[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

```
100%|██████████| 300/300 [5:42:56<00:00, 68.59s/it]
```

In [ ]:
```
# Create a DataFrame using top 300 features
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns = np.take(all_keys, byt
```

In [ ]:
```
# Save DataFrame into CSV
byte_bi_df.to_dense().to_csv('byte_bi.csv')
```

In [ ]:
```
# Read csv and start processing the data
byte_bi_df = pd.read_csv('byte_bi.csv').drop('Unnamed: 0', axis = 1)#fillna(
```

In [ ]:
```
1 byte_bi_df
```

Out[83]:

	f1 98	29 87	ae e0	f5 a0	82 cd	45 08	5a 52	ae ef	89 30	08 72	...	3b 7c	8d 45	ff 8b	5e 8b	49 00	
0	10.0	9.0	5.0	6.0	4.0	10.0	4.0	7.0	3.0	3.0	...	5.0	10.0	226.0	15.0	1310.0	
1	0.0	14.0	0.0	0.0	0.0	96.0	0.0	0.0	11.0	4703.0	...	1.0	48.0	390.0	5.0	58.0	1
2	5.0	3.0	7.0	11.0	5.0	25.0	2.0	6.0	4.0	21.0	...	1.0	30.0	823.0	36.0	573.0	1
3	1.0	2.0	0.0	1.0	2.0	25.0	3.0	2.0	5.0	7.0	...	4.0	49.0	521.0	23.0	32.0	1
4	0.0	0.0	0.0	2.0	0.0	4.0	2.0	2.0	2.0	0.0	...	2.0	7.0	10.0	1.0	5.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10863	3.0	9.0	1.0	3.0	4.0	5.0	9.0	2.0	5.0	3.0	...	3.0	7.0	2.0	13.0	5.0	
10864	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	2.0	1.0	6.0	5.0	
10865	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	1.0	3.0	5.0	13.0	
10866	0.0	1.0	1.0	0.0	0.0	0.0	2.0	1.0	0.0	1.0	...	1.0	2.0	1.0	4.0	0.0	
10867	0.0	1.0	1.0	1.0	4.0	2.0	5.0	1.0	1.0	1.0	...	1.0	1.0	1.0	6.0	0.0	

10868 rows × 300 columns

In [ ]:
```
1 result_x = byte_bi_df
2 result_y = data_y
3 result_x.tail()
```

Out[84]:

	f1 98	29 87	ae e0	f5 a0	82 cd	45 08	5a 52	ae ef	89 30	08 72	...	3b 7c	8d 45	ff 8b	5e 8b	49 00	08 03	2b 15	00 75	
10863	3.0	9.0	1.0	3.0	4.0	5.0	9.0	2.0	5.0	3.0	...	3.0	7.0	2.0	13.0	5.0	4.0	7.0	4.0	
10864	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	2.0	1.0	6.0	5.0	2.0	0.0	10.0	
10865	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	1.0	3.0	5.0	13.0	0.0	1.0	72.0	
10866	0.0	1.0	1.0	0.0	0.0	0.0	2.0	1.0	0.0	1.0	...	1.0	2.0	1.0	4.0	0.0	1.0	0.0	5.0	
10867	0.0	1.0	1.0	1.0	4.0	2.0	5.0	1.0	1.0	1.0	...	1.0	1.0	1.0	6.0	0.0	0.0	0.0	2.0	

5 rows × 300 columns

In [ ]:
```
1 result_y.head()
```

Out[85]:
```
0    9
1    2
2    9
3    1
4    8
Name: Class, dtype: int64
```

1.2 Prepare, Merge and Split data

In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, re
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tr
```

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

Out[87]: ((8694, 300), (8694,))

In [ ]:
```
result_x.shape
```

Out[88]: (10868, 300)

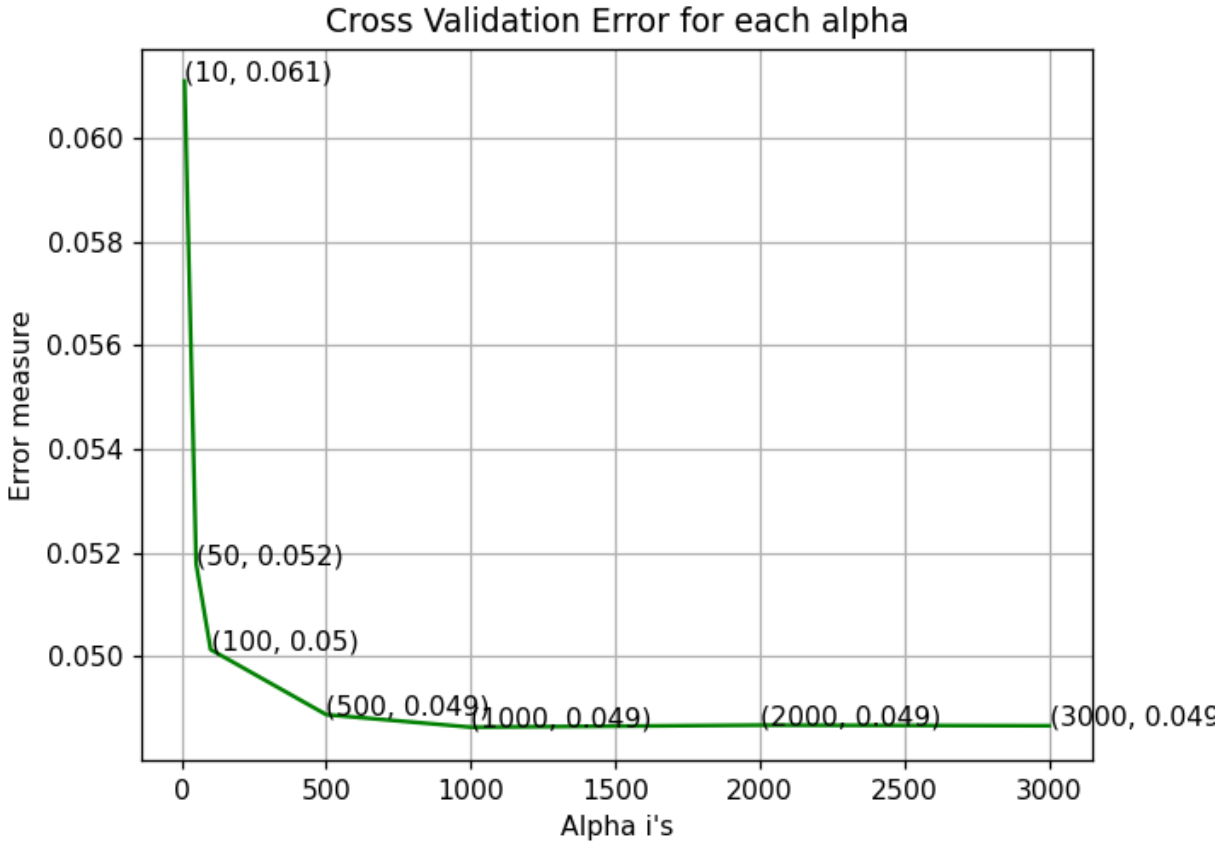
1.3 Random Forest Classifier on Bigram of ByteFiles

In [ ]:

```
%%time
plt.close()
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# ------------------------------

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.c

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()
```

```
log_loss for c =  10 is 0.06109666960402351
log_loss for c =  50 is 0.05178065880799005
log_loss for c =  100 is 0.050128628969220844
log_loss for c =  500 is 0.04887235296932197
log_loss for c =  1000 is 0.04863066872329781
```

```
log_loss for c =  2000 is 0.048669700370143405
log_loss for c =  3000 is 0.048658142912509424

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




```
Wall time: 3min 52s
Parser   : 105 ms
```

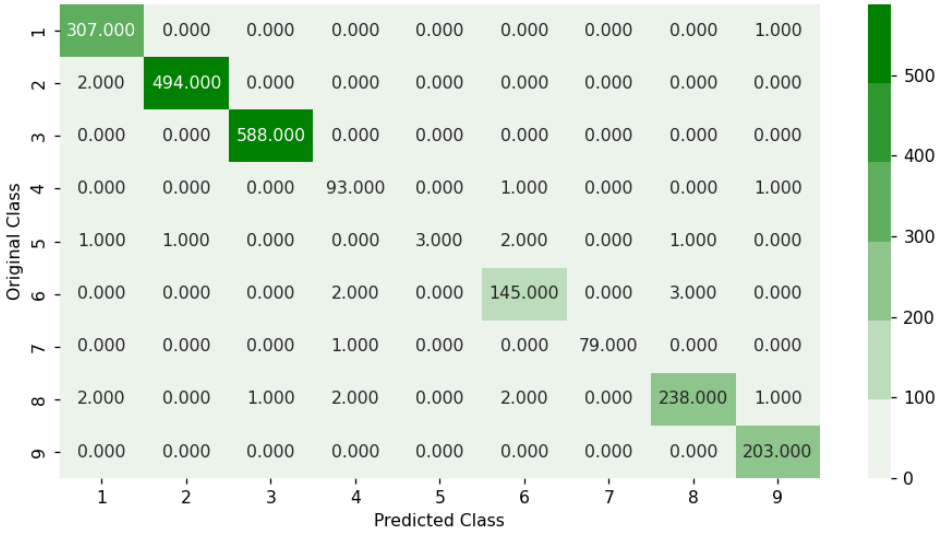
In [ ]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

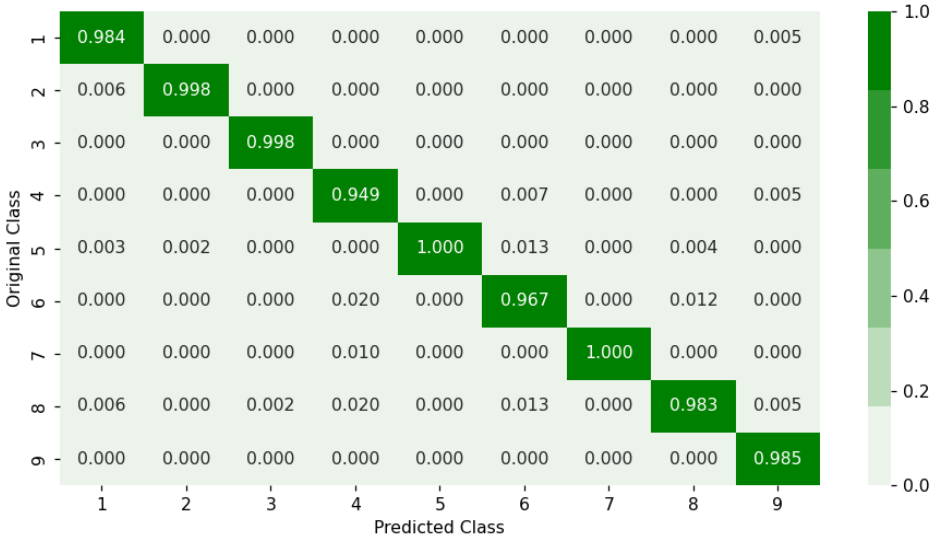
```
For values of best alpha =  1000 The train log loss is: 0.017783111288722803
For values of best alpha =  1000 The cross validation log loss is: 0.0486306687
2329781
For values of best alpha =  1000 The test log loss is: 0.04731769026164133
Wall time: 32.5 s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

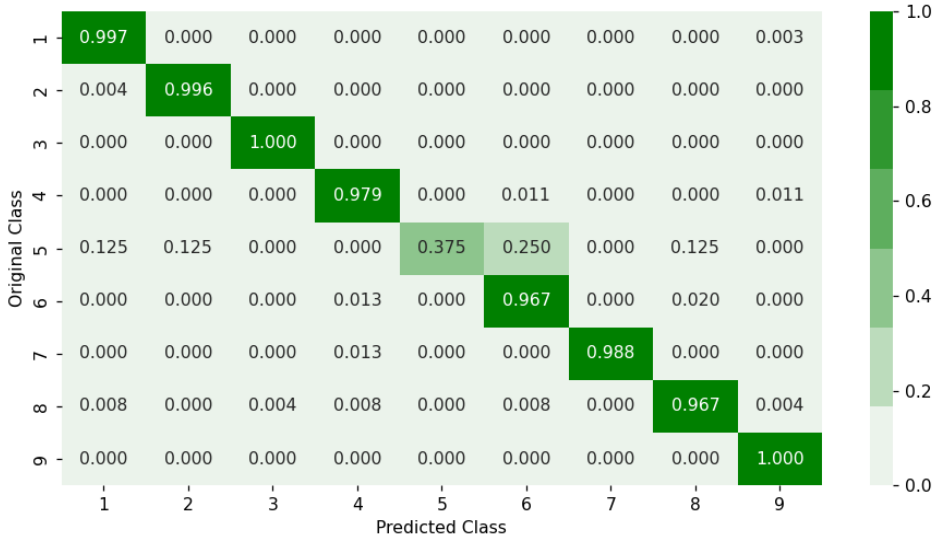
```
Number of misclassified points  1.1039558417663293
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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



