

# 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 its 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 finish 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
g, 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@n@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08    BB 42 00                  mov     dword ptr [esi], offset off_42BB08
.text:00401013 8B C6                                mov     eax, esi
.text:00401015 5E                                pop    esi
.text:00401016 C2 04 00                  retn    4
.text:00401016                                ; -----
-----
.text:00401019 CC CC CC    CC CC CC CC          align 10h
.text:00401020 C7 01 08    BB 42 00                  mov     dword ptr [cx], offset off_42BB08
.text:00401026 E9 26 1C    00 00                  jmp    sub_402C51
.text:00401026                                ; -----
-----
.text:0040102B CC CC CC    CC CC                align 10h
.text:00401030 56                                push    esi
.text:00401031 8B F1                                mov     esi, ecx
.text:00401033 C7 06 08    BB 42 00                  mov     dword ptr [esi], offset off_42BB08
.text:00401039 E8 13 1C    00 00                  call    sub_402C51
.text:0040103E F6 44 24    08 01                  test    byte ptr [esp+8], 1
.text:00401043 74 09                                jz     short loc_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                                loc_40104E: ; CODE
XREF: .text:00401043@j
.text:0040104E 8B C6                                mov     eax, esi
.text:00401050 5E                                pop    esi
.text:00401051 C2 04 00                  retn    4
.text:00401051                                ; -----

```

## .bytes file

```

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

```

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

### 2.2.1. Type of Machine Learning Problem

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

### 2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

## 2 2 3 Machine Learning Objectives and Constraints

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=VLQTRILGz5Y>

<https://github.com/dchad/malware-detection>

<http://vizsec.org/files/2011/Nataraj.pdf>

[https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu\\_pIB6ua?dl=0](https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0)

" Cross validation is more trustworthy than domain knowledge."

## 3. Exploratory Data Analysis

In [1]:

```

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

```

In [3]:

```

source = 'train'
destination = 'byteFiles'

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

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

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

```

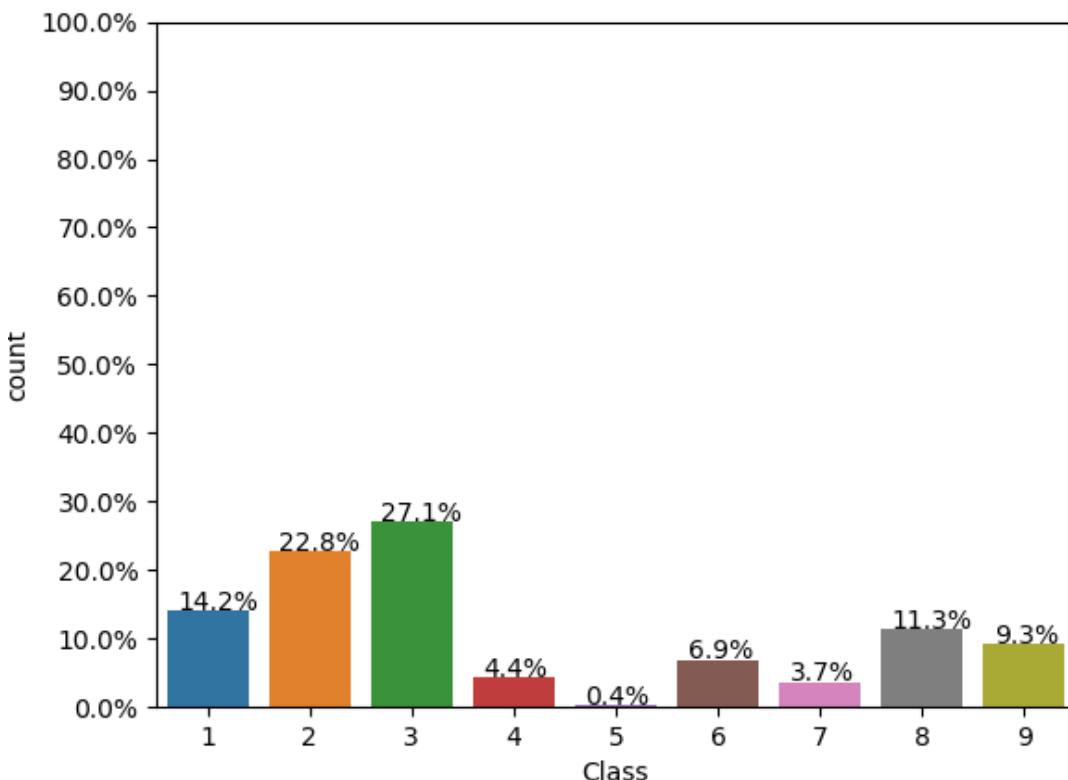
### 3.1. Distribution of malware classes in whole data set

In [2]:

```
Y=pd.read_csv("trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
    ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()))
#put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
ax.yaxis.set_ticks(np.linspace(0, total, 11))

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

&lt;IPython.core.display.Javascript object&gt;



## 3.2. Feature extraction

### 3.2.1 File size of byte files as a feature

In [3]:

```

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

```

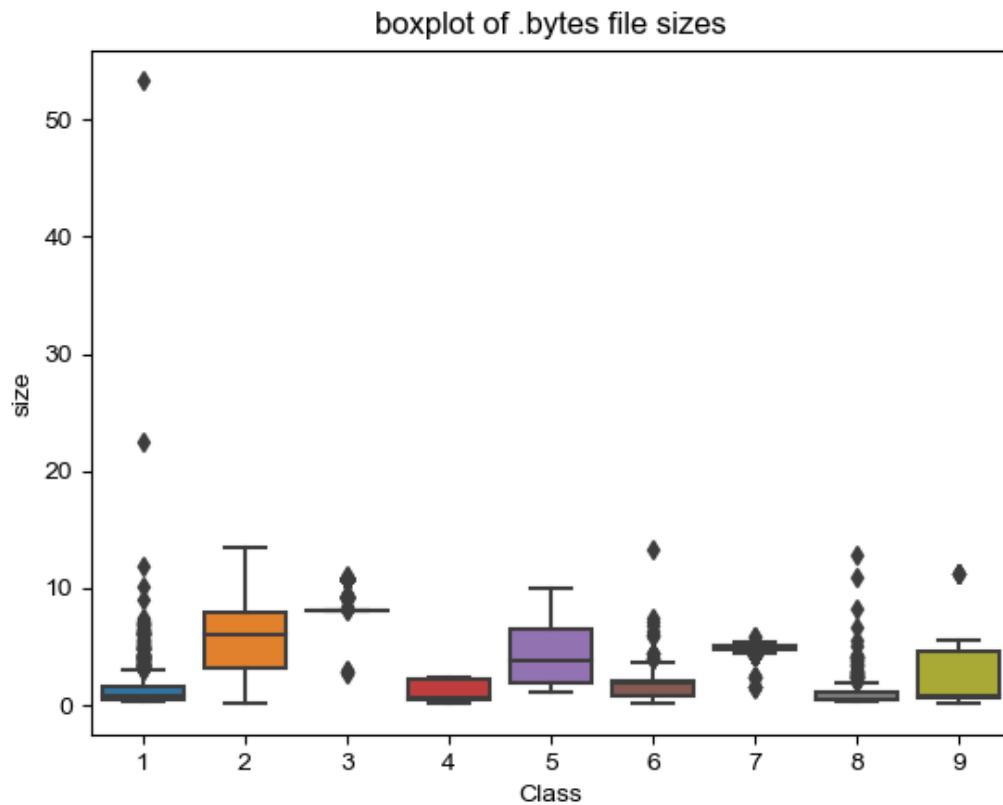
	ID	size	Class
0	01azqd4InC7m9JpocGv5	5.012695	9
1	01IsoiSMh5gxyDYTl4CB	6.556152	2
2	01jsnpXSAlg6aPeDxrU	4.602051	9
3	01kcPWA9K2B0xQeS5Rju	0.679688	1
4	01SuzwMJEIXsK7A8dQbl	0.438965	8

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

In [4]:

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

&lt;IPython.core.display.Javascript object&gt;



### 3.2.3 feature extraction from byte files

In [5]:

```
#removal of address 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+')
        file = file+'.bytes'
        with open('byteFiles/'+file,"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)
        text_file.close()

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

In [7]:

```
#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,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file, "r") as byte_file:
            for lines in byte_file:
                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_file.close()
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i)+",")
    byte_feature_file.write("\n")
k += 1

byte_feature_file.close()
```

mk

In [19]:

```
result = pd.merge(byte_features, data_size_byte, on='ID', how='left')
result.head()
```

Out[19]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...
0	0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...
1	1	01lsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	...
2	2	01jsnpXSAlg6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	...
3	3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	...
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	...

5 rows × 263 columns

In [4]:

```
byte_features=pd.read_csv("result.csv")
print (byte_features.head())
```

	Unnamed: 0	ID	0	1	2	3	4	5	\			
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242				
1	1	01IsoiSMh5gxyDYT14CB	39755	8337	7249	7186	8663	6844				
2	2	01jsnpXSAlgwaPeDxrU	93506	9542	2568	2438	8925	9330				
3	3	01kcPWA9K2B0xQeS5Rju	21091	1213	726	817	1257	625				
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410				
	6	7	...	f9	fa	fb	fc	fd	fe	ff	??	\
0	3650	3201	...	3101	3211	3097	2758	3099	2759	5753	1824	
1	8420	7589	...	439	281	302	7639	518	17001	54902	8588	
2	9007	2342	...	2242	2885	2863	2471	2786	2680	49144	468	
3	550	523	...	485	462	516	1133	471	761	7998	13940	
4	262	249	...	350	209	239	653	221	242	2199	9008	
	size	Class										
0	5.012695	9										
1	6.556152	2										
2	4.602051	9										
3	0.679688	1										
4	0.438965	8										

[5 rows x 261 columns]

In [5]:

```
result=byte_features
```

In [6]:

```
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
result = normalize(result)
```

In [7]:

```
from sklearn.externals import joblib
```

In [8]:

```
joblib.dump(result, 'result.pkl')
```

Out[8]:

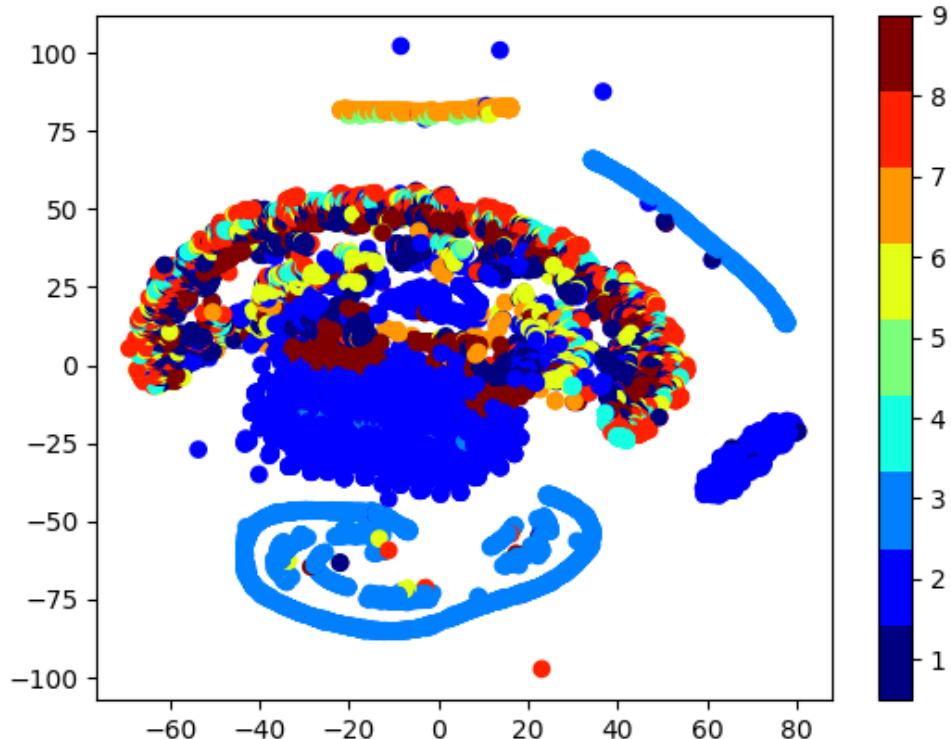
['result.pkl']

### 3.2.4 Multivariate Analysis

In [11]:

```
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()
```

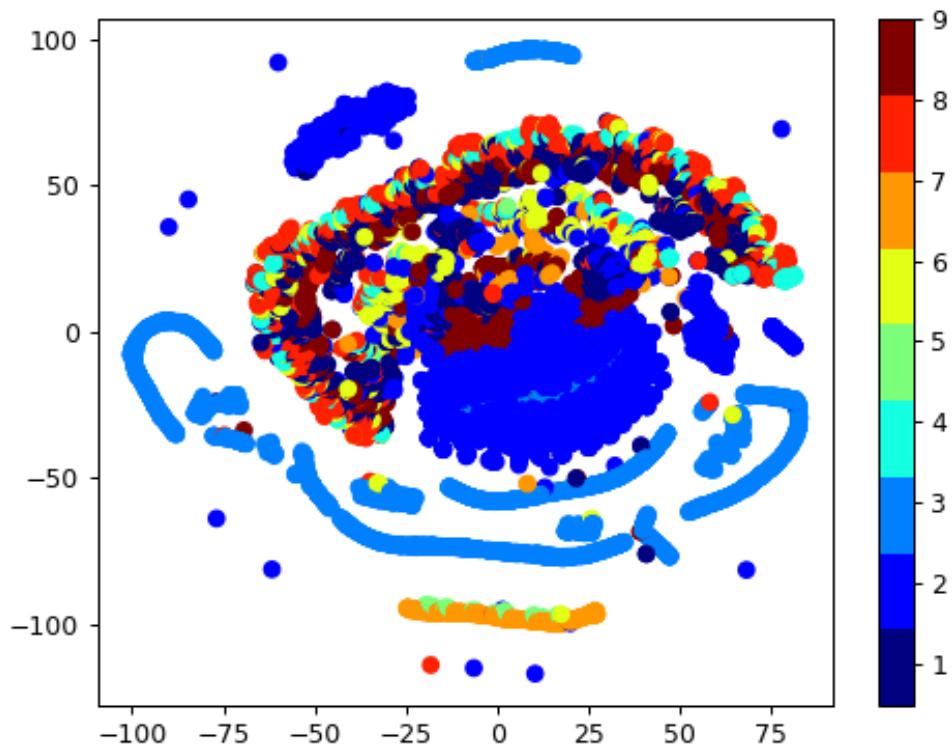
&lt;IPython.core.display.Javascript object&gt;



In [15]:

```
#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 [9]:

```
data_y = result['Class']
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), da
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [10]:

```
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 [11]:

```
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95', '#f0f0f0']
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()

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', train_class_distribution.values[i], '(')

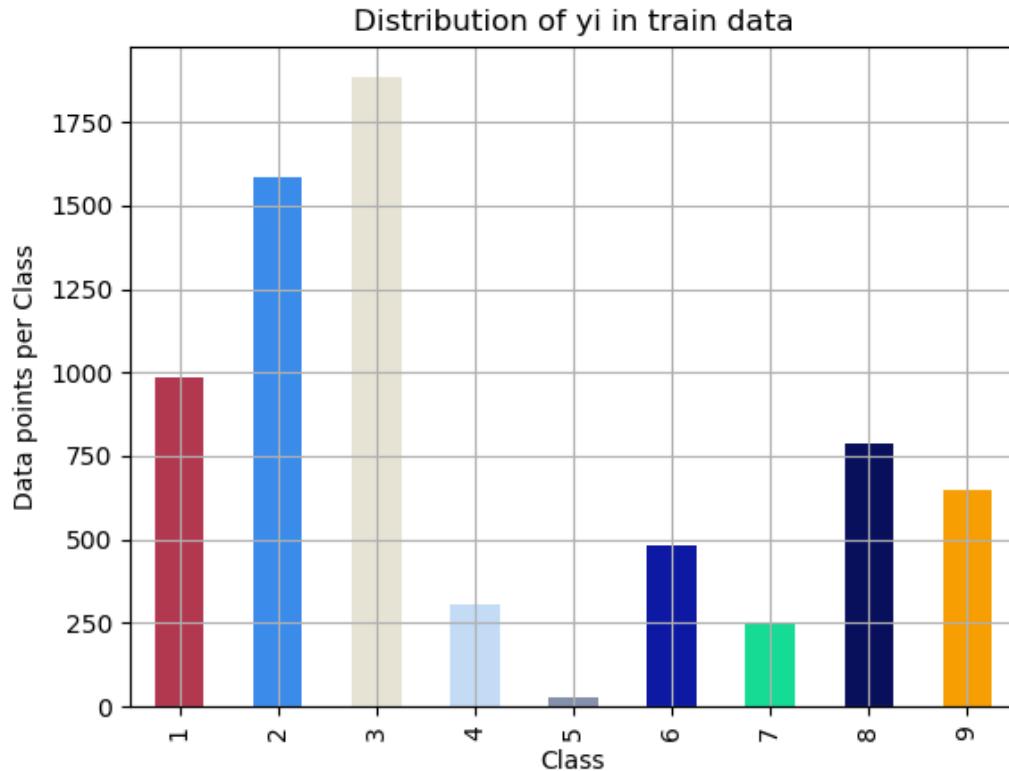
print('*'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95', '#f0f0f0']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', test_class_distribution.values[i], '(')

print('*'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95', '#f0f0f0']
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()

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', cv_class_distribution.values[i], ')')
```

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



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

---

---

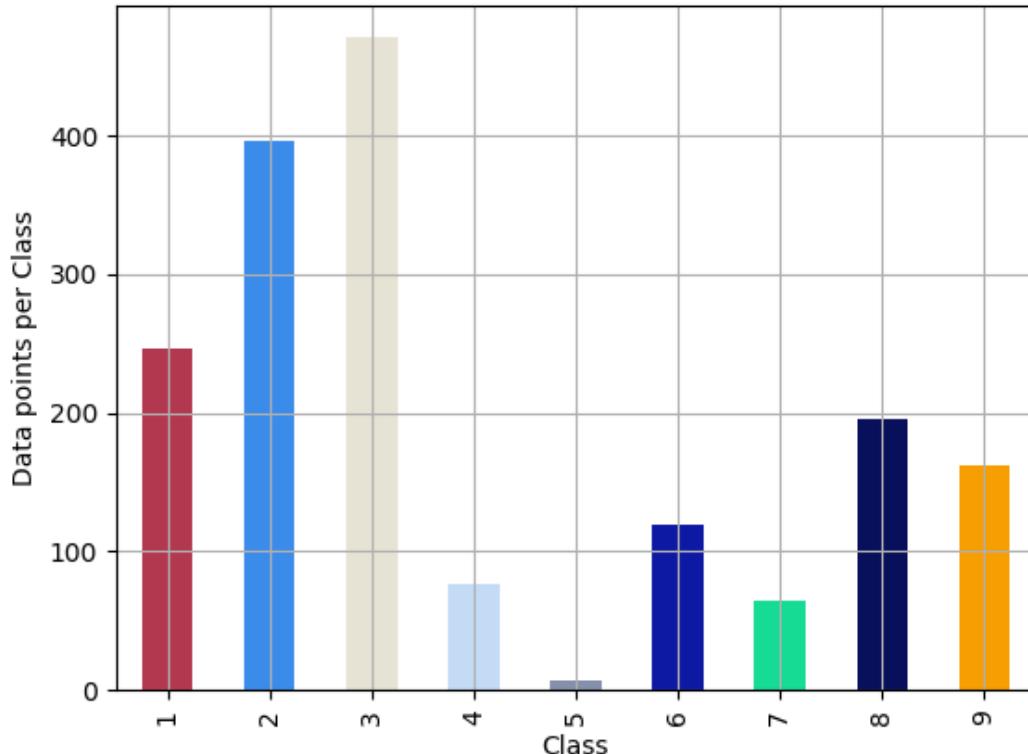
<IPython.core.display.Javascript object>

### Distribution of yi in test data

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

<IPython.core.display.Javascript object>

### Distribution of yi in cross validation data



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 [13]:

```

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

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

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

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

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

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

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

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

```

## 4. Machine Learning Models

### 4.1. Machine Leaning Models on bytes files

#### 4.1.1. Random Model

In [20]:

```

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

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

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

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

```

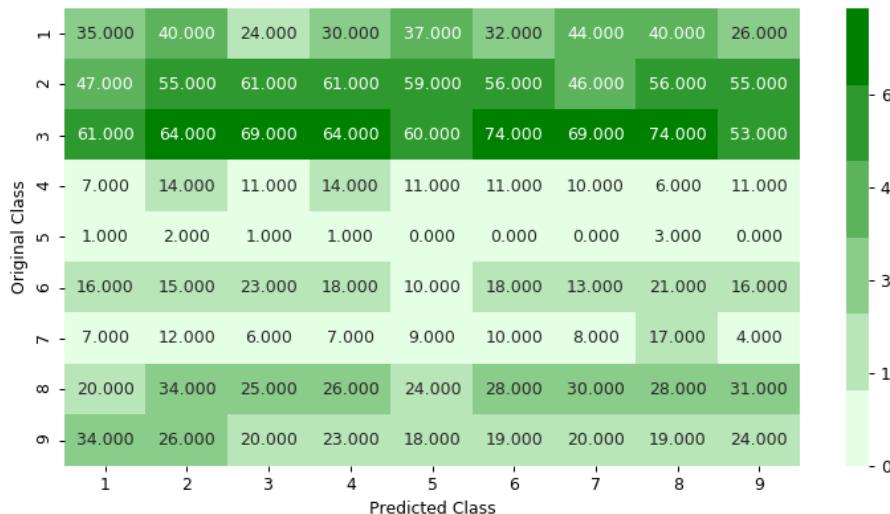
Log loss on Cross Validation Data using Random Model 2.4987116946656167

Log loss on Test Data using Random Model 2.4553327958473936

Number of misclassified points 88.45446182152715

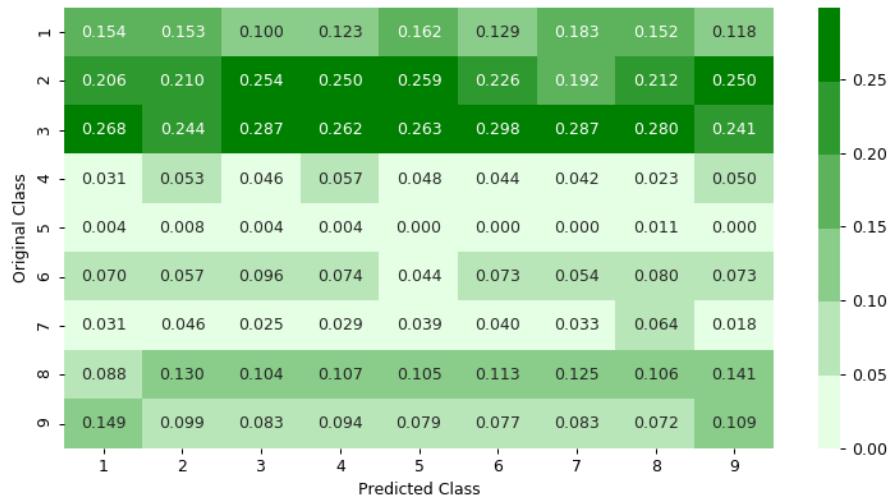
----- Confusion matrix -----

&lt;IPython.core.display.Javascript object&gt;



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

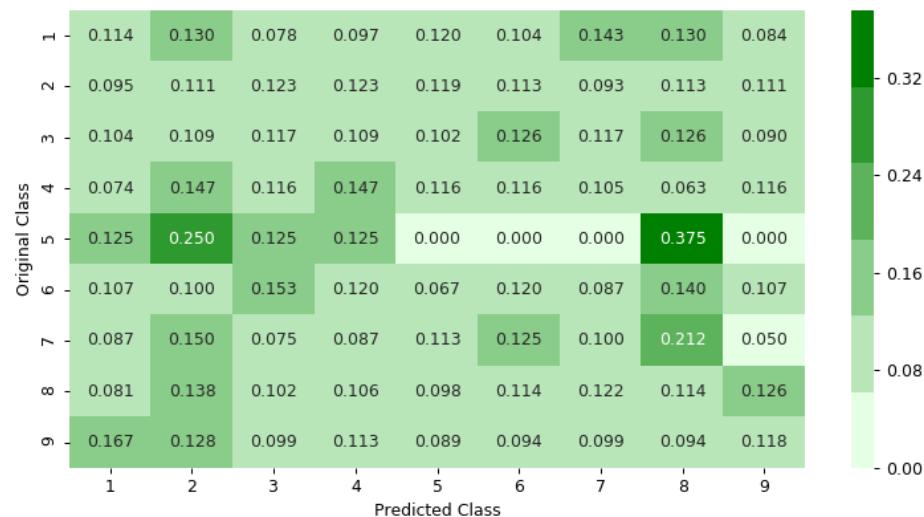
&lt;IPython.core.display.Javascript object&gt;



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 [21]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

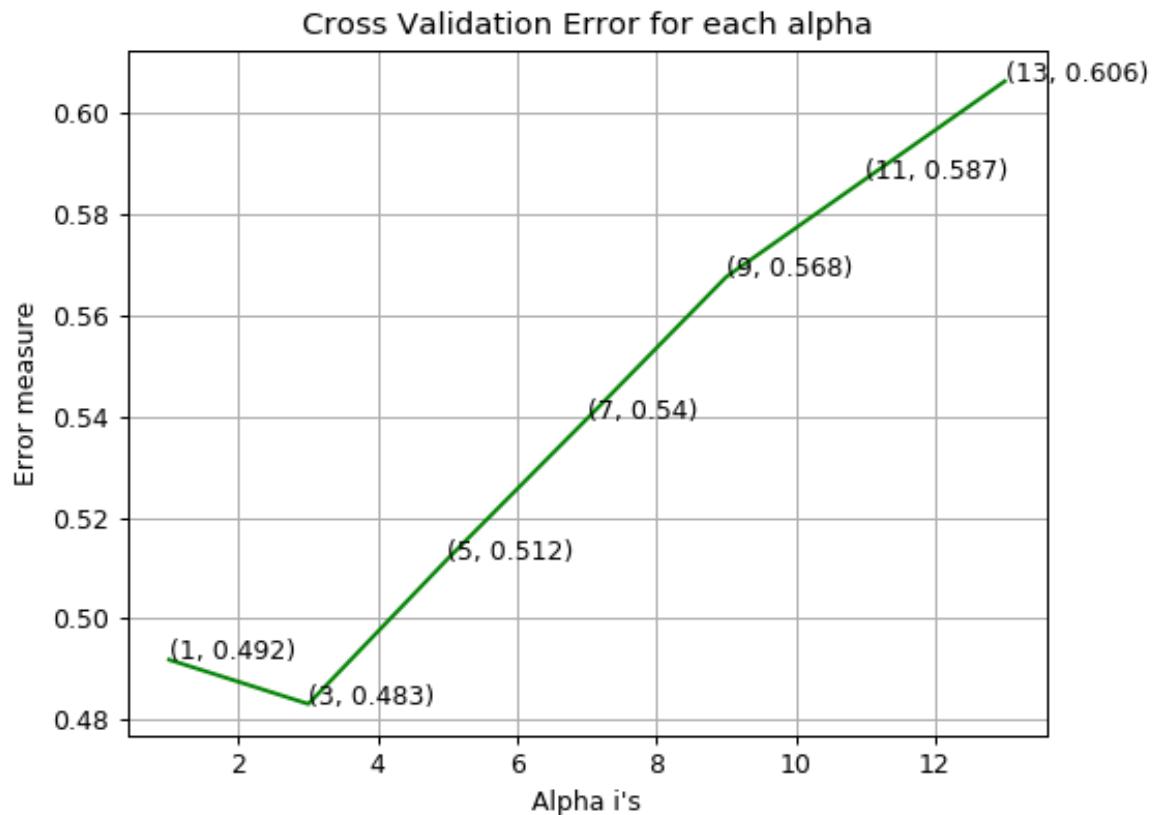
k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```

log\_loss for k = 1 is 0.49188045368463196  
log\_loss for k = 3 is 0.483116902642161  
log\_loss for k = 5 is 0.5118350087441232  
log\_loss for k = 7 is 0.5395490778512431  
log\_loss for k = 9 is 0.5676371813660702  
log\_loss for k = 11 is 0.5870170308367498  
log\_loss for k = 13 is 0.606375118318671

&lt;IPython.core.display.Javascript object&gt;



For values of best alpha = 3 The train log loss is: 0.29320594139515405

For values of best alpha = 3 The cross validation log loss is: 0.483116902642161

For values of best alpha = 3 The test log loss is: 0.4851954874286108

Number of misclassified points 12.97148114075437

----- Confusion matrix -----

<IPython.core.display.Javascript object>

Original Class	Predicted Class								
	1	2	3	4	5	6	7	8	9
1	271.000	0.000	1.000	4.000	0.000	8.000	1.000	15.000	8.000
2	21.000	417.000	3.000	3.000	0.000	16.000	1.000	3.000	32.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	3.000	1.000	0.000	86.000	0.000	3.000	0.000	2.000	0.000
5	1.000	0.000	0.000	3.000	0.000	0.000	4.000	0.000	0.000
6	30.000	1.000	3.000	11.000	0.000	98.000	0.000	2.000	5.000
7	4.000	1.000	1.000	0.000	0.000	5.000	66.000	1.000	2.000
8	14.000	0.000	0.000	5.000	1.000	5.000	1.000	211.000	9.000
9	26.000	4.000	1.000	1.000	0.000	5.000	0.000	11.000	155.000

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

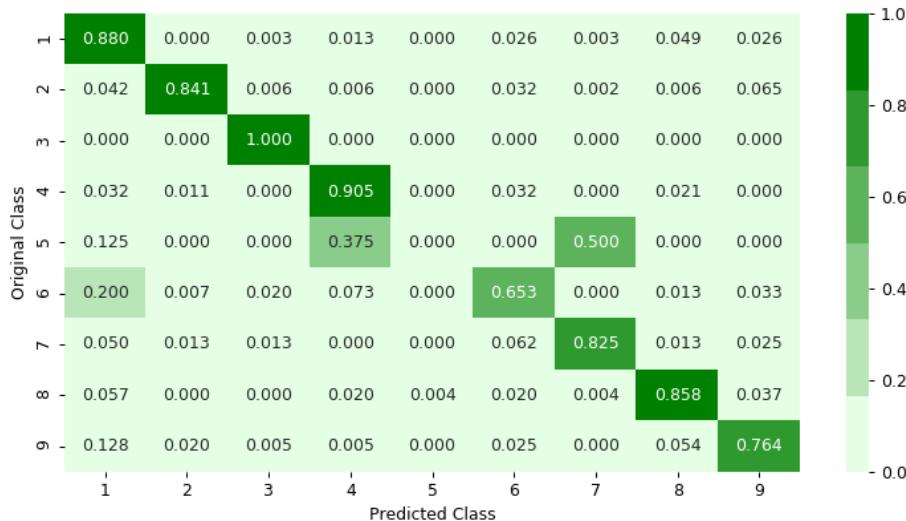
&lt;IPython.core.display.Javascript object&gt;



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

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

&lt;IPython.core.display.Javascript object&gt;



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

#### 4.1.3. Logistic Regression

In [22]:

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

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

best_alpha = np.argmin(cv_log_error_array)

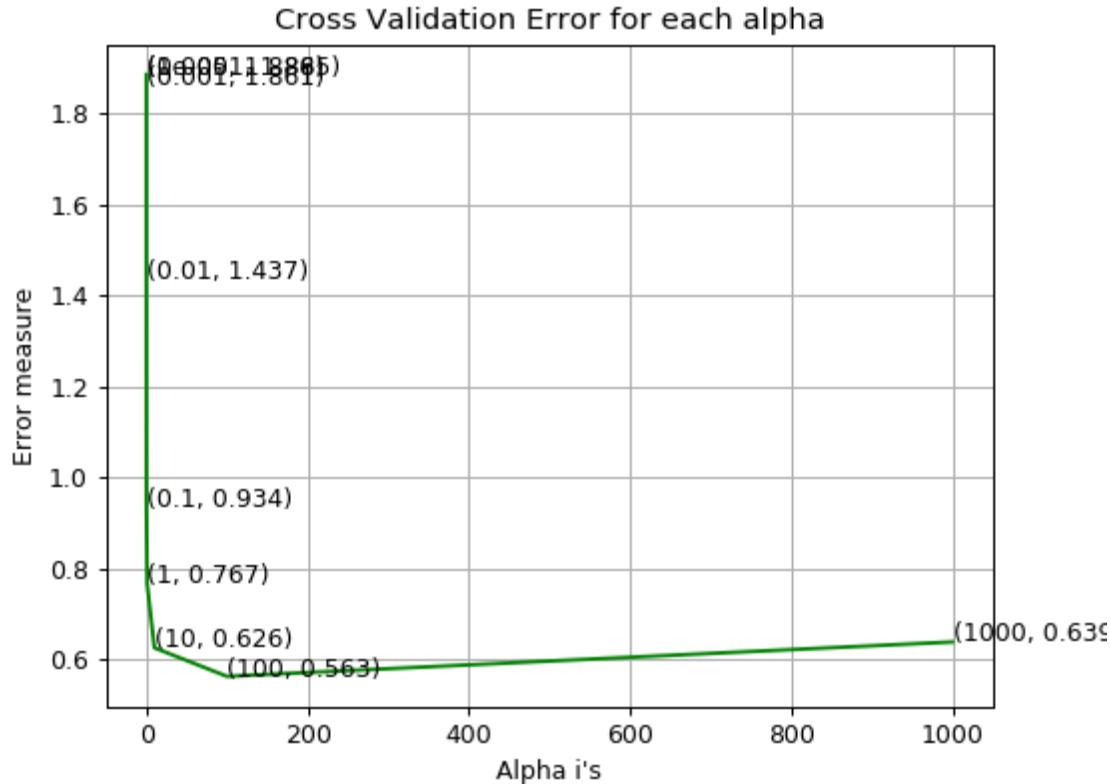
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

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

```
log_loss for c =  1e-05 is 1.8861109311791922
log_loss for c =  0.0001 is 1.8845880471197165
log_loss for c =  0.001 is 1.8614285515198798
log_loss for c =  0.01 is 1.437434379095915
log_loss for c =  0.1 is 0.9337959204695321
log_loss for c =  1 is 0.7667190017910965
log_loss for c =  10 is 0.6257185173536978
log_loss for c =  100 is 0.56294262675526
log_loss for c =  1000 is 0.6385231825855628
```

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



log loss for train data 0.4871437301878761

log loss for cv data 0.56294262675526

log loss for test data 0.5294168099375685

Number of misclassified points 12.603495860165593

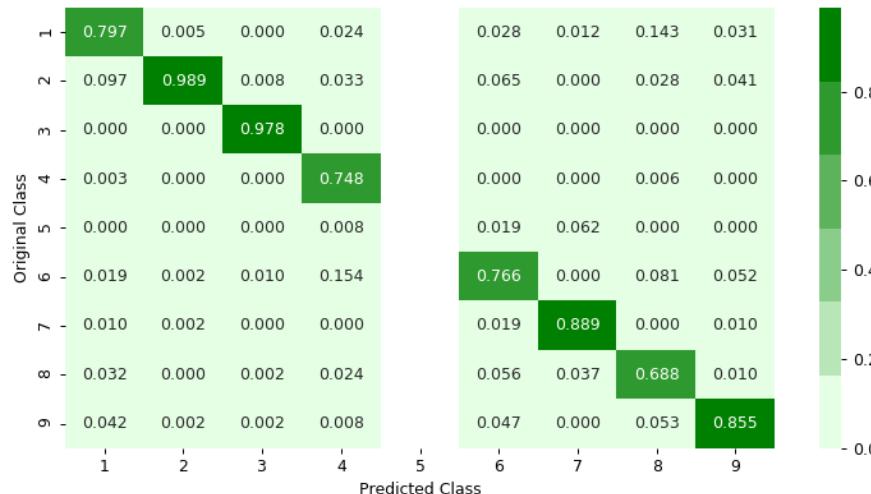
----- Confusion matrix -----

<IPython.core.display.Javascript object>

Original Class	Predicted Class									
	1	2	3	4	5	6	7	8	9	
1	247.000	2.000	0.000	3.000	0.000	3.000	1.000	46.000	6.000	
2	30.000	433.000	5.000	4.000	0.000	7.000	0.000	9.000	8.000	
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000	
4	1.000	0.000	0.000	92.000	0.000	0.000	0.000	2.000	0.000	
5	0.000	0.000	0.000	1.000	0.000	2.000	5.000	0.000	0.000	
6	6.000	1.000	6.000	19.000	0.000	82.000	0.000	26.000	10.000	
7	3.000	1.000	0.000	0.000	0.000	2.000	72.000	0.000	2.000	
8	10.000	0.000	1.000	3.000	0.000	6.000	3.000	221.000	2.000	
9	13.000	1.000	1.000	1.000	0.000	5.000	0.000	17.000	165.000	