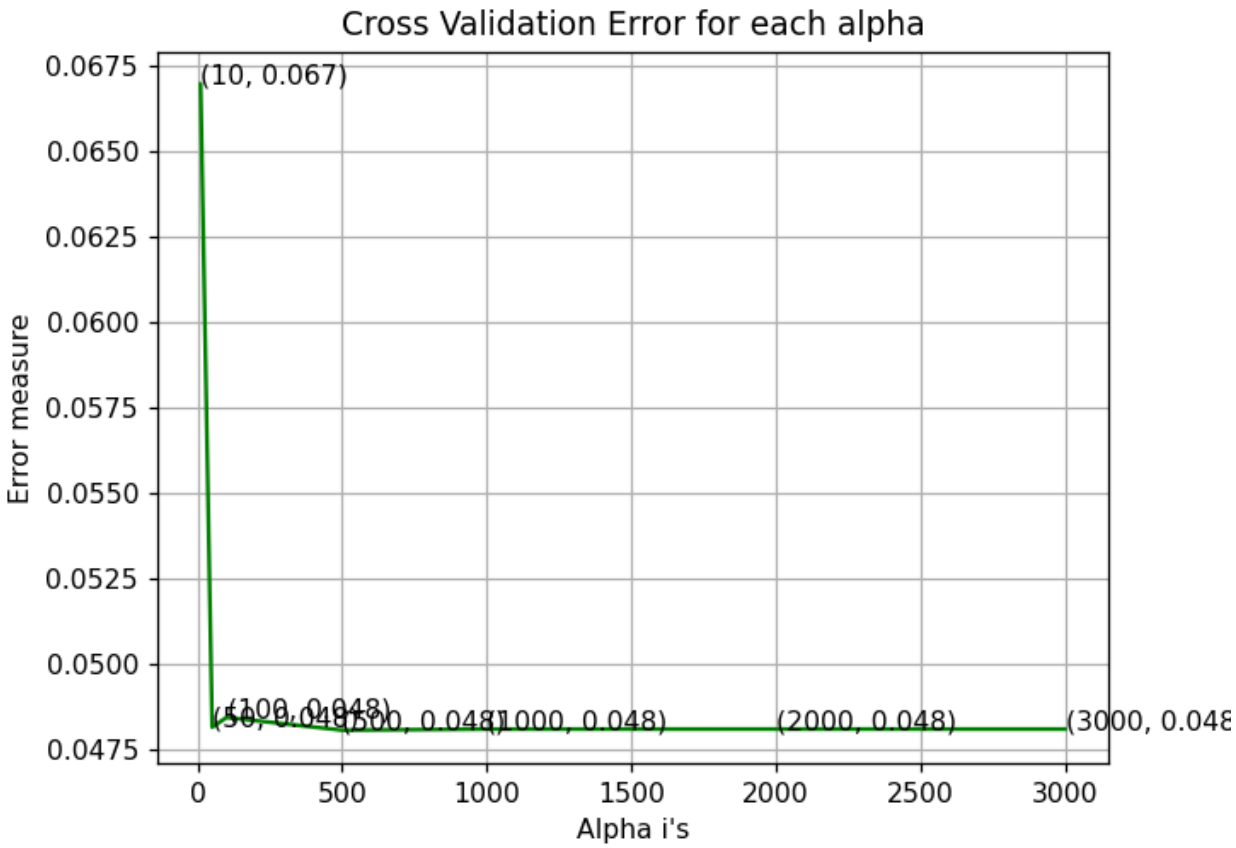

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

1.4 Xgboost Classifier on Bigram of ByteFiles

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

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

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()
```

```
log_loss for c =  10 is 0.06697824095419165
log_loss for c =  50 is 0.04814280570737251
log_loss for c =  100 is 0.04842637811150022
log_loss for c =  500 is 0.04805438434135443
log_loss for c =  1000 is 0.048085724130986024
log_loss for c =  2000 is 0.04808664180432906
log_loss for c =  3000 is 0.048086354374810125

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




```
Wall time: 26min 44s
```

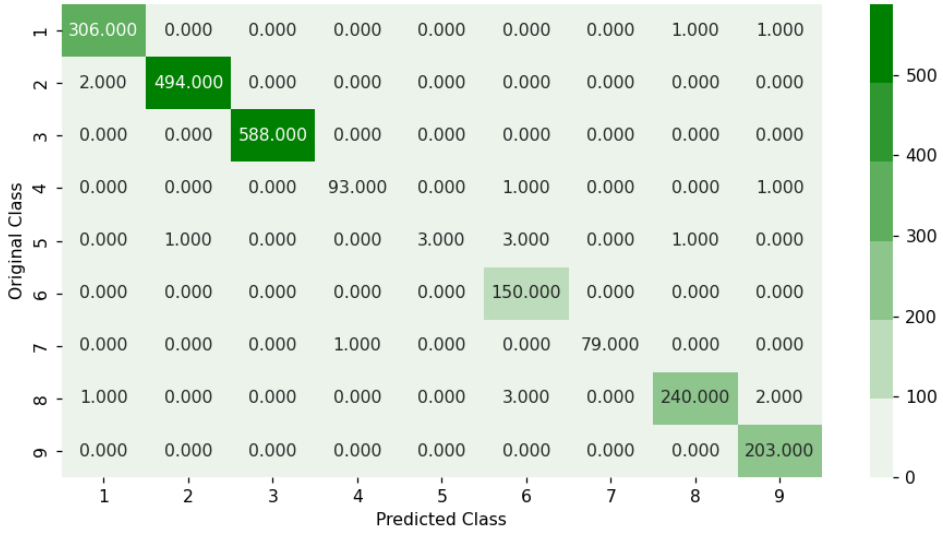
In [ ]:

```
%%time
# plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

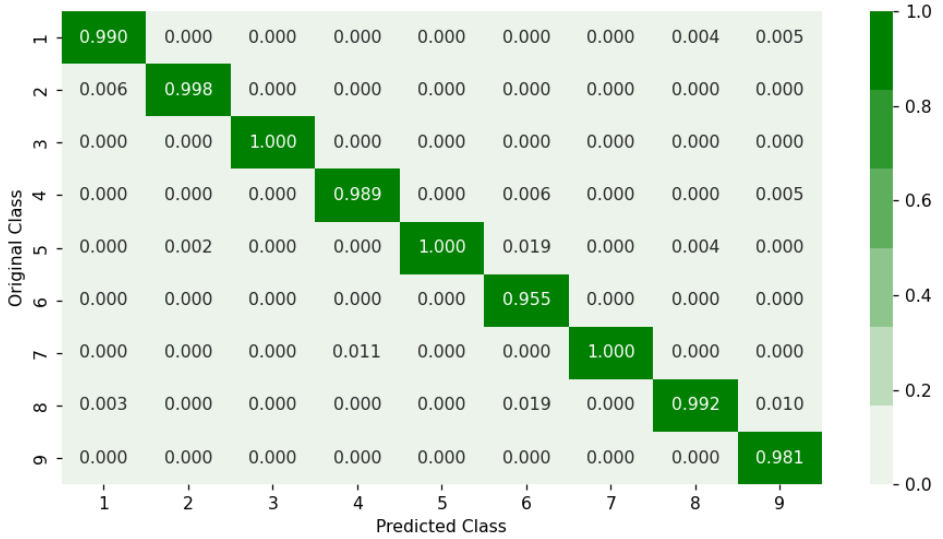
```
For values of best alpha =  500 The train log loss is: 0.016338414233495562
For values of best alpha =  500 The cross validation log loss is: 0.04805438434
135443
For values of best alpha =  500 The test log loss is: 0.04741783436921634
Wall time: 9min 24s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

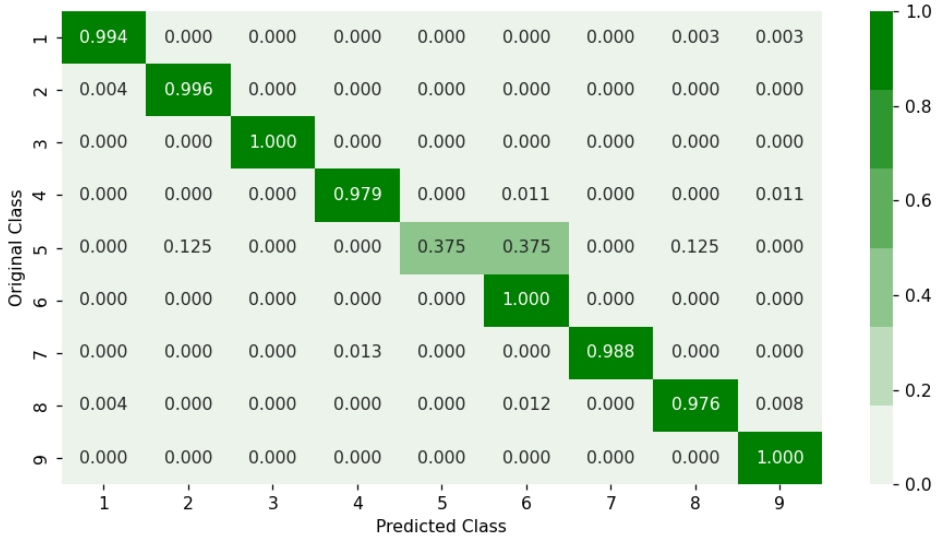
```
Number of misclassified points  0.8279668813247469
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

1.5 Xgboost Classifier (Best Hyper Parameter) on Bigram of ByteFiles

In [ ]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuni
x_cfl=XGBClassifier(nthread=-1,eval_metric='merror')

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_j
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[100]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=10)
```

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

```
{'subsample': 0.3, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.2,
'colsample_bytree': 0.1}
```

In [ ]:

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# ----------------------
x_cfl=XGBClassifier(n_estimators=random_cfl.best_params_['n_estimators'],max
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)
```

```
Wall time: 1min 1s
```

Out[104]:

```
CalibratedClassifierCV(base_estimator=XGBClassifier(base_score=0.5,
                                                    booster='gbtree',
                                                    colsample_bylevel=1,
                                                    colsample_bynode=1,
                                                    colsample_bytree=0.1,
                                                    enable_categorical=False,
                                                    eval_metric='merror',
                                                    gamma=0, gpu_id=-1,
                                                    importance_type=None,
                                                    interaction_constraints='',
                                                    learning_rate=0.2,
                                                    max_delta_step=0,
                                                    max_depth=3,
                                                    min_child_weight=1,
                                                    missing=nan,
                                                    monotone_constraints='()',
                                                    n_estimators=500, n_jobs=8,
                                                    nthread=-1,
                                                    num_parallel_tree=1,
                                                    objective='multi:softprob',
                                                    predictor='auto',
                                                    random_state=0, reg_alpha=
0,
                                                    reg_lambda=1,
                                                    scale_pos_weight=None,
                                                    subsample=0.3,
                                                    tree_method='exact',
                                                    validate_parameters=1,
...))
```

In [ ]:

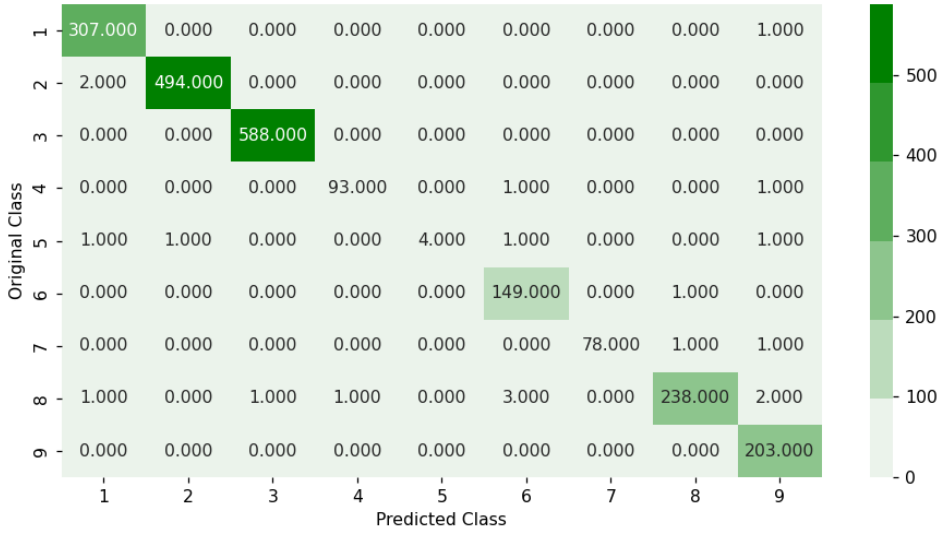
```
predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)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

```
For values of best alpha =  500 The train log loss is: 0.01575087655743497
For values of best alpha =  500 The cross validation log loss is: 0.04902037615
733592
For values of best alpha =  500 The test log loss is: 0.049200895459516676
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

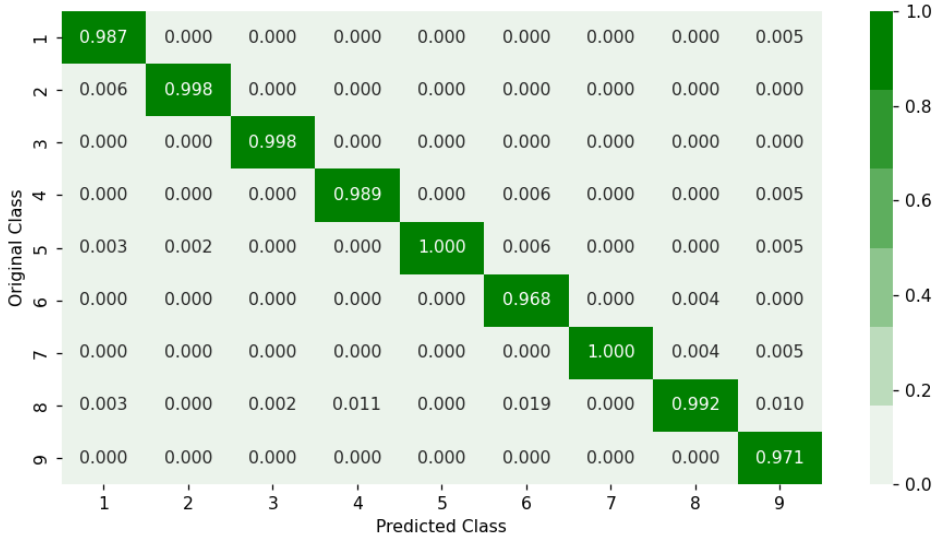
```
Number of misclassified points  0.9199632014719411
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```

<IPython.core.display.Javascript object>



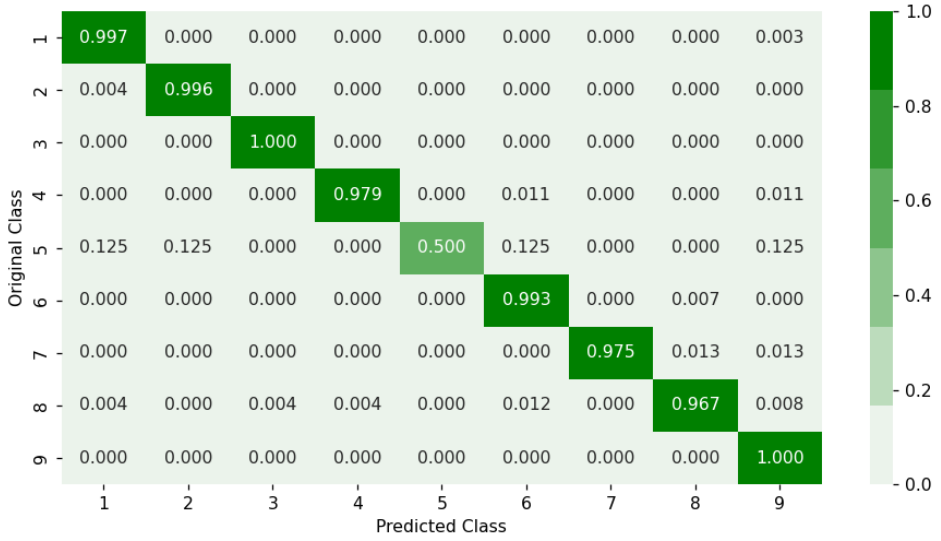

```
---------------------------------------------------- Precision matrix -----------
----------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

In [3]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names  = ['Model', 'Best Hyper Parameter', 'Train Log Loss', 'CV

table.add_row(['Random Forest Classifier', 1000,  0.017783111288722803, 0.04
table.add_row(['XgBoost Classifier', 500, 0.016338414233495562,0.04805438434
table.add_row(['XgBoost Classifier with Best Hyper Parameter', 500, 0.015750


print(table)
```

```
+-----------------------------------------------+----------------------+--------
--------------+--------------------+----------------------+--------------------
-------------+
|                    Model                      | Best Hyper Parameter |    Trai
n Log Loss    |     CV Log Loss      |    Test Log Loss     | Number of Misclass
ified Points |
+-----------------------------------------------+----------------------+--------
--------------+--------------------+----------------------+--------------------
-------------+
|           Random Forest Classifier            |         1000         | 0.01778
3111288722803 | 0.04863066872329781 | 0.04731769026164133  |       1.1039558417
663293       |
|             XgBoost Classifier                |         500          | 0.01633
8414233495562 | 0.04805438434135443 | 0.04741783436921634  |       0.8279668813
247469       |
| XgBoost Classifier with Best Hyper Parameter |         500          | 0.01575
087655743497  | 0.04902037615733592 | 0.049200895459516676 |       0.9199632014
719411       |
+-----------------------------------------------+----------------------+--------
--------------+--------------------+----------------------+--------------------
-------------+
```

Assignment Part 2

- 2 Use features from link and reduce log loss https://github.com/dchad/malware-detection (https://github.com/dchad/malware-detection)

2.1 Image Based Features Extraction from asm files

In [ ]:

```
from multiprocessing import Pool
import os
from csv import writer
import numpy as np
import math
import scipy.misc
import array
import time as tm

import numpy as np
import scipy as sp
import pandas as pd
import sklearn as skl
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.feature_selection import chi2
from sklearn.metrics import log_loss, confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.model_selection import cross_val_score, KFold
```

In [ ]:

```
# you read the file in binary
# get the length of the file
# the length can vary say you have length of file as 1000, (you can set your

# now your width is 256, so each row in the image will have 256 pixels/value

# to make this happen you need what is the maximum rows you can have each of

# so you find what need to be cut of , so reminder = length % width

# 1000 % 256
# quotient = 3 (max 3 rows are possible which will be fully filled )
# remainder = 232 ( these values remain at the end as they cannot form a ful
# then you reshape the one dimensional binary array , to 2 dim array with ro

# now to convert this binary  to grayscale , you need to club every 8 bits a

# ( grayscale images have pixel values from 0 to 255 ie 256 values , and to
# after converting you have a image now which you can save or print

# but for our task we don't need whole image , we only need top 800 pixel va
def read_image(filename):
    '''
    Read image data
    '''
    f = open(filename,'rb')
    ln = os.path.getsize(filename) # length of file in bytes
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
#     here we read the file in the form of unsigned 8 bit integer from the a
    a.fromfile(f,ln-rem)
#     print("Max value of a is ", max(a))
#     print("The value of a is ",a[1:4],type(a))
    f.close()
    g = np.reshape(a,(int(len(a)/width), width))
#     print("The value of g is after reshaping ",max(g[1]))
    g = np.uint8(g)
#     print("the max value of g is ",max(g[1]))
    g = np.resize(g, (1000,))
    return list(g)
```

In [ ]:

```
x=read_image("asmFiles/01IsoiSMh5gxyDYTl4CB.asm")
```

```
Max value of a is  254
The value of g is after reshaping  124
the max value of g is  124
```

In [ ]:
```
from tqdm import tqdm
def extract_asm_image_features(tfiles):
    '''
    Extract image features from the asm files
    '''
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    # Generate feature file csv
    pid = os.getpid()
    feature_file = str(pid) + '-image-features-asm.csv'

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in
        fw.writerow(column_names)
        for idx, fname in tqdm(enumerate(asm_files)):
            file_id = fname.split('.')[0]
            print('asmFiles/'+ fname)
            image_data = read_image('asmFiles/'+ fname)
            outrows.append([file_id] + image_data)
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []
```

In [ ]:
```
# Now divide the train files into four groups for processing
# As processing the whole files will make the ram full and slow down the ope
start_time = tm.time()
tfiles = os.listdir('asmFiles')
quart = int(len(tfiles)/4)
# print(quart)
train1 = tfiles[:quart]
train2 = tfiles[quart:(2*quart)]
train3 = tfiles[(2*quart):(3*quart)]
train4 = tfiles[(3*quart):]
print(len(tfiles), quart, (len(train1)+len(train2)+len(train3)+len(train4)))
trains = [train1, train2, train3, train4]
print("Elapsed time: {:.2f} hours.".format((tm.time() - start_time)/3600.0))
```

```
10868 2717 10868
Elapsed time: 0.00 hours.
```

In [ ]:
```
extract_asm_image_features(train1)
```

In [ ]:
```
extract_asm_image_features(train2)
```

In [ ]:
```
extract_asm_image_features(train3)
```

```
2717it [01:40, 27.03it/s]
```

In [ ]:
```
extract_asm_image_features(train3)
```

```
2717it [01:42, 26.51it/s]
```

In [ ]:
```
extract_asm_image_features(train4)
```

```
2717it [01:47, 25.25it/s]
```

In [ ]:
```
#merging all generated csv files

labels = pd.read_csv('trainLabels.csv')
d1 = pd.read_csv('Train_1-image-features-asm.csv')
d2 = pd.read_csv('Train_2-image-features-asm.csv')
d3 = pd.read_csv('Train_3-image-features-asm.csv')
d4 = pd.read_csv('Train_4-image-features-asm.csv')
d4.shape
```

Out[108]: (2717, 1001)

In [ ]:
```
data = pd.concat([d1, d2, d3, d4])
data.shape
```

Out[109]: (10868, 1001)

In [ ]:
```
data.reset_index(drop=True, inplace=True)
```

In [ ]:
```
labels.head()
```

Out[111]:

	Id	Class
0	01kcPWA9K2BOxQeS5Rju	1
1	04EjIdbPV5e1XroFOpiN	1
2	05EeG39MTRrI6VY21DPd	1
3	05rJTUWYAKNegBk2wE8X	1
4	0AnoOZDNbPXIr2MRBSCJ	1

In [ ]:
```
sorted_train_data = data.sort_values(by='filename', axis=0, ascending=True,
sorted_train_labels = labels.sort_values(by='Id', axis=0, ascending=True, in
X = sorted_train_data.iloc[:,1:]
y = np.array(sorted_train_labels.iloc[:,1])
```

In [ ]:
```
X.shape, y.shape
```

Out[113]: ((10868, 1000), (10868,))

2.1.1 Selecting top 50% variance features

In [ ]:
```
# find the top 50 percent variance features, from 1000 -> 500 features
fsp = SelectPercentile(chi2, percentile=50)
X_new_50 = fsp.fit_transform(X,y)
X_new_50.shape
```

Out[114]: (10868, 500)

In [ ]:
```
selected_names = fsp.get_support(indices=True)
selected_names = selected_names + 1
selected_names
```

Out[115]:
```
array([  2,   4,   5,  15,  21,  22,  24,  25,  26,  27,  29,  30,  32,
        33,  34,  35,  41,  42,  43,  44,  48,  50, 125, 126, 135, 136,
       138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 151, 152,
       154, 155, 156, 157, 158, 160, 161, 162, 163, 164, 165, 167, 169,
       173, 174, 179, 186, 188, 190, 198, 201, 202, 205, 215, 216, 217,
       219, 220, 221, 222, 223, 224, 226, 227, 229, 236, 240, 241, 242,
       243, 244, 245, 246, 247, 248, 249, 252, 253, 260, 261, 262, 263,
       264, 265, 266, 267, 268, 269, 271, 272, 273, 282, 287, 291, 292,
       293, 294, 295, 296, 297, 307, 308, 310, 311, 312, 313, 314, 315,
       316, 317, 318, 319, 321, 323, 326, 327, 328, 330, 334, 337, 338,
       339, 340, 341, 343, 344, 345, 346, 349, 350, 351, 352, 353, 354,
       356, 357, 358, 359, 366, 367, 368, 370, 371, 372, 373, 374, 375,
       376, 378, 379, 380, 381, 384, 385, 386, 387, 388, 390, 391, 392,
       399, 400, 401, 402, 403, 404, 405, 408, 409, 410, 412, 413, 414,
       415, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431,
       436, 437, 439, 440, 441, 442, 443, 445, 446, 447, 448, 449, 450,
       451, 452, 453, 457, 458, 459, 460, 461, 464, 465, 466, 467, 477,
       478, 479, 480, 481, 482, 538, 539, 555, 556, 557, 558, 559, 560,
       561, 563, 564, 567, 568, 571, 572, 573, 580, 581, 582, 583, 584,
       585, 586, 587, 588, 589, 590, 597, 598, 600, 601, 602, 603, 606,
       607, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624,
       627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 640, 641,
       642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654,
       655, 656, 657, 658, 659, 662, 664, 670, 671, 672, 673, 674, 675,
       676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688,
       689, 691, 692, 693, 694, 695, 696, 701, 702, 703, 704, 708, 709,
       711, 712, 713, 714, 715, 717, 718, 719, 720, 721, 722, 723, 724,
       725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 738,
       739, 740, 743, 744, 751, 752, 753, 754, 755, 756, 757, 758, 759,
       760, 761, 762, 763, 765, 774, 775, 776, 777, 778, 779, 780, 781,
       782, 784, 785, 786, 787, 788, 789, 793, 798, 801, 802, 813, 814,
       818, 819, 820, 830, 831, 835, 836, 837, 838, 840, 841, 847, 848,
       849, 850, 851, 852, 853, 855, 856, 857, 866, 867, 868, 869, 870,
       873, 874, 875, 876, 877, 878, 879, 882, 898, 899, 904, 907, 908,
       919, 920, 923, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939,
       940, 941, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957,
       958, 959, 960, 961, 962, 963, 965, 966, 967, 968, 973, 974, 975,
       976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 989, 990, 991,
       992, 995, 996, 997, 998, 999], dtype=int64)
```

In [ ]:
```
data_trimmed = sorted_train_data.iloc[:,selected_names]
data_fnames = pd.DataFrame(sorted_train_data['filename'])
data_reduced = data_fnames.join(data_trimmed)
data_reduced.head()
```

Out[116]:

	filename	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23	ASM_2
1	01IsoiSMh5gxyDYTl4CB	116	120	116	9	32	32	32	3
4	01SuzwMJEIXsK7A8dQbl	69	68	69	48	9	9	13	1
0	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13	1
2	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13	1
3	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13	1

5 rows × 501 columns

In [ ]:
```
data_reduced.to_csv('sorted-features-asm-50percent.csv', index=False)
```

In [ ]:
```
data_reduced = pd.read_csv('sorted-features-asm-50percent.csv')
data_reduced.shape
```

Out[118]:
```
(10868, 501)
```

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

In [ ]:
```
data_reduced.shape
```

Out[120]:
```
(10868, 501)
```

In [ ]:
```
data_reduced.head()
```

Out[121]:

	ID	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23	ASM_2
0	01IsoiSMh5gxyDYTl4CB	116	120	116	9	32	32	32	3
1	01SuzwMJEIXsK7A8dQbl	69	68	69	48	9	9	13	1
2	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13	1
3	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13	1
4	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13	1

5 rows × 501 columns

2.2 Implement asm image features + bytes uni-gram features

2.2.1 Merge asm image features + bytes uni-gram features

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

Out[122]:

	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	0.003
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000

5 rows × 758 columns

In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, re
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tr
```

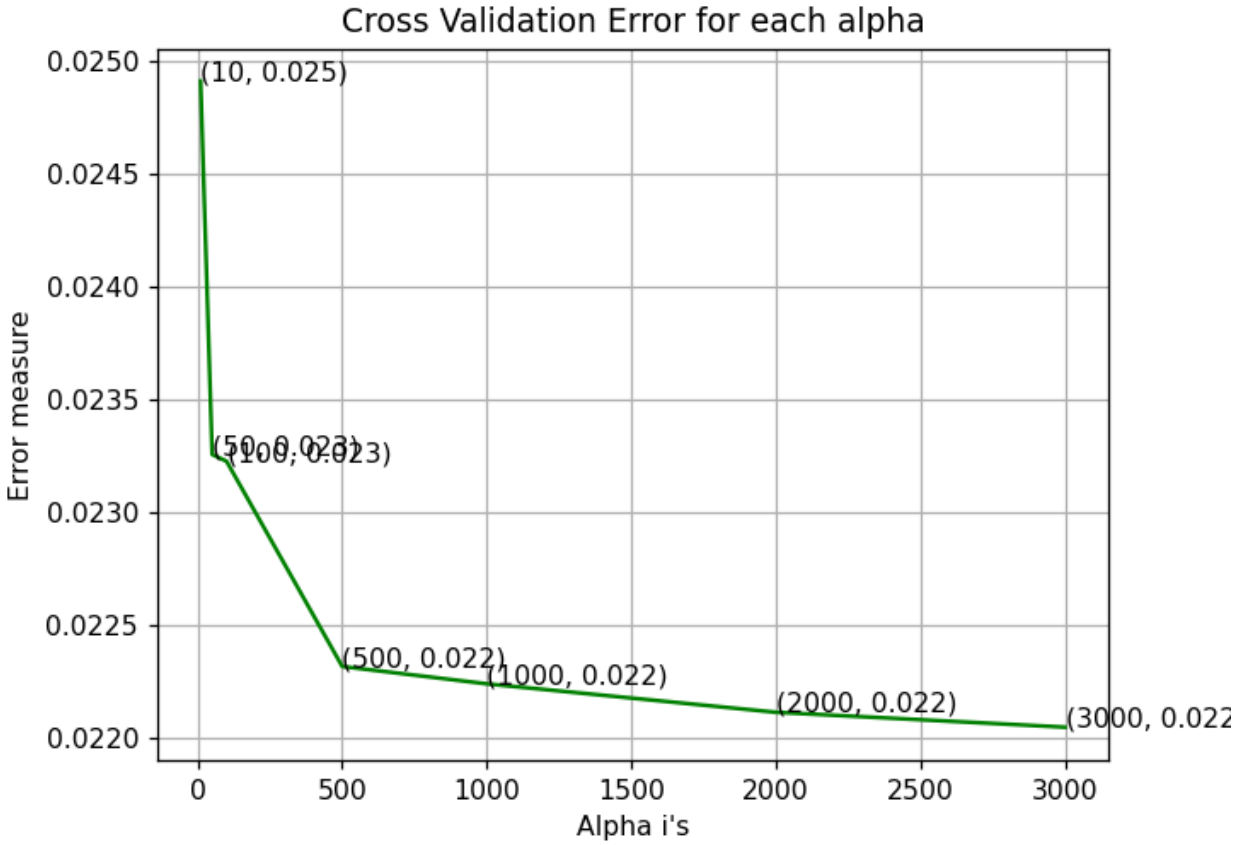
2.2.2 Random Forest Classifier asm image features + bytes uni-gram features

In [ ]:

```
%%time
plt.close()
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# ------------------------------

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.c

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()
```

```
log_loss for c =  10 is 0.024914350899232596
log_loss for c =  50 is 0.023255640449964945
log_loss for c =  100 is 0.023225030837972133
log_loss for c =  500 is 0.022313746822626396
log_loss for c =  1000 is 0.0222366382799315
```

```
log_loss for c =  2000 is 0.02211023310915517
log_loss for c =  3000 is 0.02204412565712111

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




```
Wall time: 5min 56s
Parser   : 107 ms
```

2.2.3 Random Forest Classifier (Best Hypher Parameters) asm image features + bytes uni-gram features

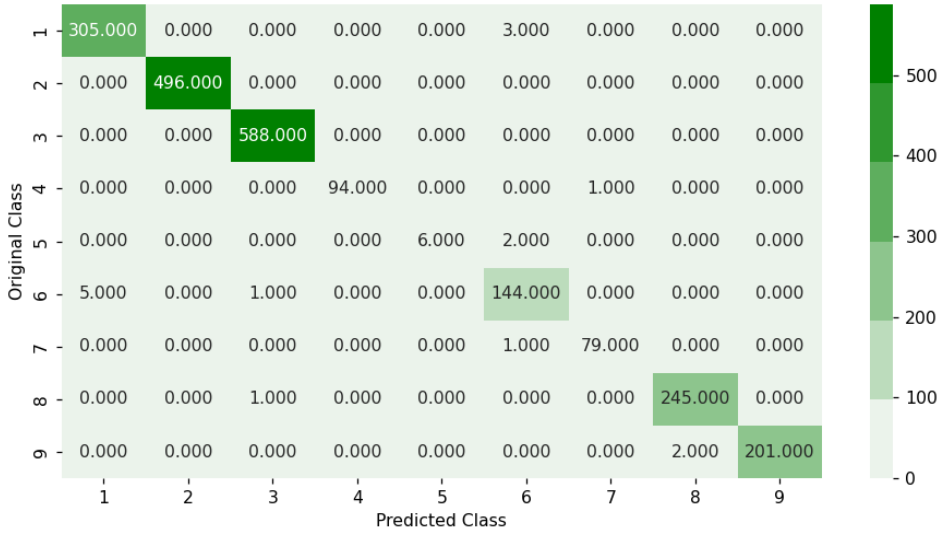
In [ ]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

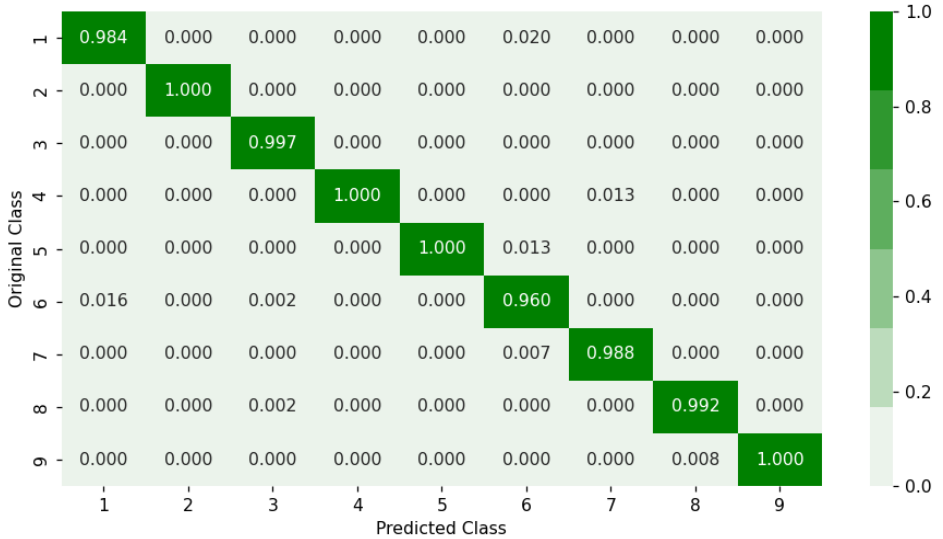
```
For values of best alpha =  3000 The train log loss is: 0.01384218603922668
For values of best alpha =  3000 The cross validation log loss is: 0.0220441256
5712111
For values of best alpha =  3000 The test log loss is: 0.03499782453759827
Wall time: 2min 37s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

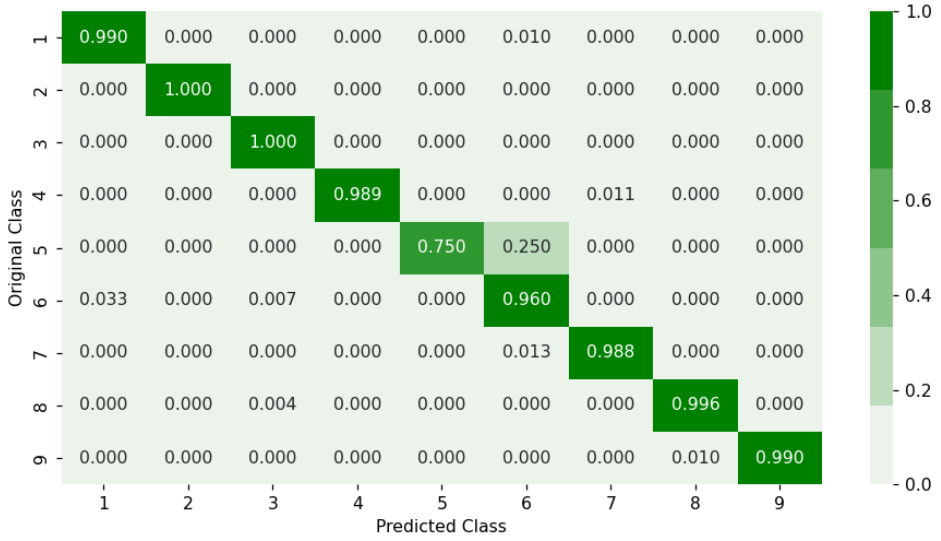
```
Number of misclassified points  0.7359705611775529
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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



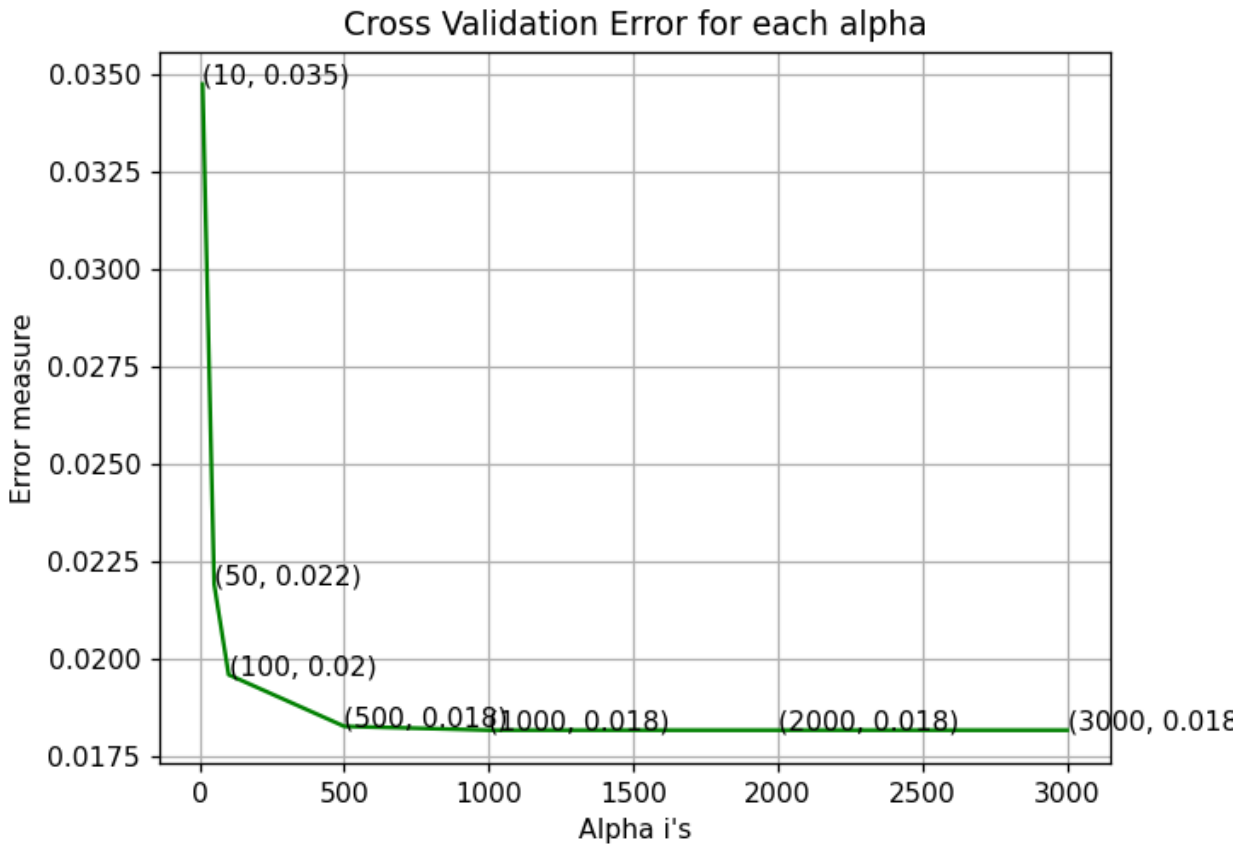

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

2.2.4 Xgboost Classifier asm image features + bytes uni-gram features

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

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

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()
```

```
log_loss for c =  10 is 0.034740347984345235
log_loss for c =  50 is 0.021922688962448953
log_loss for c =  100 is 0.019601842860830214
log_loss for c =  500 is 0.018277947960356852
log_loss for c =  1000 is 0.018180486467147844
log_loss for c =  2000 is 0.018180049167287687
log_loss for c =  3000 is 0.01817986350479278

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




```
Wall time: 49min 25s
```

In [ ]:

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)
predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

```
For values of best alpha =  3000 The train log loss is: 0.013841605706699447
For values of best alpha =  3000 The cross validation log loss is: 0.0181798635
0479278
For values of best alpha =  3000 The test log loss is: 0.03973994291747429
Wall time: 17min 39s
```

2.2.5 Xgboost Classifier (Best Hyper Parameters) asm image features + bytes uni-gram features

In [ ]:
```
x_cfl=XGBClassifier(nthread=-1,eval_metric='merror')

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=20,n_j
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[129]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=20)
```

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

```
{'subsample': 0.3, 'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.2,
'colsample_bytree': 0.5}
```

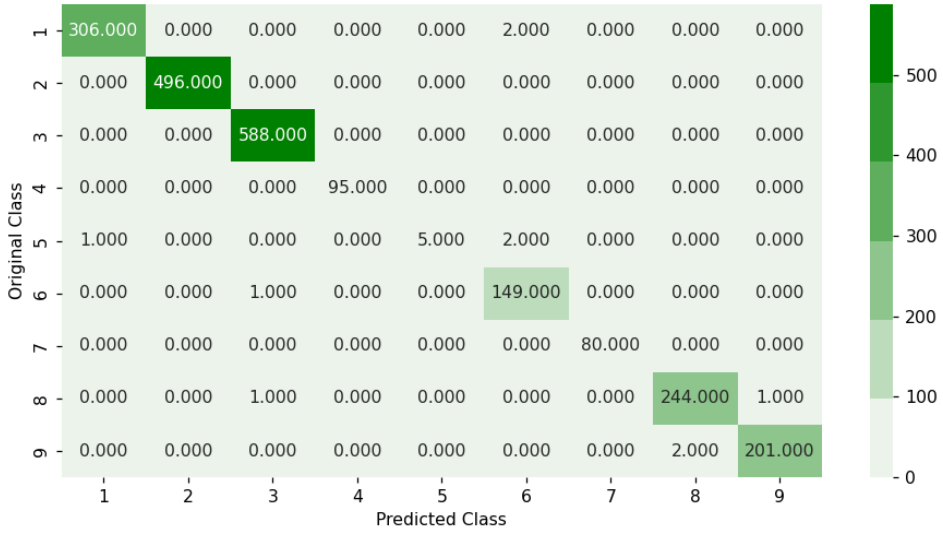
In [ ]:

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# ----------------------

x_cfl=XGBClassifier(n_estimators=random_cfl.best_params_['n_estimators'],max
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