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

<IPython.core.display.Javascript object>



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

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

<IPython.core.display.Javascript object>



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

#### 4.1.4. Random Forest Classifier

In [23]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

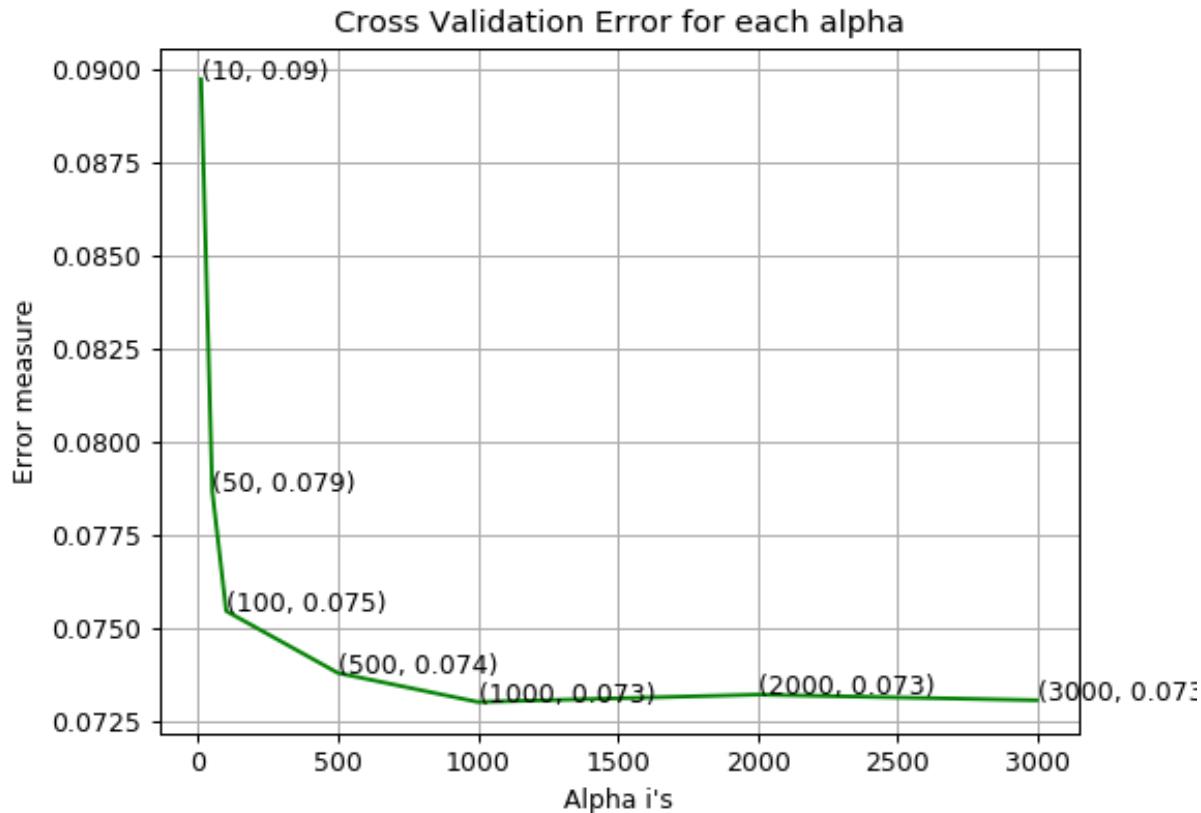
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```

log\_loss for c = 10 is 0.08975272796356877  
log\_loss for c = 50 is 0.07869374681057625  
log\_loss for c = 100 is 0.07546292664044586  
log\_loss for c = 500 is 0.07379883728342362  
log\_loss for c = 1000 is 0.07302077078516724  
log\_loss for c = 2000 is 0.07321813574020479  
log\_loss for c = 3000 is 0.073068132864414

<IPython.core.display.Javascript object>



For values of best alpha = 1000 The train log loss is: 0.02941015229514485

For values of best alpha = 1000 The cross validation log loss is: 0.07302077078516724

For values of best alpha = 1000 The test log loss is: 0.06623869322452512

Number of misclassified points 1.2419503219871204

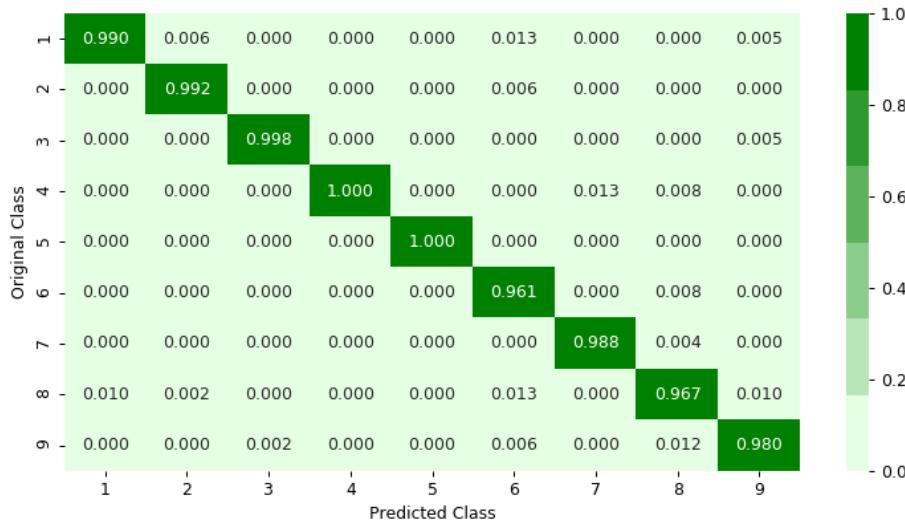
----- Confusion matrix -----

<IPython.core.display.Javascript object>

Original Class	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9
1	302.000	3.000	0.000	0.000	0.000	2.000	0.000	0.000	1.000
2	0.000	495.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
3	0.000	0.000	587.000	0.000	0.000	0.000	0.000	0.000	1.000
4	0.000	0.000	0.000	92.000	0.000	0.000	1.000	2.000	0.000
5	0.000	0.000	0.000	0.000	8.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	148.000	0.000	2.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	79.000	1.000	0.000
8	3.000	1.000	0.000	0.000	0.000	2.000	0.000	238.000	2.000
9	0.000	0.000	1.000	0.000	0.000	1.000	0.000	3.000	198.000

----- 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 [14]:

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

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

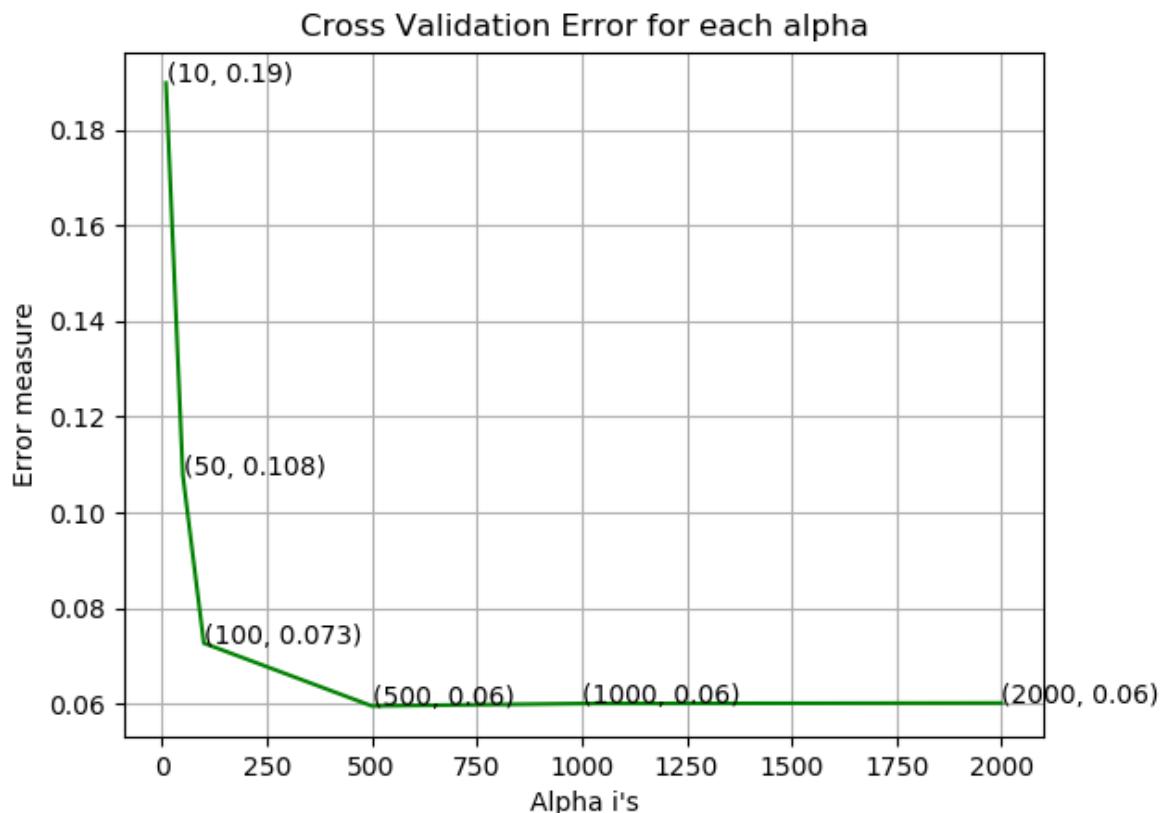
best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c = 10 is 0.18972194520294353
log_loss for c = 50 is 0.10788109266220497
log_loss for c = 100 is 0.0727450829972164
log_loss for c = 500 is 0.059635927131908094
log_loss for c = 1000 is 0.06014538270053144
log_loss for c = 2000 is 0.060249389062255305
<IPython.core.display.Javascript object>
```



For values of best alpha = 500 The train log loss is: 0.02468009568654092

For values of best alpha = 500 The cross validation log loss is: 0.059635927131908094

For values of best alpha = 500 The test log loss is: 0.07847700799402009

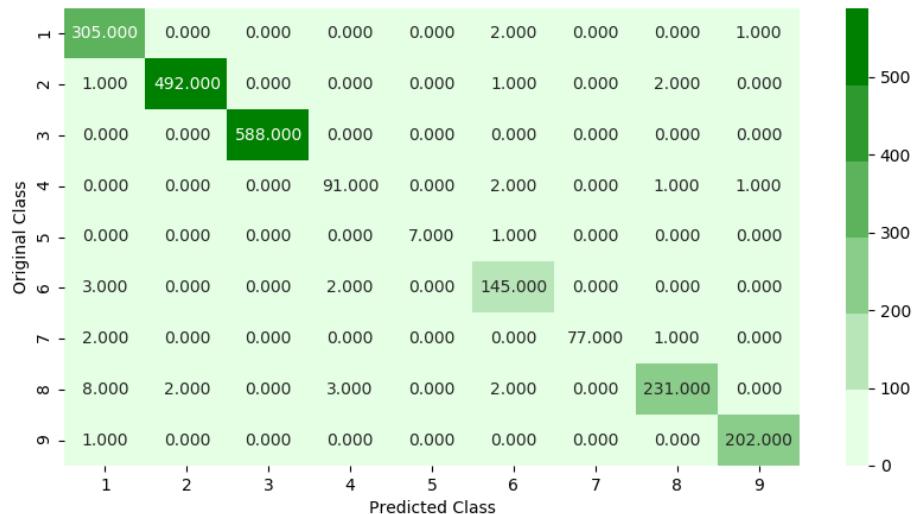
In [15]:

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

Number of misclassified points 1.6559337626494939

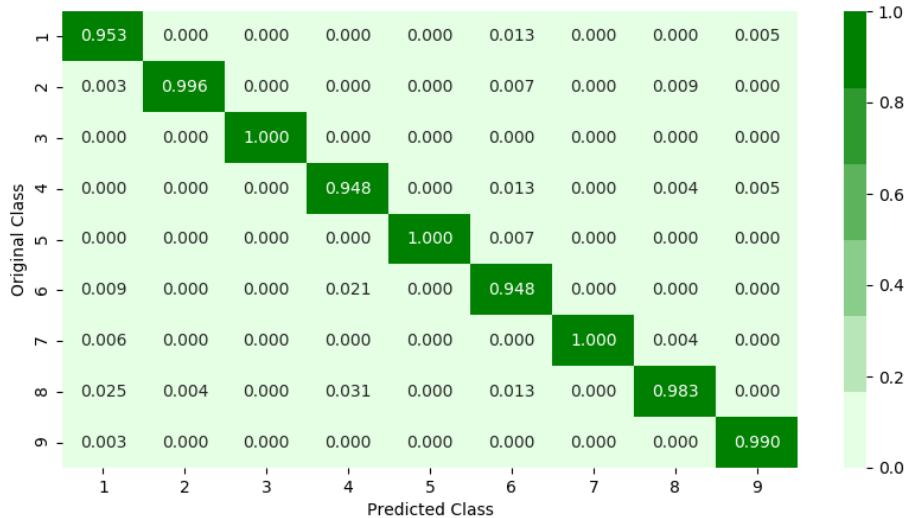
----- Confusion matrix -----

<IPython.core.display.Javascript object>



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

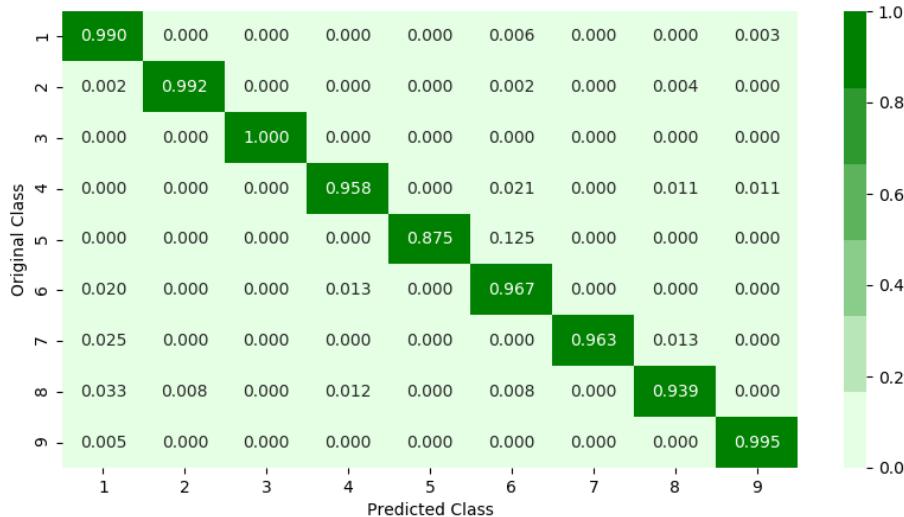
<IPython.core.display.Javascript object>



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 [22]:

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

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 31.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 48.9min remaining:  5.
4min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 54.2min finished
```