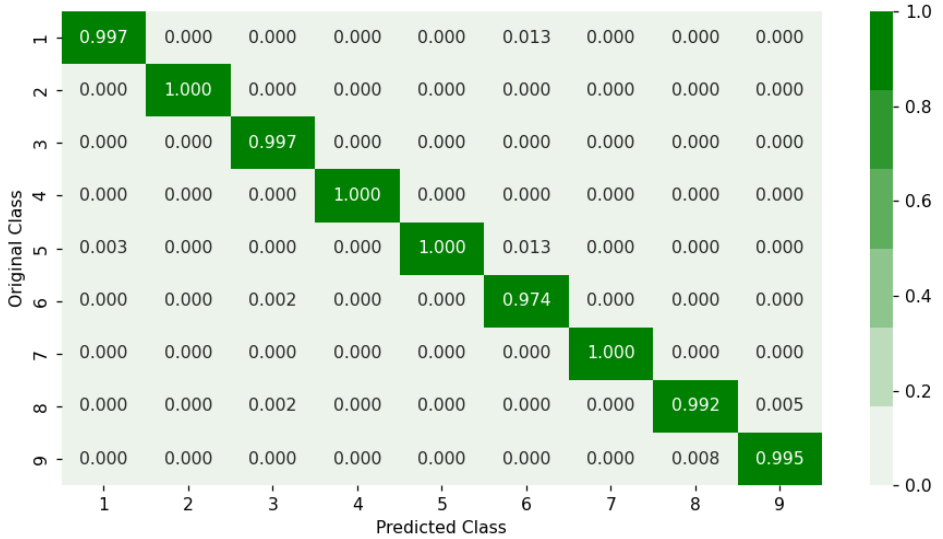
```
For values of best alpha =  3000 The train log loss is: 0.012780609169169867
For values of best alpha =  3000 The cross validation log loss is: 0.0158021993
48709006
For values of best alpha =  3000 The test log loss is: 0.031078466479681408
Wall time: 2min 34s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

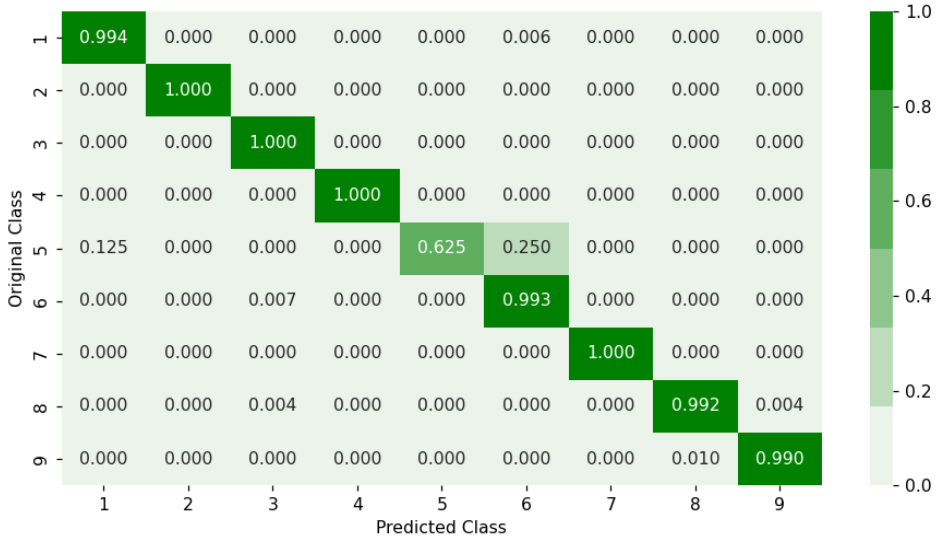
```
Number of misclassified points  0.45998160073597055
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

2.2.6 Conclusion and Model Comparision (asm image features + bytes uni-gram features)

In [4]:
```
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV

table.add_row(['Random Forest Classifier', 3000,0.01384218603922668, 0.02204
table.add_row(['Xgboost Classifier ', 3000, 0.012780609169169867,0.015802199

print(table)
```

```
+--------------------------+--------------------+----------------------+------
----------------+----------------------+--------------------------------+
|          Model           | Best HyperParameter |    Train Log Loss    |     C
V Log Loss      |    Test Log Loss     | Number of Misclassified Points |
+--------------------------+--------------------+----------------------+------
----------------+----------------------+--------------------------------+
| Random Forest Classifier |         3000        | 0.01384218603922668  | 0.022
04412565712111  | 0.03499782453759827  |       0.7359705611775529       |
|   Xgboost Classifier     |         3000        | 0.012780609169169867 | 0.015
802199348709006 | 0.031078466479681408 |      0.45998160073597055       |
+--------------------------+--------------------+----------------------+------
----------------+----------------------+--------------------------------+
```

2.3 Implement asm unigram + asm extracted image features

2.3.1 Merge asm unigram + asm extracted image features

In [ ]:
```
print(data_reduced.shape)
print(result_asm.shape)
```

```
(10868, 501)
(10868, 55)
```

In [ ]:
```
result_asm.columns
```

Out[135]:
```
Index(['ID', 'HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:',
       '.rdata:', '.edata:', '.rsrc:', '.tls:', '.reloc:', '.BSS:', '.CODE',
       'jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
       'dec', 'add', 'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror',
       'rol', 'jnb', 'jz', 'rtn', 'lea', 'movzx', '.dll', 'std::', ':dword',
       'edx', 'esi', 'eax', 'ebx', 'ecx', 'edi', 'ebp', 'esp', 'eip', 'Class',
       'size_x', 'size_y'],
      dtype='object')
```

In [ ]:
```
data_reduced.columns
```

Out[136]:
```
Index(['ID', 'ASM_1', 'ASM_3', 'ASM_4', 'ASM_14', 'ASM_20', 'ASM_21', 'ASM_23',
       'ASM_24', 'ASM_25',
       ...
       'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_994',
       'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
      dtype='object', length=501)
```

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

Out[137]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	ASM
0	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	0.0	...	
1	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	0.0	...	
2	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	0.0	...	
3	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	0.0	...	
4	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	0.0	...	

5 rows × 550 columns

In [ ]:
```
result_x.columns
```

Out[138]:
```
Index(['HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:', '.rdata:',
       '.edata:', '.rsrc:', '.tls:',
       ...
       'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_994',
       'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
      dtype='object', length=550)
```

In [ ]:
```
print(result_x.shape)
```

```
(10868, 550)
```

In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, re
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tr
```

2.3.2 Random Forest Classifier with asm unigram + asm extracted image features

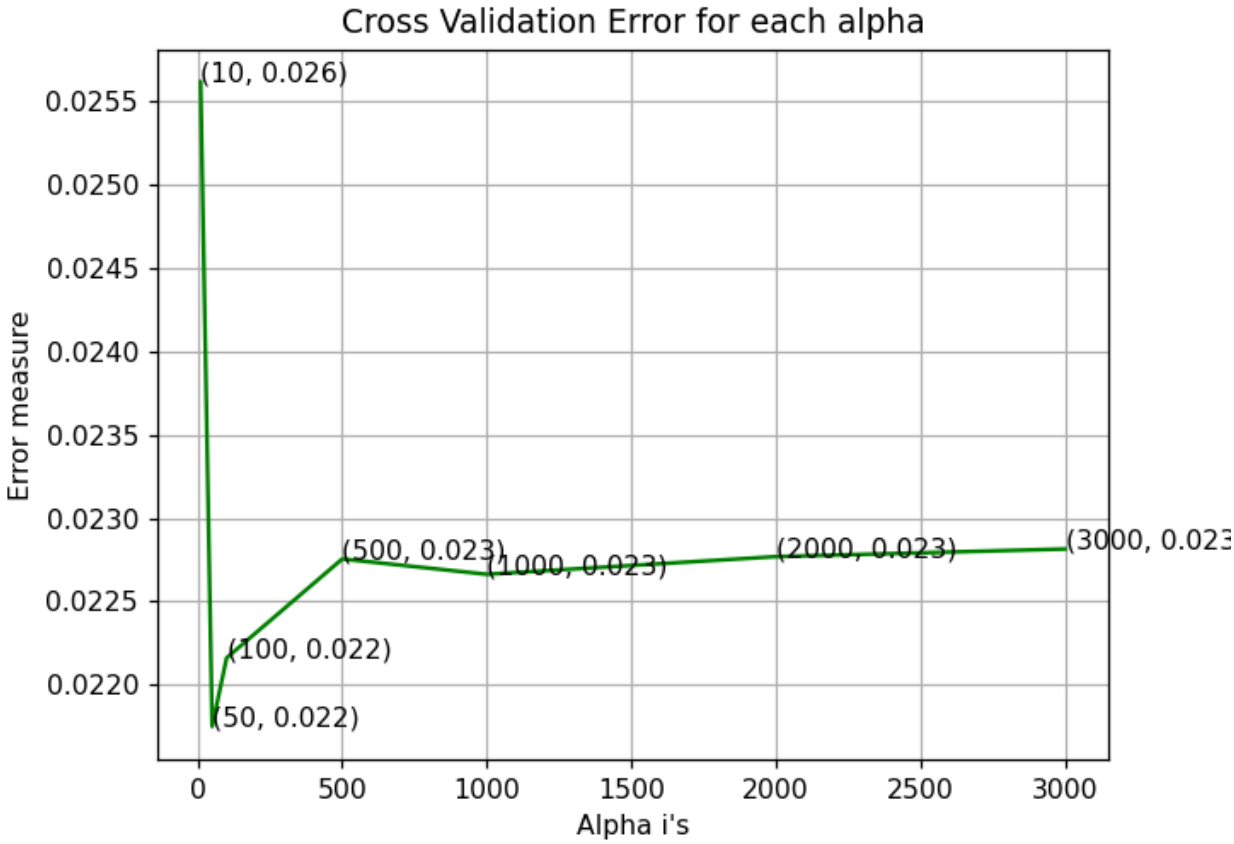
In [ ]:

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# --------------------------------
from tqdm import tqdm
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    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.c

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()
```

```
100%|████████████████████████████████████████████████████████████████████
████████████████████| 7/7 [01:53<00:00, 16.23s/it]

log_loss for c =  10 is 0.025619244065729782
log_loss for c =  50 is 0.021750408227904403
log_loss for c =  100 is 0.022161966373998608
```

```
log_loss for c =  500 is 0.02275559634480615
log_loss for c =  1000 is 0.022663581487286026
log_loss for c =  2000 is 0.022769458938220317
log_loss for c =  3000 is 0.02281532992188414
```

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




```
Wall time: 1min 54s
```

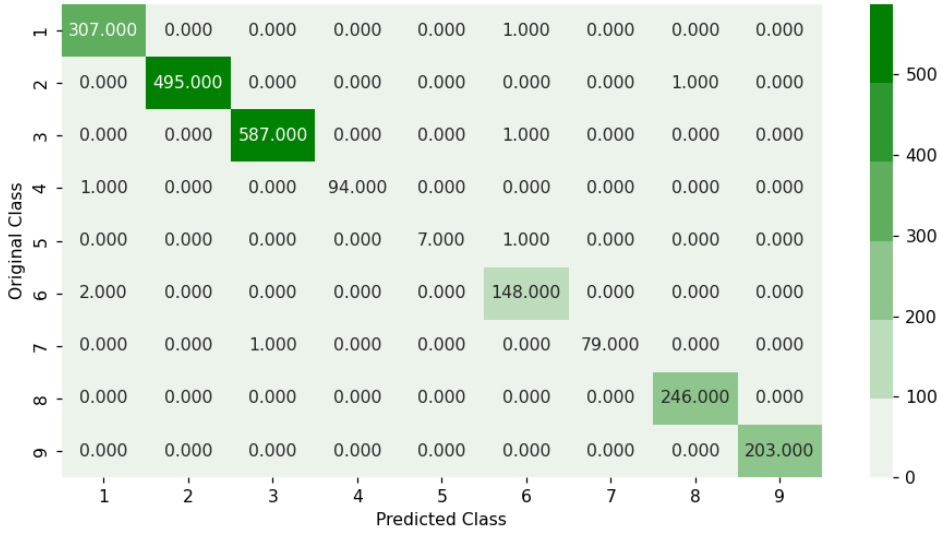
In [ ]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

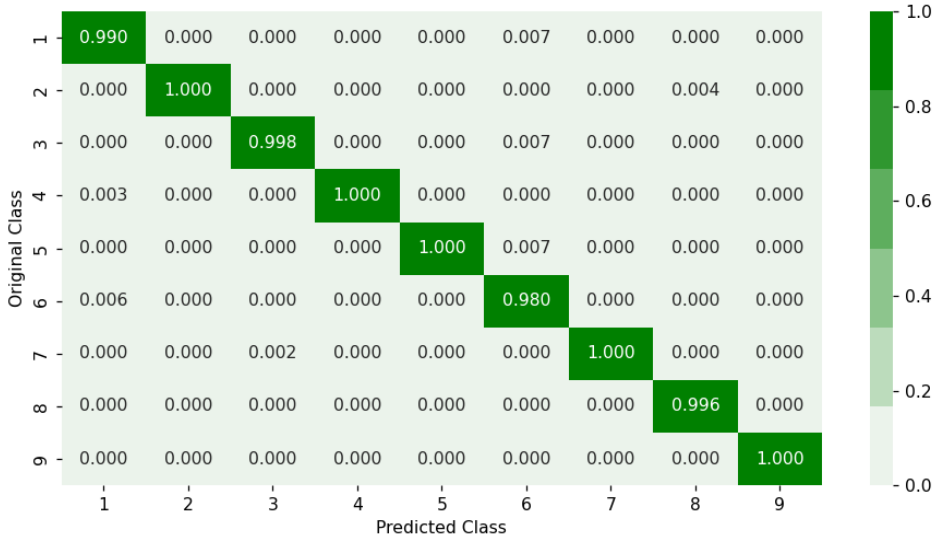
```
For values of best alpha =  50 The train log loss is: 0.012188588334491473
For values of best alpha =  50 The cross validation log loss is: 0.021750408227
904403
For values of best alpha =  50 The test log loss is: 0.02254262292519041
Wall time: 3.49 s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

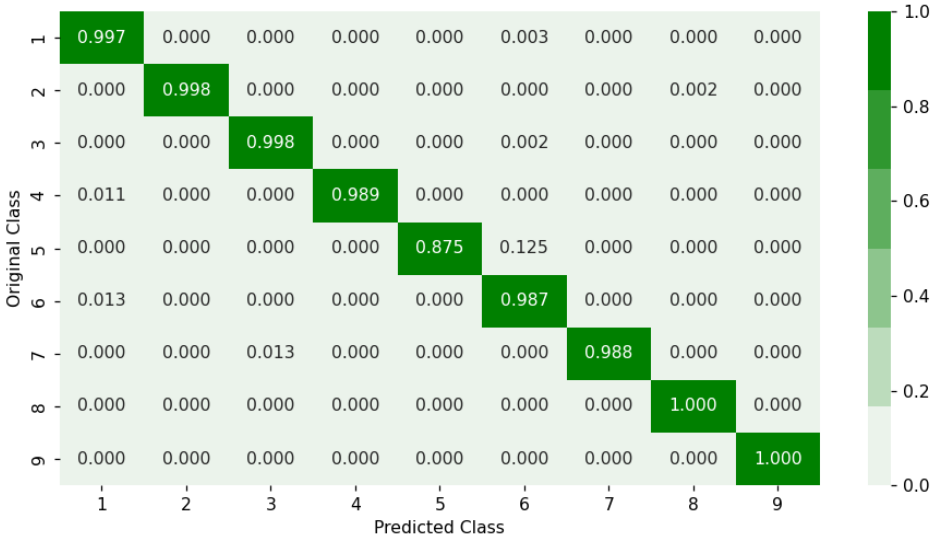
```
Number of misclassified points  0.36798528058877644
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

2.3.3 Xgboost implementation with asm unigram + asm extracted image features

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

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

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)
```

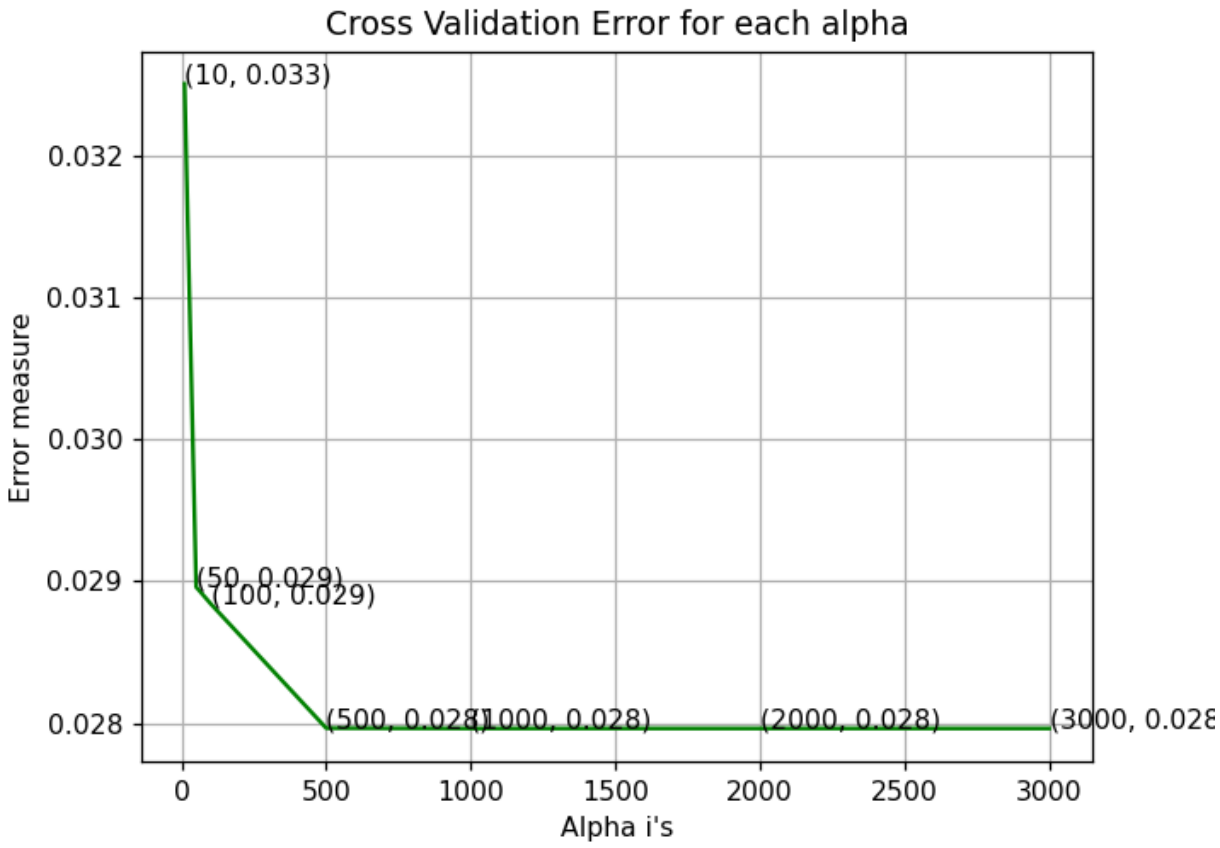
```
log_loss for c =  10 is 0.032502752150702766
log_loss for c =  50 is 0.028959652510518522
log_loss for c =  100 is 0.028842055594983925
log_loss for c =  500 is 0.027965537119622894
log_loss for c =  1000 is 0.027963417770606337
log_loss for c =  2000 is 0.02796351484301509
log_loss for c =  3000 is 0.02796348042503413

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

```
Wall time: 28min 59s
```

In [ ]:

```
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()
```

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



In [ ]:

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  1000 The train log loss is: 0.011593187131160507
For values of best alpha =  1000 The cross validation log loss is: 0.0279634177
70606337
For values of best alpha =  1000 The test log loss is: 0.021801794278533182
Wall time: 4min 8s
```

2.3.4 Xgboost implementation (Best HyperParameter) with asm unigram + asm extracted image features

In [ ]:
```
x_cfl=XGBClassifier(nthread=-1,eval_metric='merror')

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_j
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[150]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=10)
```

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

```
{'subsample': 0.5, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.1,
'colsample_bytree': 0.5}
```

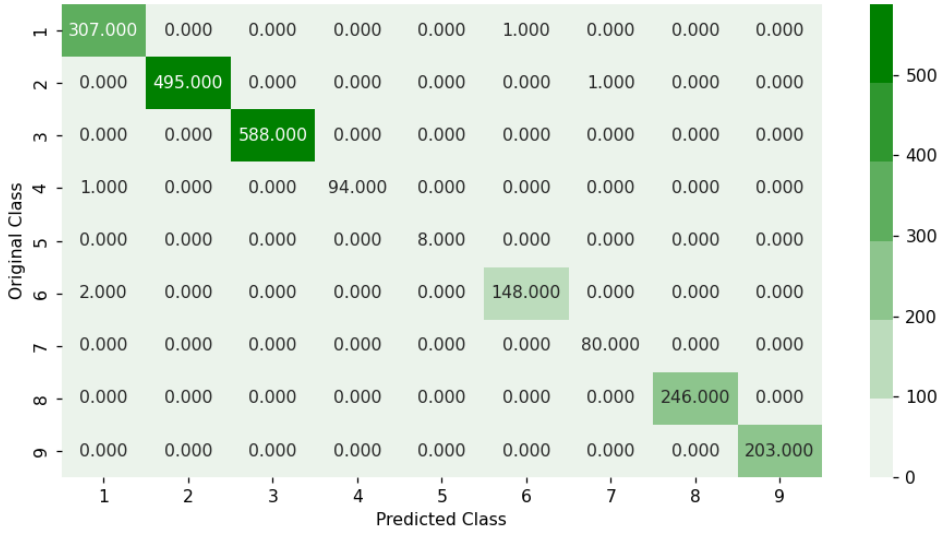
In [ ]:

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# ----------------------

x_cfl=XGBClassifier(n_estimators=random_cfl.best_params_['n_estimators'],max
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

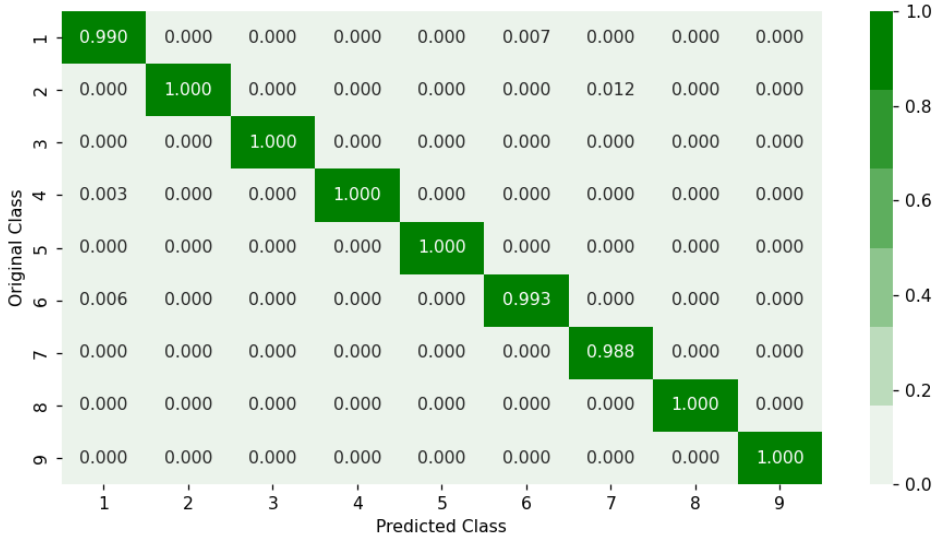
```
For values of best alpha =  1000 The train log loss is: 0.010556986389754578
For values of best alpha =  1000 The cross validation log loss is: 0.0259843805
0268895
For values of best alpha =  1000 The test log loss is: 0.022241728555480386
Wall time: 2min 25s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

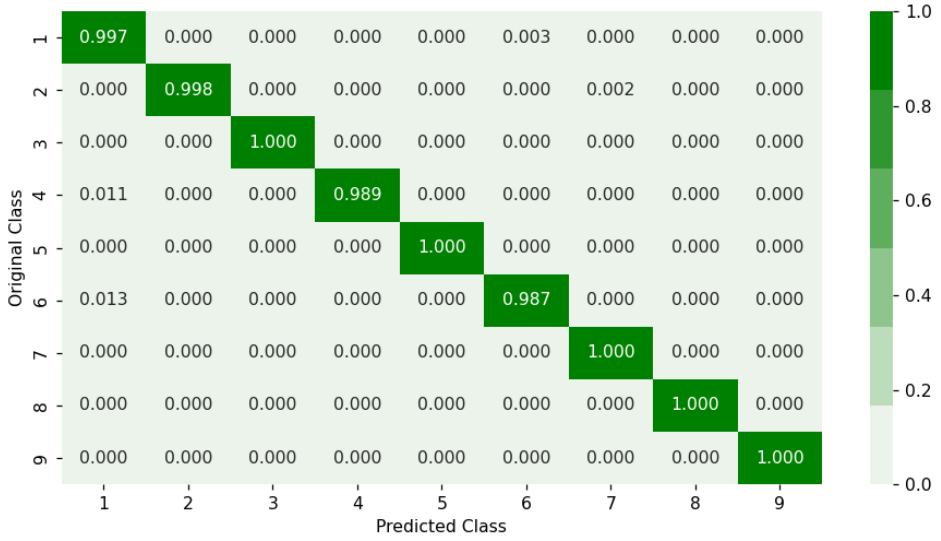
```
Number of misclassified points  0.22999080036798528
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

2.3.5 Conclusion and Model Comparision (asm unigram + asm extracted image features)

In [5]:
```
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV

table.add_row(['Random Forest Classifier', 50, 0.012188588334491473, 0.02175
table.add_row(['Xgboost Classifier', 500,0.010556986389754578,0.025984380502

print(table)
```

```
+--------------------------+--------------------+----------------------+------
----------------+----------------------+--------------------------------+
|          Model           | Best HyperParameter |    Train Log Loss    |     C
V Log Loss      |    Test Log Loss     | Number of Misclassified Points |
+--------------------------+--------------------+----------------------+------
----------------+----------------------+--------------------------------+
| Random Forest Classifier |          50         | 0.012188588334491473 | 0.021
750408227904403 | 0.02254262292519041  |      0.36798528058877644       |
|    Xgboost Classifier    |         500         | 0.010556986389754578 | 0.025
98438050268895  | 0.022241728555480386 |      0.22999080036798528       |
+--------------------------+--------------------+----------------------+------
----------------+----------------------+--------------------------------+
```

2.4 Implemented ASM unigram + ASM image features + ByteFile unigram

2.4.1 Prepare, Merge and Split (ASM unigram + ASM image features + ByteFile unigram) data

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

```
(10868, 501)
(10868, 55)
(10868, 261)
```

In [ ]:
```
data_reduced.columns
```

Out[155]:
```
Index(['ID', 'ASM_1', 'ASM_3', 'ASM_4', 'ASM_14', 'ASM_20', 'ASM_21', 'ASM_23',
       'ASM_24', 'ASM_25',
       ...
       'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_994',
       'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
      dtype='object', length=501)
```

In [ ]:
```
result_asm.columns
```

Out[156]:
```
Index(['ID', 'HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:',
       '.rdata:', '.edata:', '.rsrc:', '.tls:', '.reloc:', '.BSS:', '.CODE',
       'jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
       'dec', 'add', 'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror',
       'rol', 'jnb', 'jz', 'rtn', 'lea', 'movzx', '.dll', 'std::', ':dword',
       'edx', 'esi', 'eax', 'ebx', 'ecx', 'edi', 'ebp', 'esp', 'eip', 'Class',
       'size_x', 'size_y'],
      dtype='object')
```

In [ ]:
```
result.columns
```

Out[157]:
```
Index(['ID', '0', '1', '2', '3', '4', '5', '6', '7', '8',
       ...
       'fa', 'fb', 'fc', 'fd', 'fe', 'ff', '??', 'Unnamed: 0', 'size',
       'Class'],
      dtype='object', length=261)
```

In [ ]:
```
result_y
```

Out[158]:
```
0        1
1        1
2        1
3        1
4        1
        ..
10863    2
10864    2
10865    2
10866    2
10867    2
Name: Class, Length: 10868, dtype: int64
```

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

Out[159]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	
0	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	0.0	...	0.00
1	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	0.0	...	0.00
2	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	0.0	...	0.00
3	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	0.0	...	0.00
4	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	0.0	...	0.00

5 rows × 810 columns

In [ ]:
```
result_x = result_x.drop(['Class_y'], axis=1)
```

In [ ]:
```
result_x.shape
```

Out[161]:
```
(10868, 809)
```

In [ ]:
```
result_x.columns[150]
```

Out[162]:
```
'ASM_272'
```

In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, re
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tr
```