Out[22]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                    estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                                            colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
                                            max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
                                            n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
                                            reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                                            silent=True, subsample=1),
                    fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
                    param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]}, pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score='warn', scoring=None, verbose=10)
```

In [ ]:

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

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 5, 'learning_rate': 0.01, 'colsample_bytree': 0.5}
```

In [16]:

```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.01, colsample_bytree=0.5, max_depth=5)
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 0.024348854759529453
cv loss 0.06210692336009718
test loss 0.07717065282799755
```

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

source='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]], 'third')
    elif i%5 ==3:
        shutil.move(source+files[data[i]], 'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]], 'fifth')
```

In [ ]:

```

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

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

    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'de
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmssmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
        with codecs.open('first/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                # https://www.tutorialspoint.com/python3/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixes in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
            #pushing the values into the file after reading whole file
            for prefix in prefixescount:
                file1.write(str(prefix)+",")

```



```

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

```

```

def fourthprocess():
prefixes = ['HEADER','.text','.Pav','.idata','.data','.bss','.rdata','.edata',''
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'de
keywords = ['.dll','std::',':dword']
registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
file1=open("output\hugeasmfile.txt","w+")
files = os.listdir('fourth')
for f in files:
    prefixescount=np.zeros(len(prefixes),dtype=int)
    opcodescount=np.zeros(len(opcodes),dtype=int)
    keywordcount=np.zeros(len(keywords),dtype=int)
    registerscount=np.zeros(len(registers),dtype=int)
    features=[]
    f2=f.split('.')[0]
    file1.write(f2+",")

```

```

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

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

```

```
for i in range(len(registers)):
    for li in line:
        if registers[i] in li and ('text' in l or 'CODE' in l):
            registerscount[i]+=1
    for i in range(len(keywords)):
        for li in line:
            if keywords[i] in li:
                keywordcount[i]+=1
for prefix in prefixescount:
    file1.write(str(prefix)+",")
for opcode in opcodescount:
    file1.write(str(opcode)+",")
for register in registerscount:
    file1.write(str(register)+",")
for key in keywordcount:
    file1.write(str(key)+",")
file1.write("\n")
file1.close()

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

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

In [14]:

```
# asmoutputfile.csv(output generated from the above two cells) will contain all the extract  
# 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[14]:

	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 [15]:

```
#file sizes of byte files

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

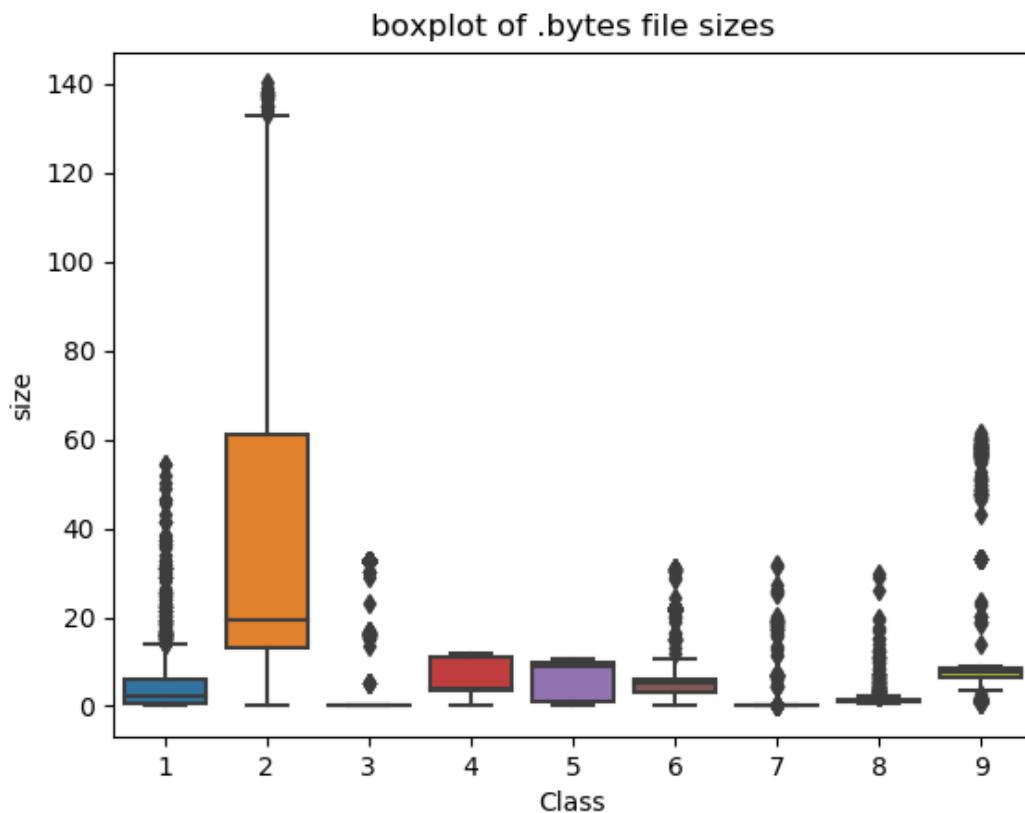
	ID	size	Class
0	01azqd4InC7m9JpocGv5	56.229886	9
1	01IsoiSMh5gxyDYTl4CB	13.999378	2
2	01jsnpXSAlg6aPeDxrU	8.507785	9
3	01kcPWA9K2B0xQeS5Rju	0.078190	1
4	01SuzwMJEIXsK7A8dQb1	0.996723	8