2.4.2 Random Forest Classifier ASM unigram + ASM image features + ByteFile unigram

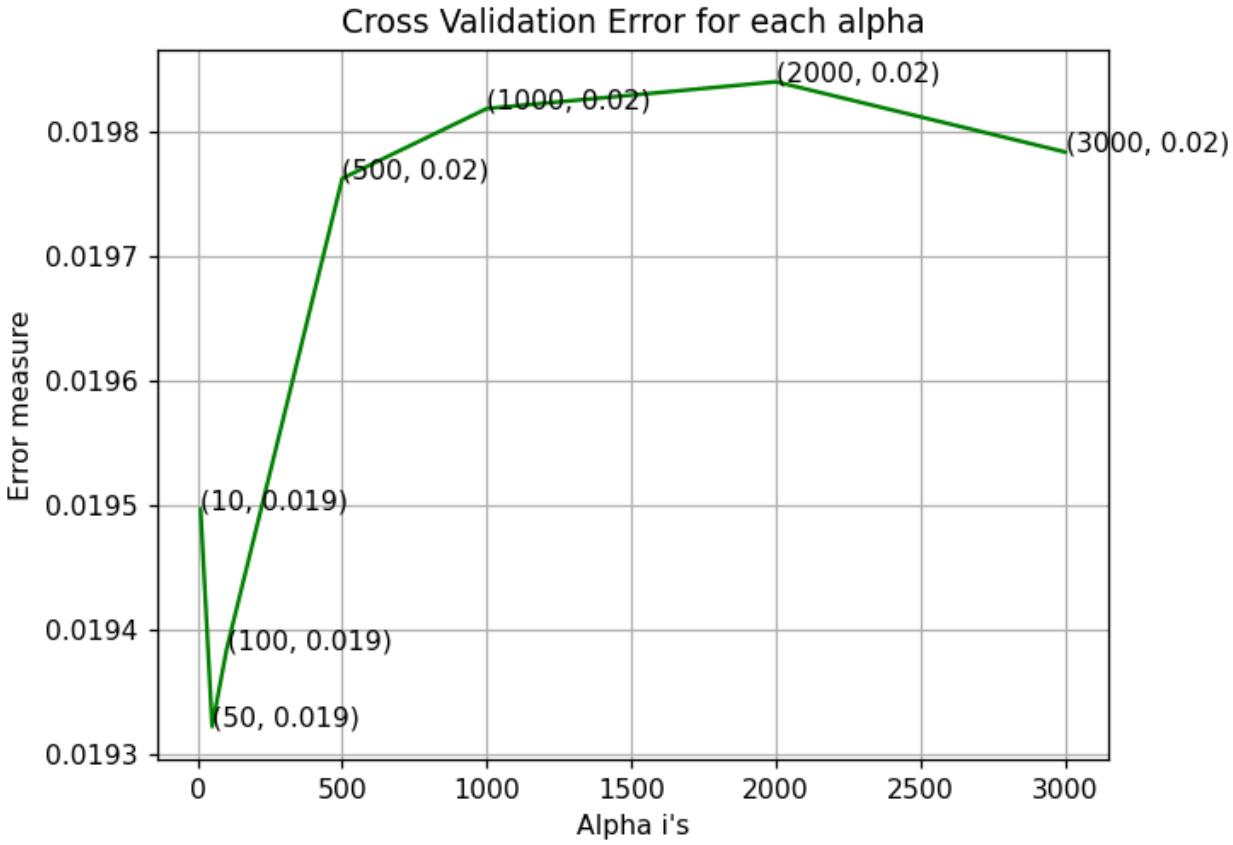
In [ ]:

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tra
# 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-onlin
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    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.c

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()
```

```
100%|██████████| 7/7 [03:49<00:00, 32.81s/it]

log_loss for c =  10 is 0.019496099101500255
log_loss for c =  50 is 0.019321749449855537
log_loss for c =  100 is 0.019383650130507878
```

```
log_loss for c =  500 is 0.019761818499614513
log_loss for c =  1000 is 0.019817942650659646
log_loss for c =  2000 is 0.01983946525949842
log_loss for c =  3000 is 0.019783056509931767
```

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




```
Wall time: 3min 49s
```

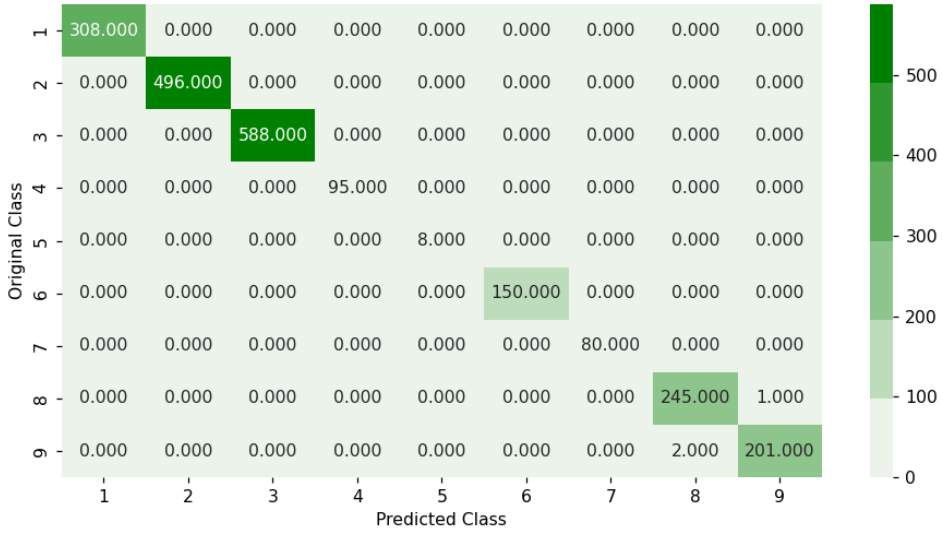
In [ ]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatio
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
```

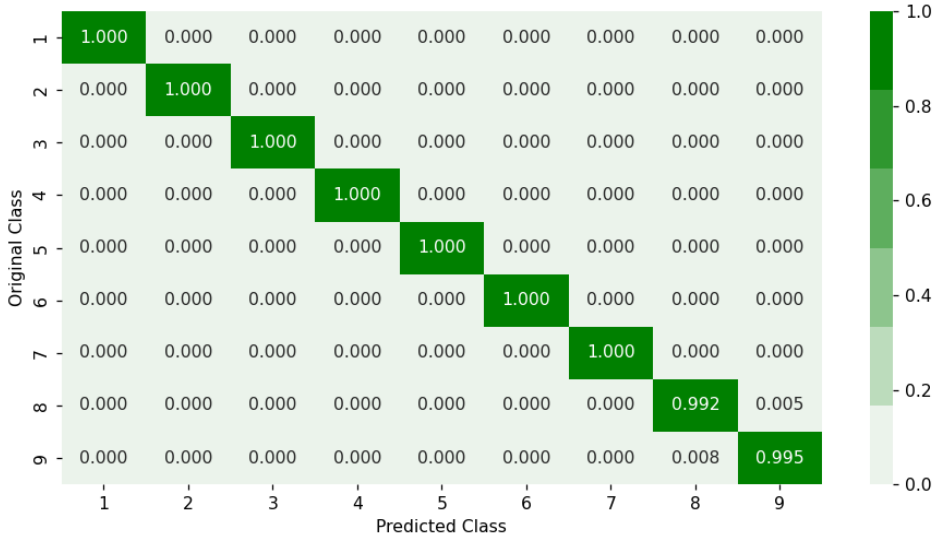
```
For values of best alpha =  50 The train log loss is: 0.010837297758792103
For values of best alpha =  50 The cross validation log loss is: 0.019321749449
855537
For values of best alpha =  50 The test log loss is: 0.014657795456301009
Wall time: 4.68 s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

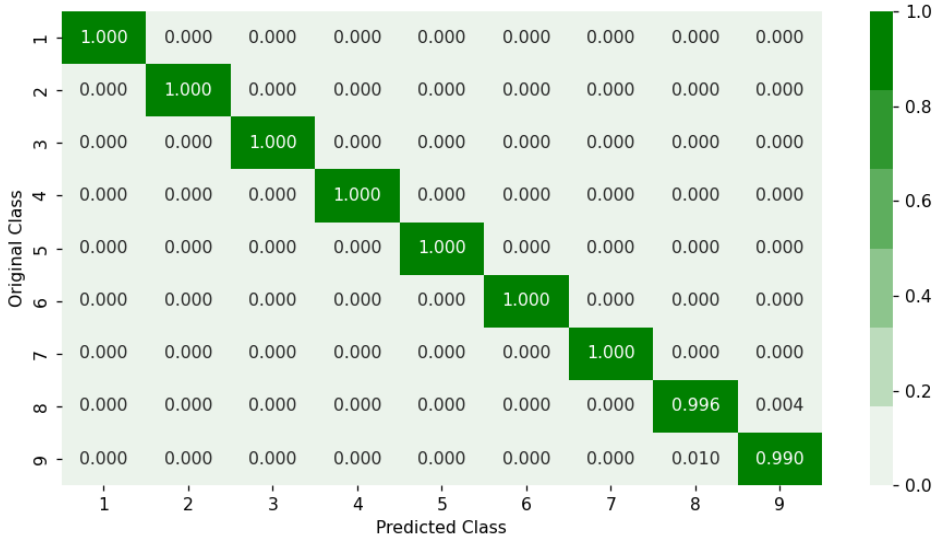
```
Number of misclassified points  0.13799448022079117
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

2.4.3 XgBoost Classifier ASM unigram + ASM image features + ByteFile unigram

In [ ]:

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# -----------------------

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

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)


```

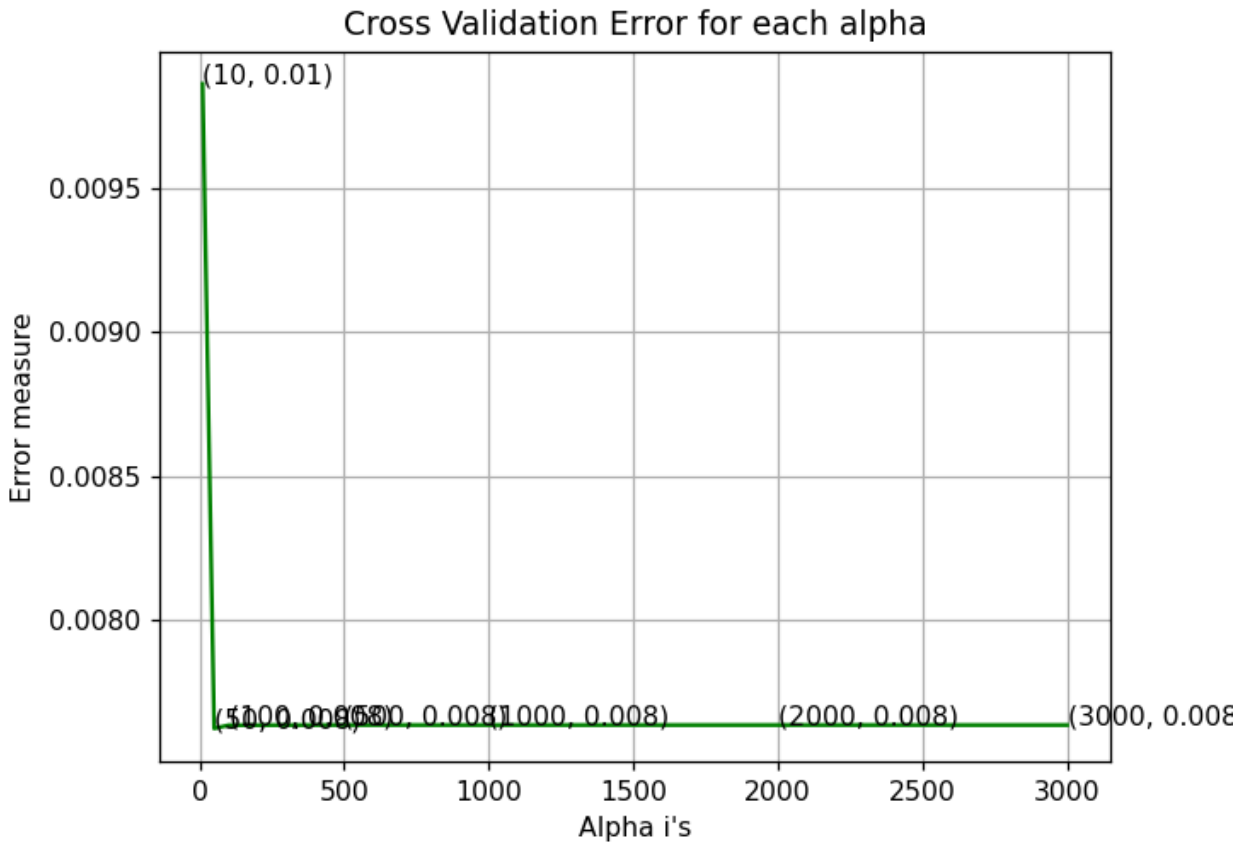
```
100%|████████████████████████████████████████████████████████████████████████████| 7/7 [39:20<00:00, 337.26s/it]

log_loss for c =  10 is 0.009861487489183307
log_loss for c =  50 is 0.007622889355136446
log_loss for c =  100 is 0.007634168901908475
log_loss for c =  500 is 0.007634720772572728
log_loss for c =  1000 is 0.007634663445961885
log_loss for c =  2000 is 0.0076347357067424095
log_loss for c =  3000 is 0.0076346686537671115


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

```
Wall time: 39min 20s
```

In [ ]:

```
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()
```

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



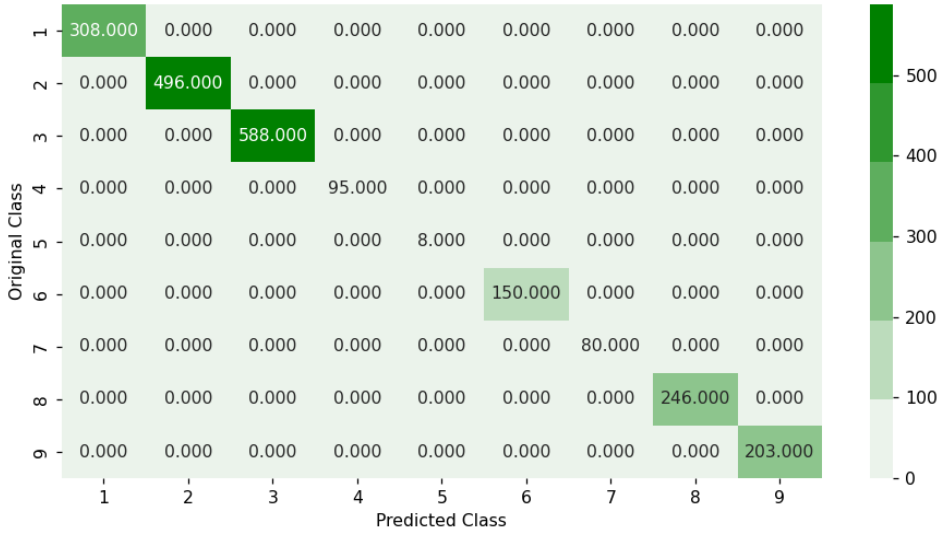
In [ ]:

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1,eval_metric='m
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

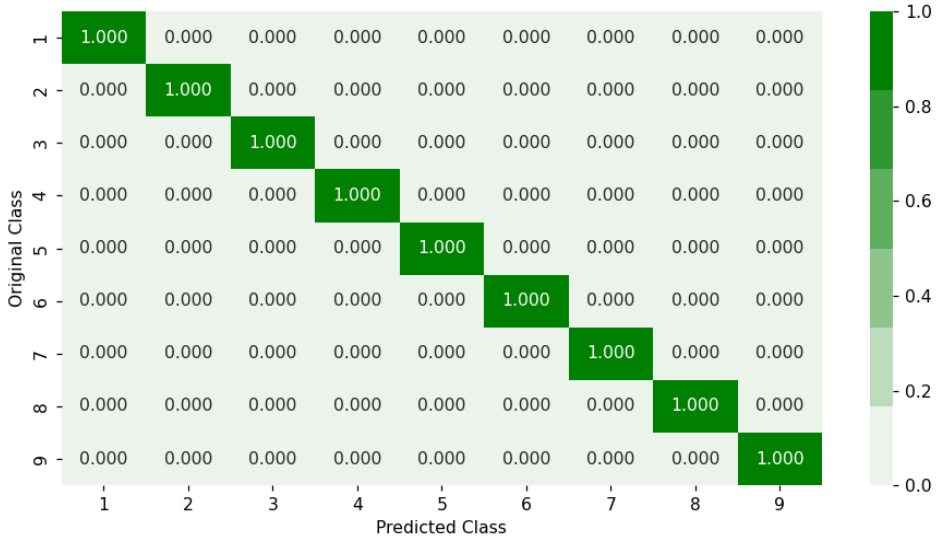
```
For values of best alpha =  50 The train log loss is: 0.007765535480565201
For values of best alpha =  50 The cross validation log loss is: 0.007622889355
136446
For values of best alpha =  50 The test log loss is: 0.007591880189546484
Wall time: 1min 20s
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

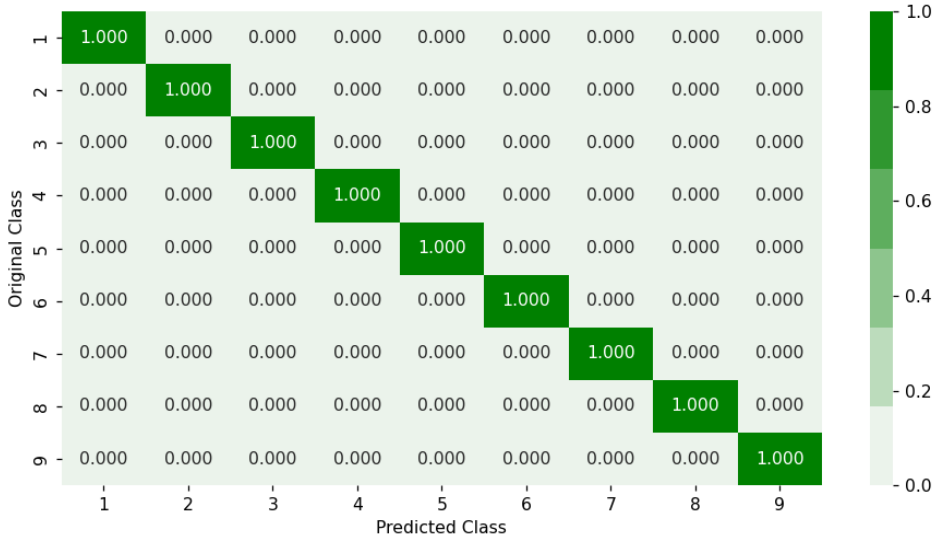
```
Number of misclassified points  0.0
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

2.4.5 XgBoost Classifier (Best HyperParameter) ASM unigram + ASM image features + ByteFile unigram

In [ ]:
```
x_cfl=XGBClassifier(eval_metric='merror')

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_j
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[173]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False,
                                           eval_metric='merror', gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           mono...
                                           predictor=None, random_state=None,
                                           reg_alpha=None, reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.
1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   verbose=10)
```

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

```
{'subsample': 0.3, 'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.2,
'colsample_bytree': 0.5}
```

In [ ]:

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=1
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gam
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, re
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=Non

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopp
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOT
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onli
# ----------------------

x_cfl=XGBClassifier(n_estimators=random_cfl.best_params_['n_estimators'],max
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)
```

```
Wall time: 2min 9s
```

Out[177]:

```
CalibratedClassifierCV(base_estimator=XGBClassifier(base_score=0.5,
                                                    booster='gbtree',
                                                    colsample_bylevel=1,
                                                    colsample_bynode=1,
                                                    colsample_bytree=0.5,
                                                    enable_categorical=False,
                                                    eval_metric='merror',
                                                    gamma=0, gpu_id=-1,
                                                    importance_type=None,
                                                    interaction_constraints='',
                                                    learning_rate=0.2,
                                                    max_delta_step=0,
                                                    max_depth=10,
                                                    min_child_weight=1,
                                                    missing=nan,
                                                    monotone_constraints='()',
                                                    n_estimators=500, n_jobs=8,
                                                    nthread=-1,
                                                    num_parallel_tree=1,
                                                    objective='multi:softprob',
                                                    predictor='auto',
                                                    random_state=0, reg_alpha=
0,
                                                    reg_lambda=1,
                                                    scale_pos_weight=None,
                                                    subsample=1,
                                                    tree_method='exact',
                                                    validate_parameters=1,
...))
```

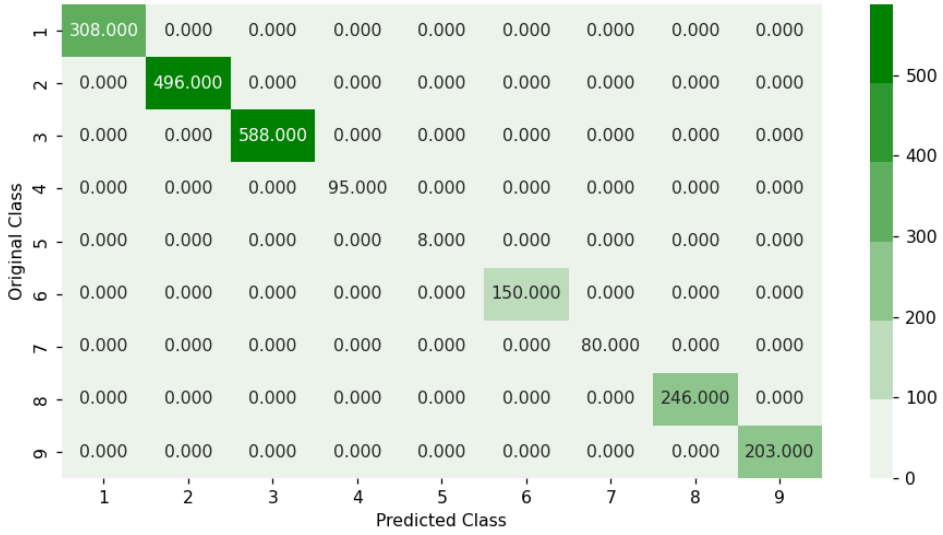
In [ ]:

```
predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', random_cfl.best_params_['n_estimators'
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ',random_cfl.best_params_['n_estimators'],
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', random_cfl.best_params_['n_estimators']
```

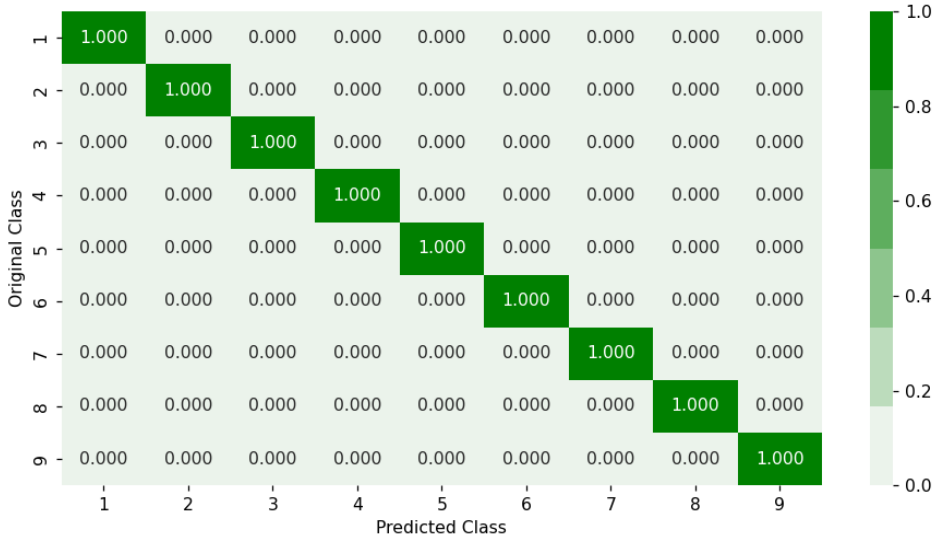
```
For values of best alpha =  500 The train log loss is: 0.007656023268911571
For values of best alpha =  500 The cross validation log loss is: 0.00750239659
4345622
For values of best alpha =  500 The test log loss is: 0.007433716266945984
```

In [ ]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

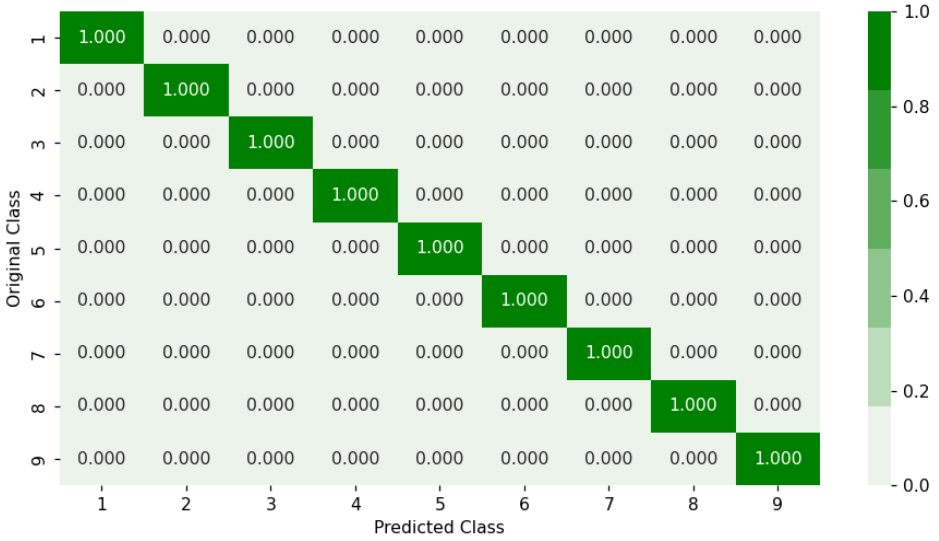
```
Number of misclassified points  0.0
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

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




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

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




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

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




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

2.4.6 Conclusion and Model Comparision (ASM unigram + ASM image features + ByteFile unigram)

In [6]:

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV

table.add_row(['Random Forest Classifier', 50, 0.010837297758792103,0.019321
table.add_row(['XgBoost Classifier', 50, 0.007765535480565201,0.007622889355
table.add_row(['XgBoost Classifier With Best HyperParameter', 500, 0.0076560

print(table)
```

```
+-------------------------------------------+--------------------+----------
------------+---------------------+---------------------+--------------------
------------+
|                   Model                   | Best HyperParameter |    Train
Log Loss    |     CV Log Loss     |    Test Log Loss    | Number of Misclassi
fied Points |
+-------------------------------------------+--------------------+----------
------------+---------------------+---------------------+--------------------
------------+
|           Random Forest Classifier        |         50         | 0.0108372
97758792103 | 0.019321749449855537 | 0.014657795456301009 |      0.137994480220
79117       |
|              XgBoost Classifier           |         50         | 0.0077655
35480565201 | 0.007622889355136446 | 0.007591880189546484 |              0.0
|
| XgBoost Classifier With Best HyperParameter |        500         | 0.0076560
23268911571 | 0.007502396594345622 | 0.007433716266945984 |              0.0
|
+-------------------------------------------+--------------------+----------
------------+---------------------+---------------------+--------------------
------------+
```

- Conclusion
  - I followed following feature engineering approach:
  - Tried Bigram of ByteFiles
  - ASM image features + bytes uni-gram features
  - ASM unigram + ASM extracted image features
  - ASM unigram + ASM image features + ByteFile unigram
  - **ASM unigram + ASM image features + ByteFile unigram (XgBoost Classifier With Best HyperParameter) reduced the log-loss < 0.01 (0.007433716266945984)**