#### 4.2.1.2 Distribution of .asm file sizes

In [15]:

```
#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 [16]:

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

(10868, 53)  
(10868, 3)

Out[16]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2B0xQeS5Rju	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 × 54 columns

In [17]:

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

Out[17]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2B0xQeS5Rju	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

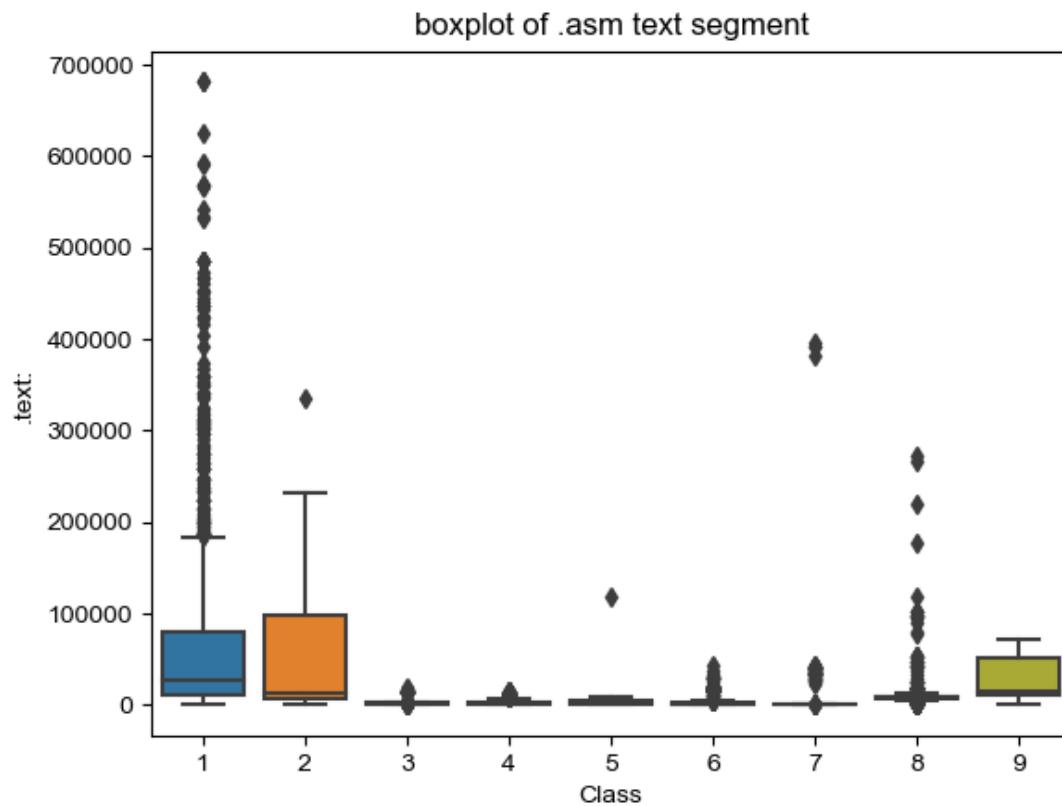
5 rows × 54 columns

## 4.2.2 Univariate analysis on asm file features

In [18]:

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

<IPython.core.display.Javascript object>

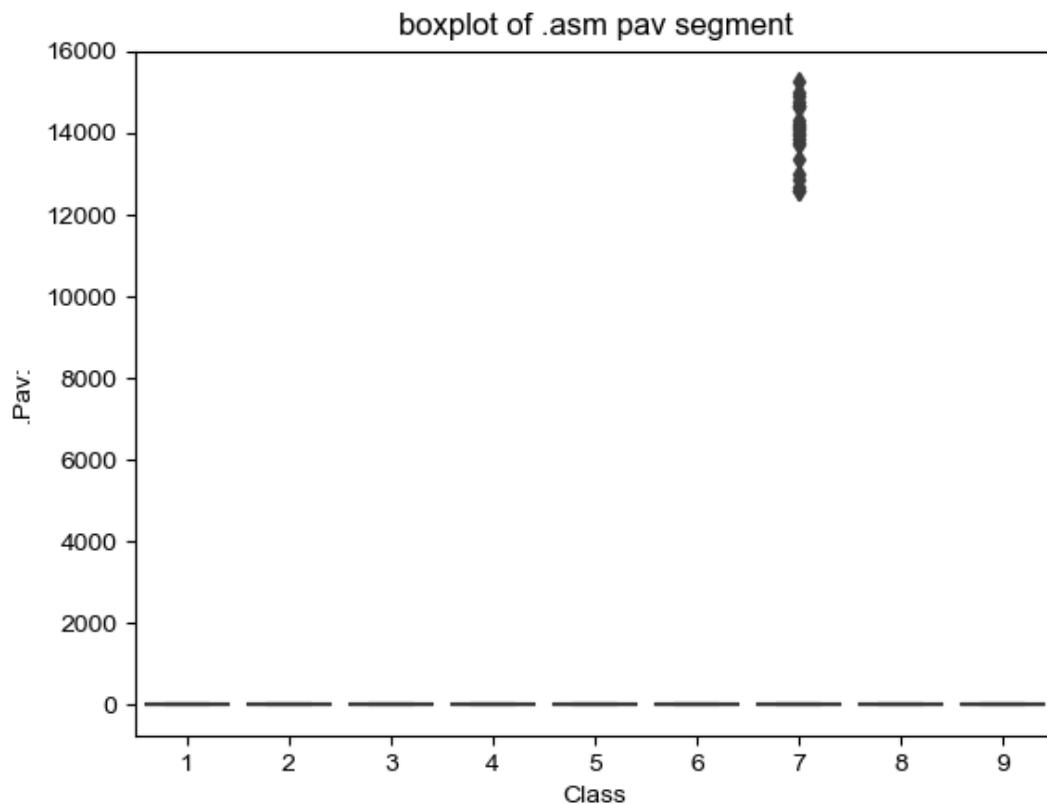


The plot is between Text and class  
Class 1,2 and 9 can be easly separated

In [19]:

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

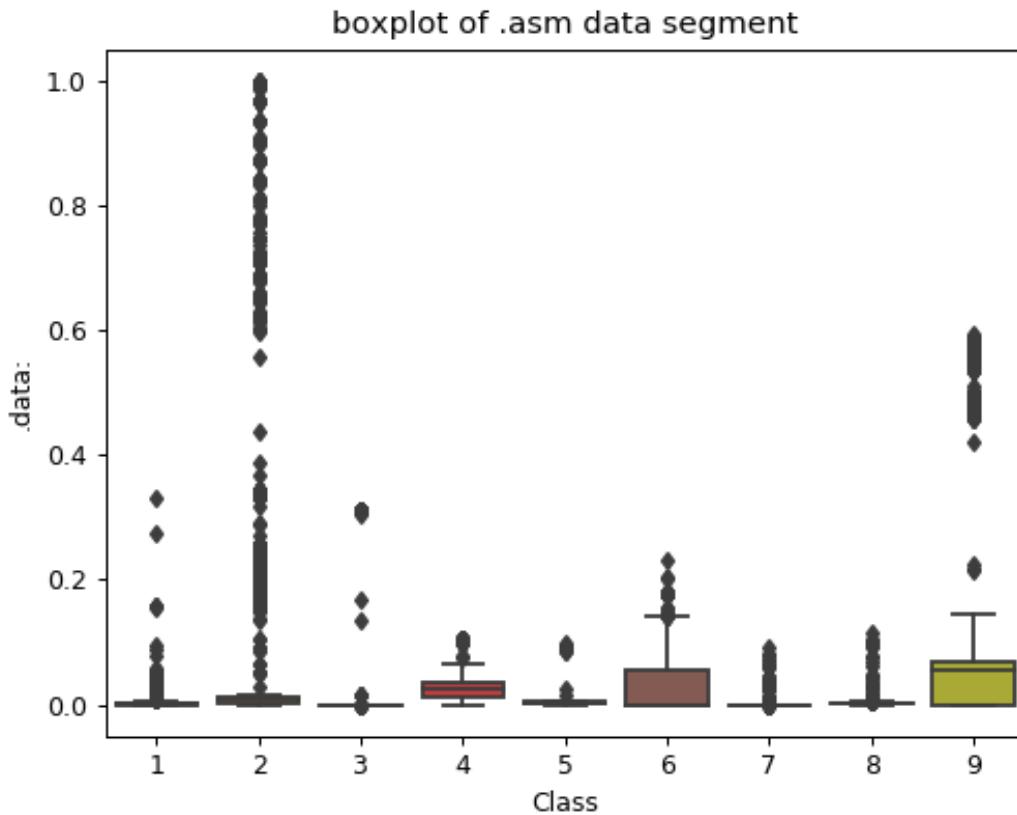
<IPython.core.display.Javascript object>



In [19]:

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

<IPython.core.display.Javascript object>

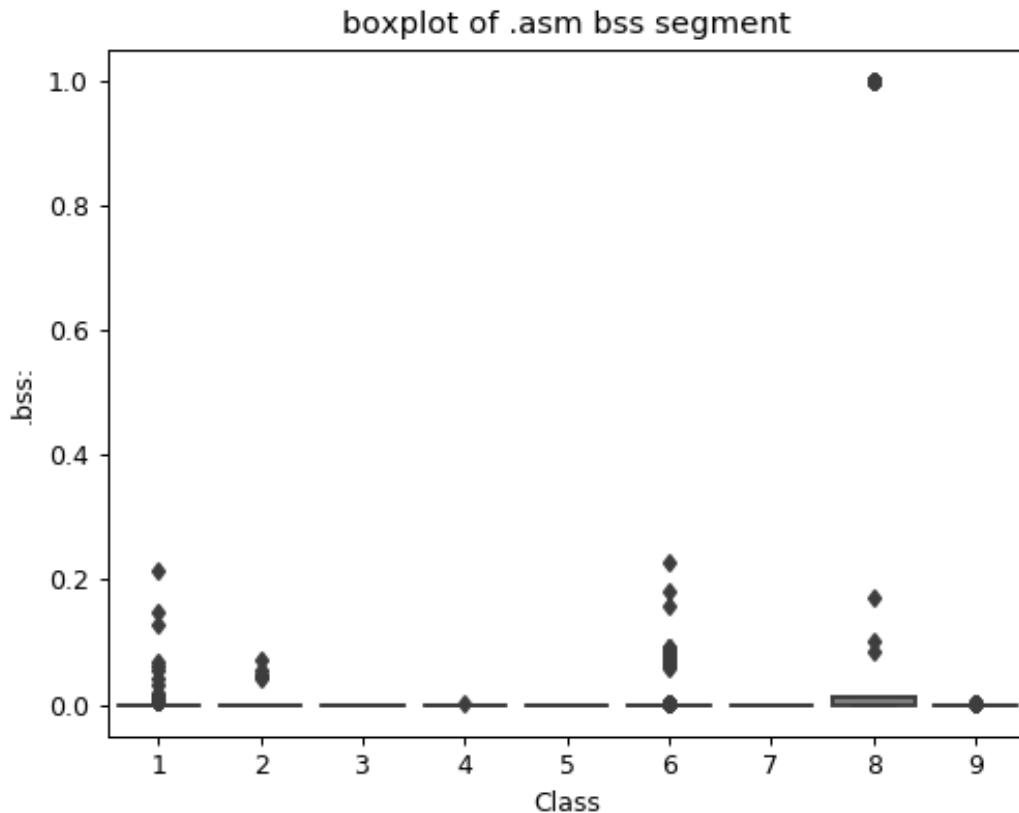


The plot is between data segment and class label  
class 6 and class 9 can be easily separated from given points

In [20]:

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

<IPython.core.display.Javascript object>

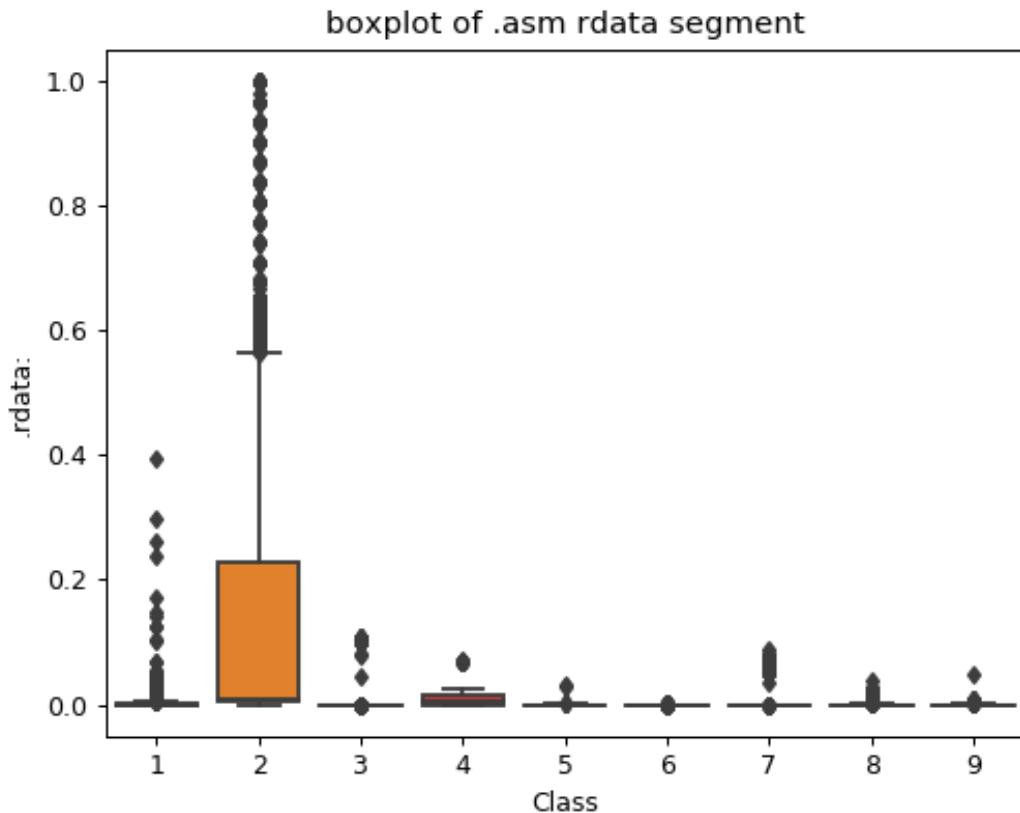


plot between bss segment and class label  
very less number of files are having bss segment

In [21]:

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

<IPython.core.display.Javascript object>



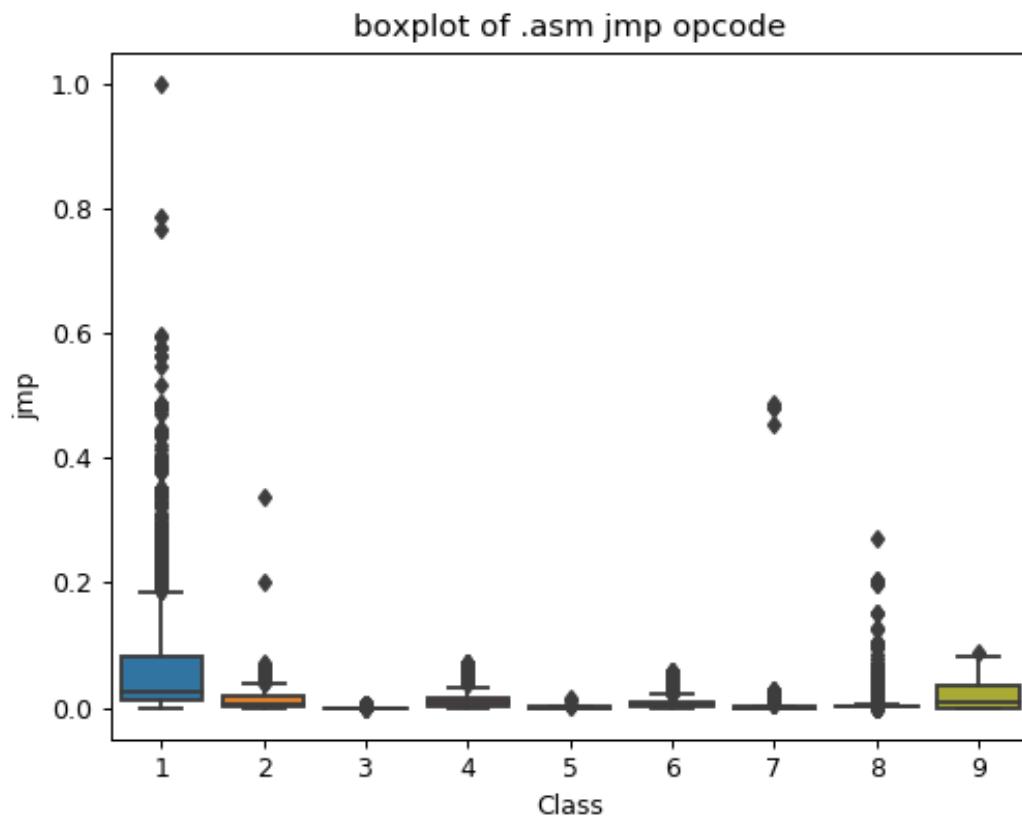
Plot between rdata segment and Class segment

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

In [22]:

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

<IPython.core.display.Javascript object>



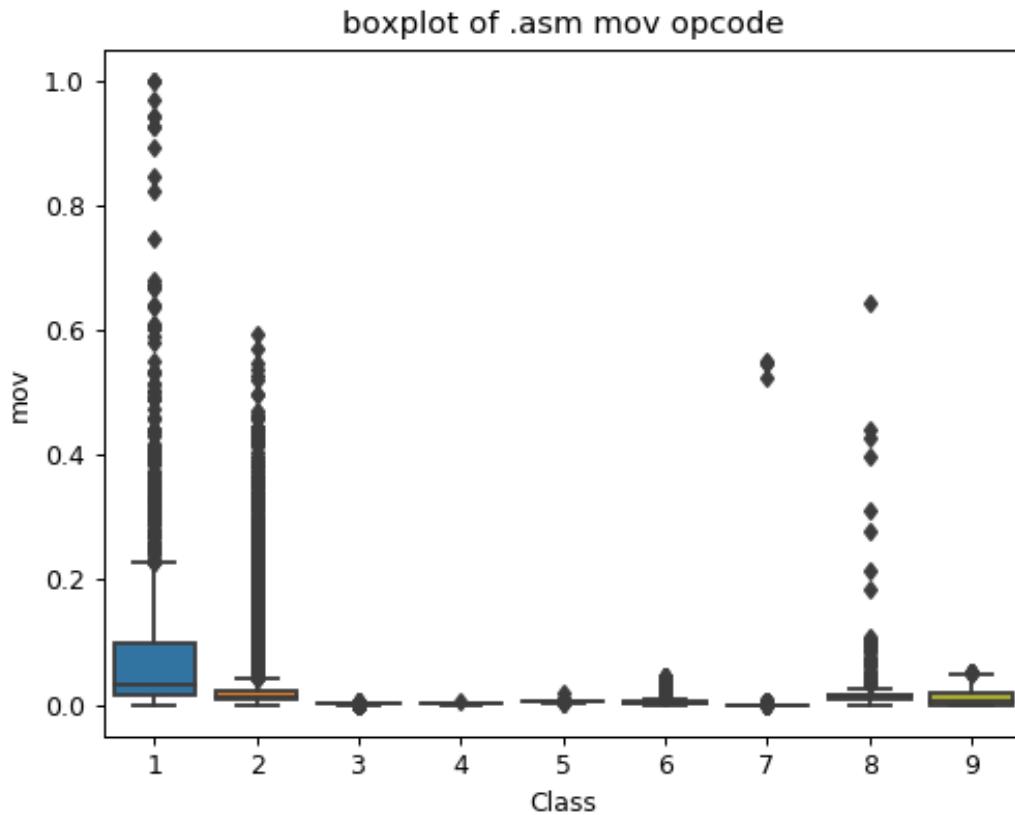
plot between jmp and Class label

Class 1 is having frequency of 2000 approx in 75 percentile of files

In [23]:

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

<IPython.core.display.Javascript object>

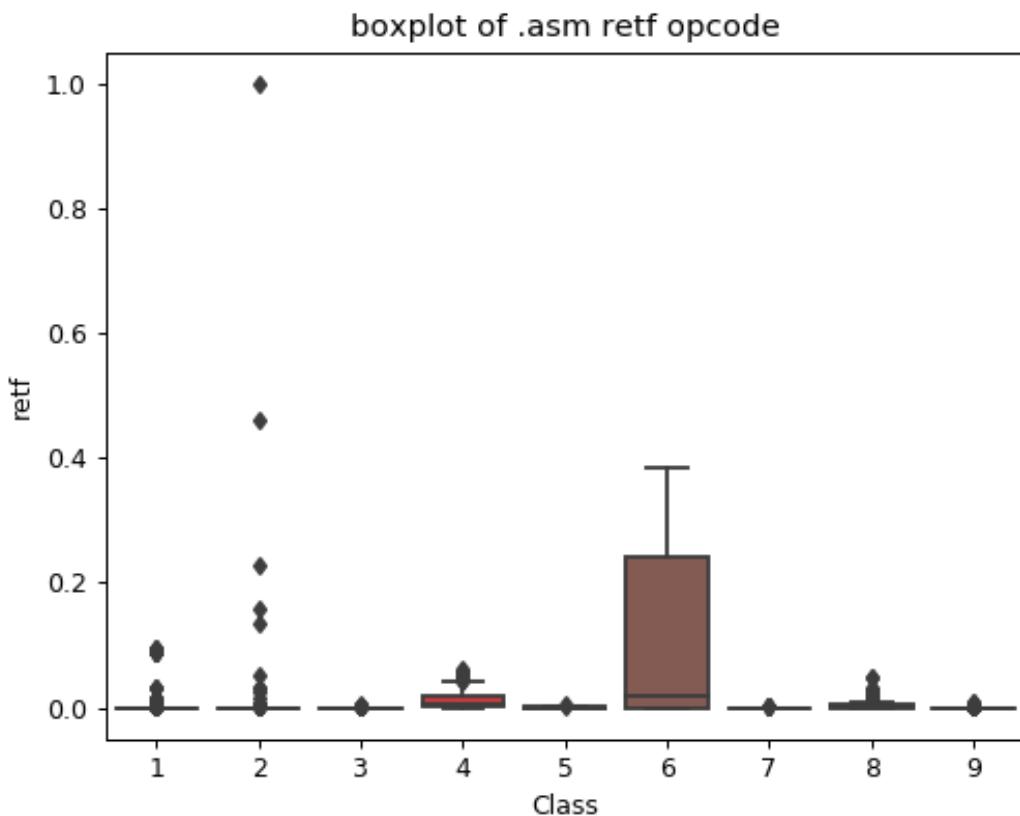


plot between Class label and mov opcode  
Class 1 is having frequency of 2000 approx in 75 percentile of files

In [24]:

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

<IPython.core.display.Javascript object>

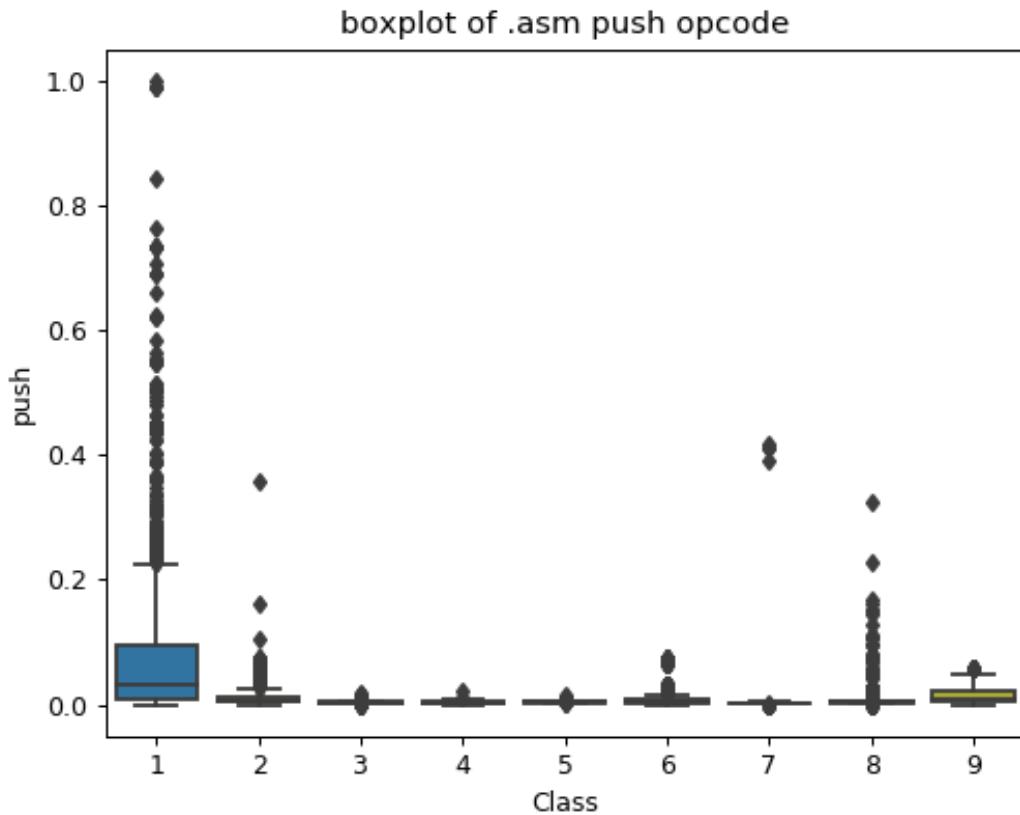


plot between Class label and retf  
Class 6 can be easily separated with opcode retf  
The frequency of retf is approx of 250.

In [25]:

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

<IPython.core.display.Javascript object>



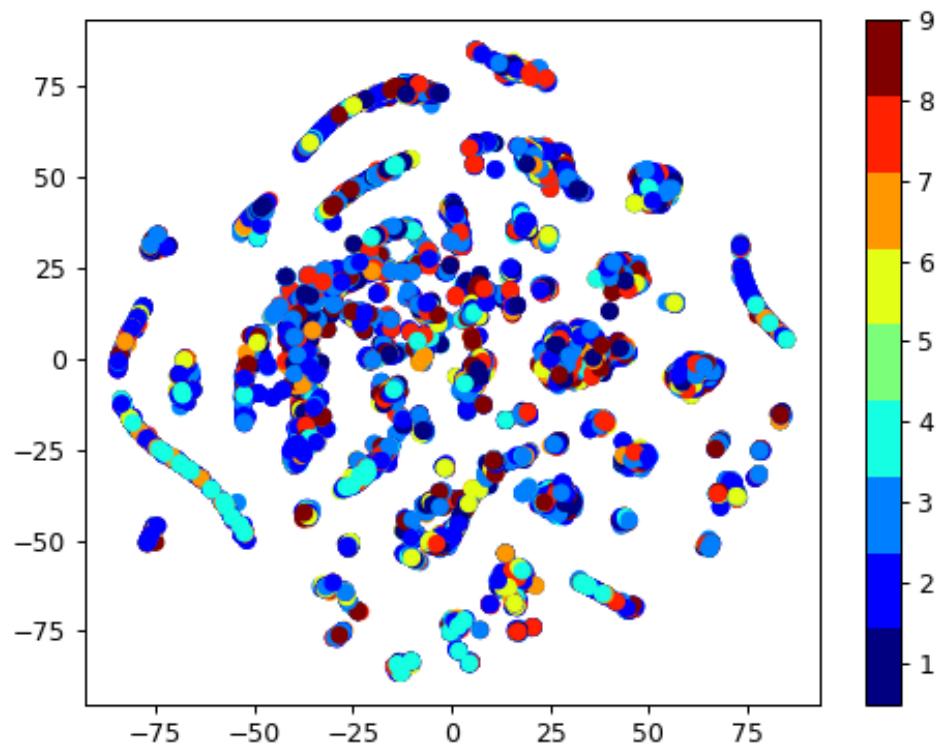
plot between push opcode and Class label  
Class 1 is having 75 precentile files with push opcodes of frequency 1000

#### 4.2.2 Multivariate Analysis on .asm file features

In [16]:

```
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()
```

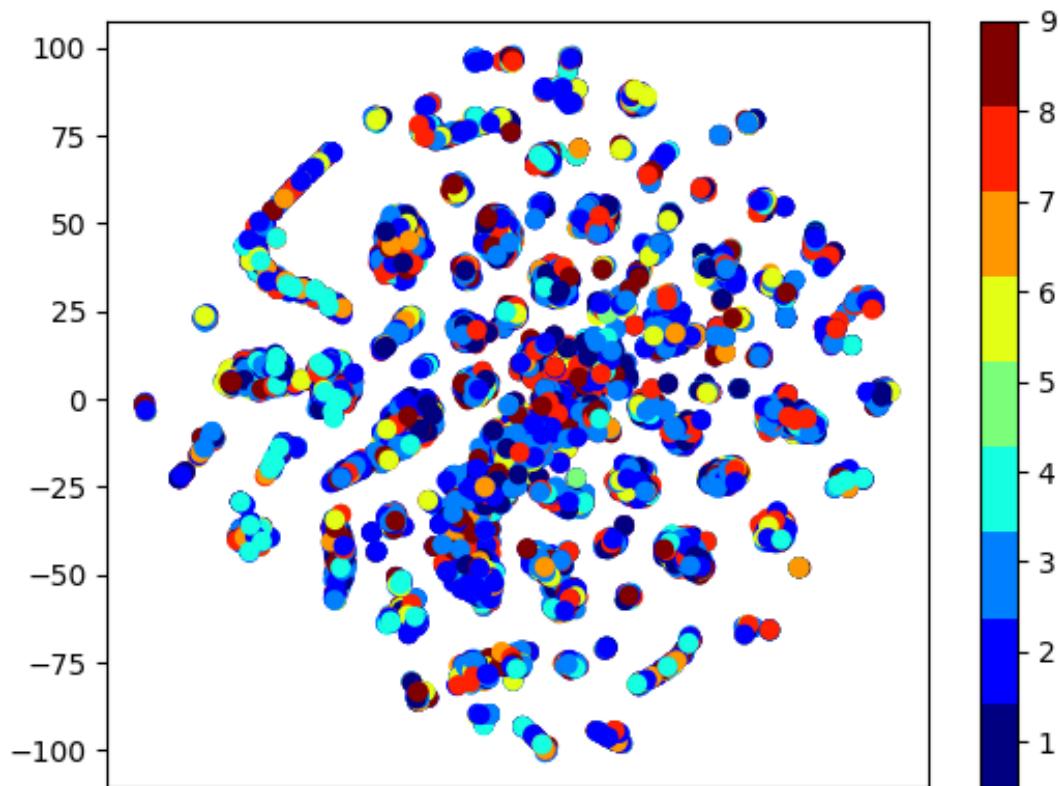
&lt;IPython.core.display.Javascript object&gt;



In [30]:

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

&lt;IPython.core.display.Javascript object&gt;



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 [18]:

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

In [19]:

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

In [20]:

```
print( X_cv_asm.isnull().all())
```

```
HEADER:      False
.text:       False
.Pav:        False
.idata:      False
.data:       False
.bss:        False
.rdata:      False
.edata:      False
.rsrc:       False
.tls:        False
.reloc:      False
jmp          False
mov          False
retf         False
push         False
pop          False
xor          False
retn         False
nop          False
sub          False
inc          False
dec          False
add          False
imul         False
xchg         False
or           False
shr          False
cmp          False
call         False
shl          False
ror          False
rol          False
jnb          False
jz           False
lea           False
movzx        False
.dll         False
std:::       False
:dword        False
edx          False
esi          False
eax          False
ebx          False
ecx          False
edi          False
ebp          False
esp          False
eip          False
size         False
dtype: bool
```

## 4.4. Machine Learning models on features of .asm files

### 4.4.1 K-Nearest Neighbors



In [35]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

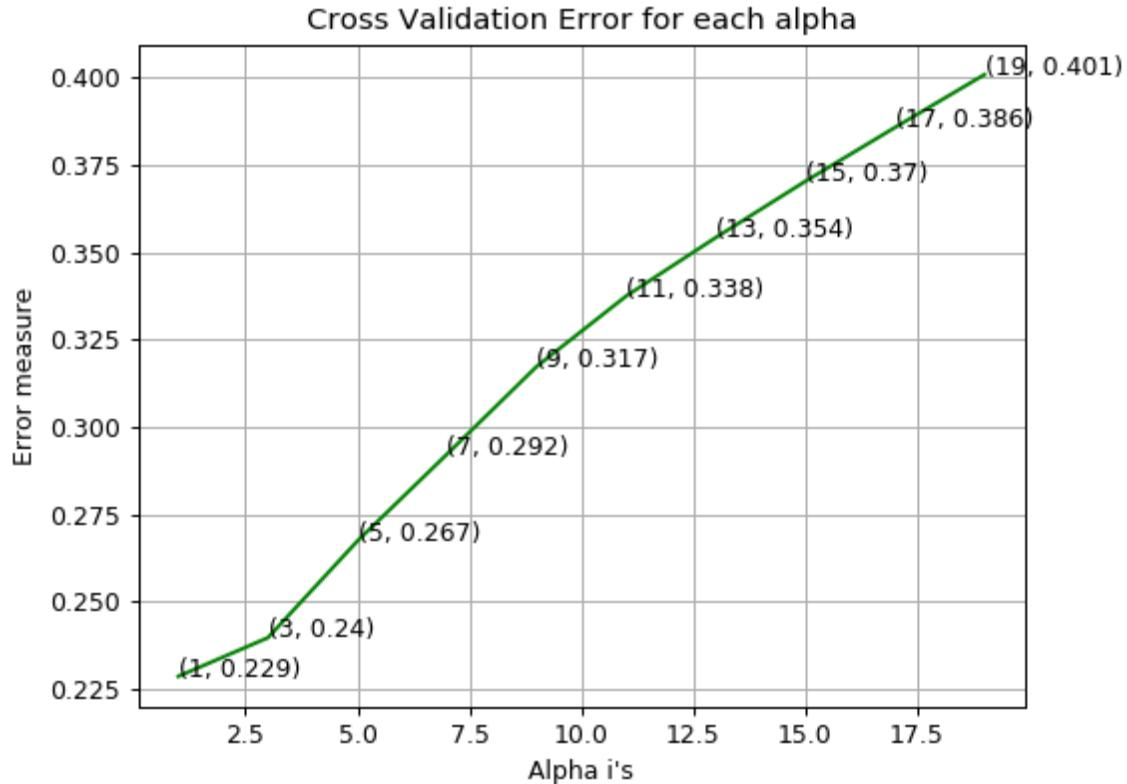
k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)

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

```

log\_loss for k = 1 is 0.2286951008264786  
log\_loss for k = 3 is 0.23974604909767921  
log\_loss for k = 5 is 0.26743767182569295  
log\_loss for k = 7 is 0.2922812711849662  
log\_loss for k = 9 is 0.3173517943176062  
log\_loss for k = 11 is 0.3375272301343973  
log\_loss for k = 13 is 0.3542581717184334  
log\_loss for k = 15 is 0.3703567252351854  
log\_loss for k = 17 is 0.3857570471590401  
log\_loss for k = 19 is 0.40090334916939535

<IPython.core.display.Javascript object>



log loss for train data 0.07371820550676085

log loss for cv data 0.2286951008264786

log loss for test data 0.2135354369723588

Number of misclassified points 4.001839926402944

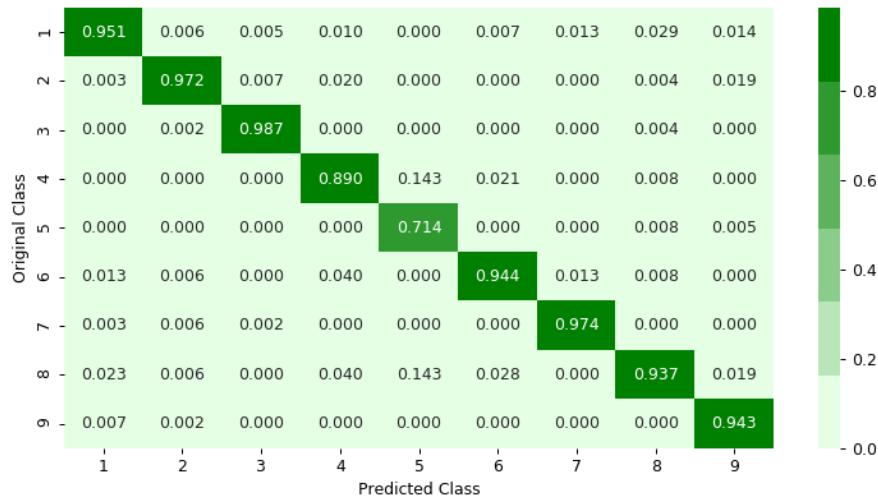
----- Confusion matrix -----

<IPython.core.display.Javascript object>

Original Class	Predicted Class								
	1	2	3	4	5	6	7	8	9
1	289.000	3.000	3.000	1.000	0.000	1.000	1.000	7.000	3.000
2	1.000	484.000	4.000	2.000	0.000	0.000	0.000	1.000	4.000
3	0.000	1.000	586.000	0.000	0.000	0.000	0.000	1.000	0.000
4	0.000	0.000	0.000	89.000	1.000	3.000	0.000	2.000	0.000
5	0.000	0.000	0.000	0.000	5.000	0.000	0.000	2.000	1.000
6	4.000	3.000	0.000	4.000	0.000	136.000	1.000	2.000	0.000
7	1.000	3.000	1.000	0.000	0.000	0.000	75.000	0.000	0.000
8	7.000	3.000	0.000	4.000	1.000	4.000	0.000	223.000	4.000
9	2.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	200.000

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

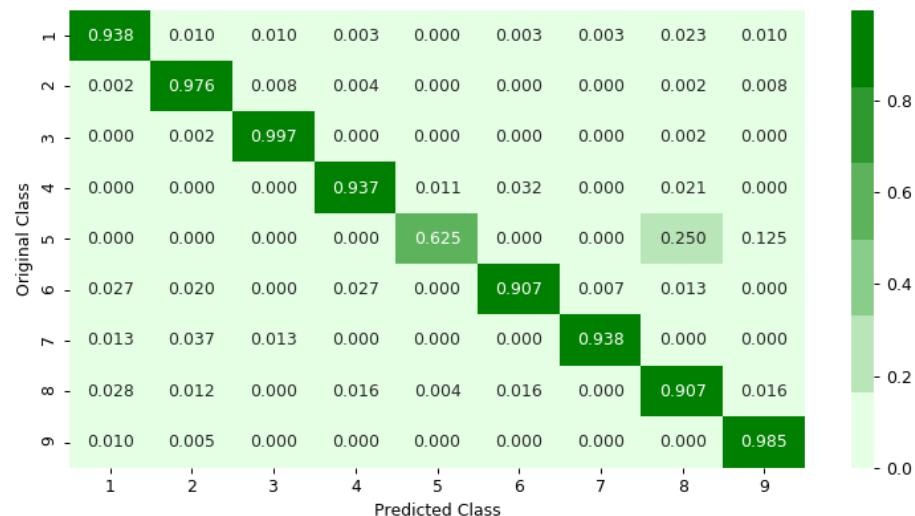
<IPython.core.display.Javascript object>



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.]

#### 4.4.2 Logistic Regression

In [36]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

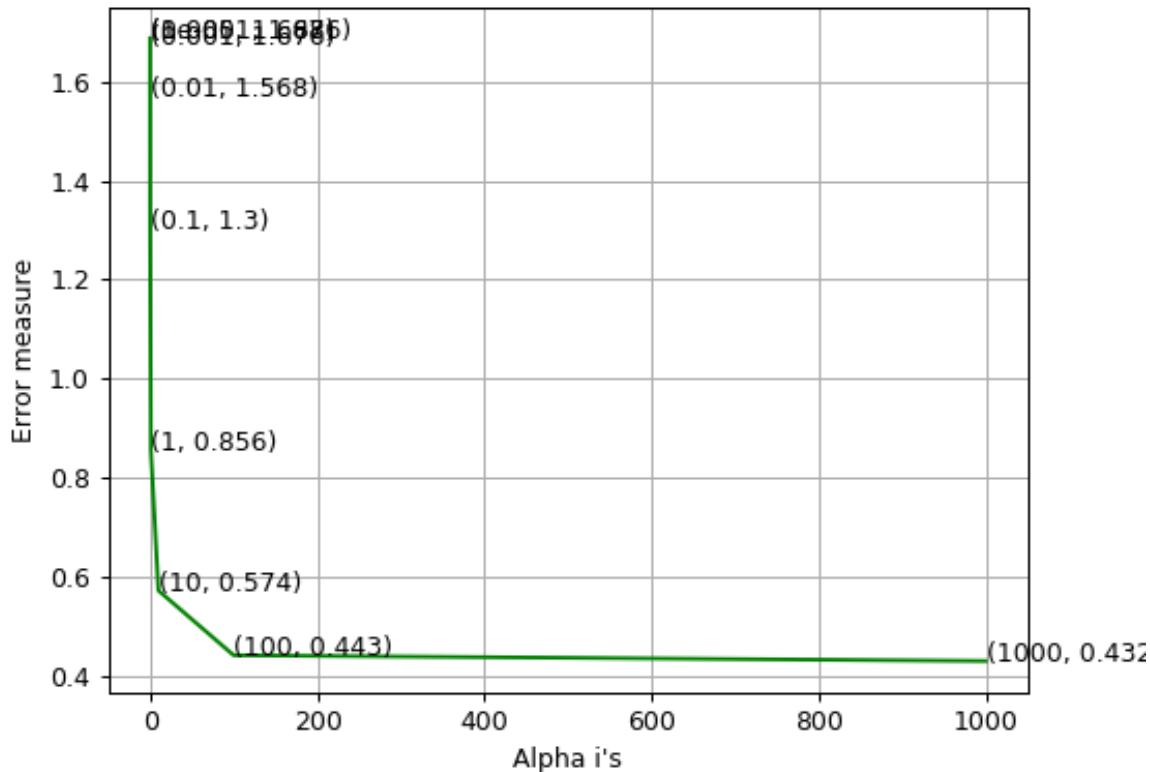
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=logisticR.classes_
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15)
predict_y = sig_clf.predict_proba(X_test_asm)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

```

log\_loss for c = 1e-05 is 1.6869859868957804  
 log\_loss for c = 0.0001 is 1.6855010192472757  
 log\_loss for c = 0.001 is 1.6755781562152487  
 log\_loss for c = 0.01 is 1.5677273322121714  
 log\_loss for c = 0.1 is 1.3002573116338927  
 log\_loss for c = 1 is 0.856048258533692  
 log\_loss for c = 10 is 0.5735687649879864  
 log\_loss for c = 100 is 0.4431214718098947  
 log\_loss for c = 1000 is 0.43157353232283385

&lt;IPython.core.display.Javascript object&gt;

### Cross Validation Error for each alpha



log loss for train data 0.38150548196180933

log loss for cv data 0.43157353232283385

log loss for test data 0.38308926013384326

Number of misclassified points 7.405703771849126

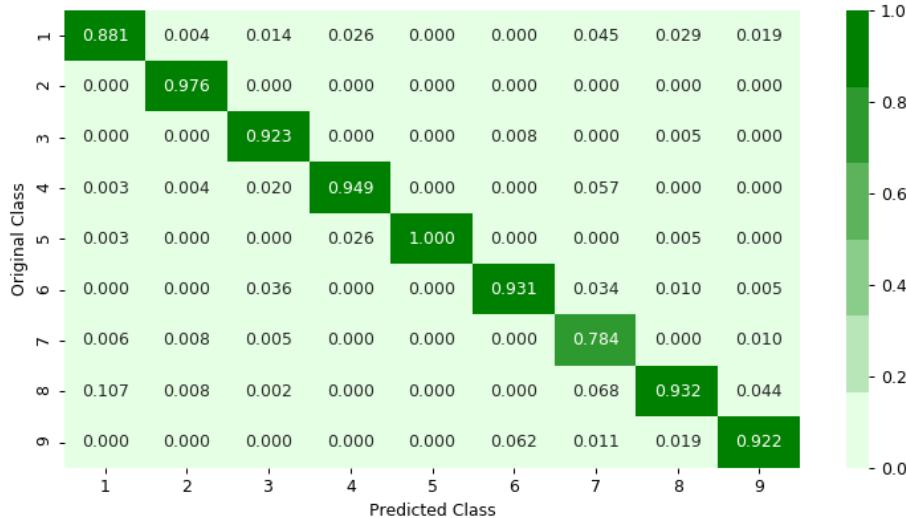
----- Confusion matrix -----

<IPython.core.display.Javascript object>

Original Class	Predicted Class									
	1	2	3	4	5	6	7	8	9	
1	281.000	2.000	9.000	2.000	0.000	0.000	4.000	6.000	4.000	
2	0.000	496.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
3	0.000	0.000	586.000	0.000	0.000	1.000	0.000	1.000	0.000	
4	1.000	2.000	13.000	74.000	0.000	0.000	5.000	0.000	0.000	
5	1.000	0.000	0.000	2.000	4.000	0.000	0.000	1.000	0.000	
6	0.000	0.000	23.000	0.000	0.000	121.000	3.000	2.000	1.000	
7	2.000	4.000	3.000	0.000	0.000	0.000	69.000	0.000	2.000	
8	34.000	4.000	1.000	0.000	0.000	0.000	6.000	192.000	9.000	
9	0.000	0.000	0.000	0.000	0.000	8.000	1.000	4.000	190.000	

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

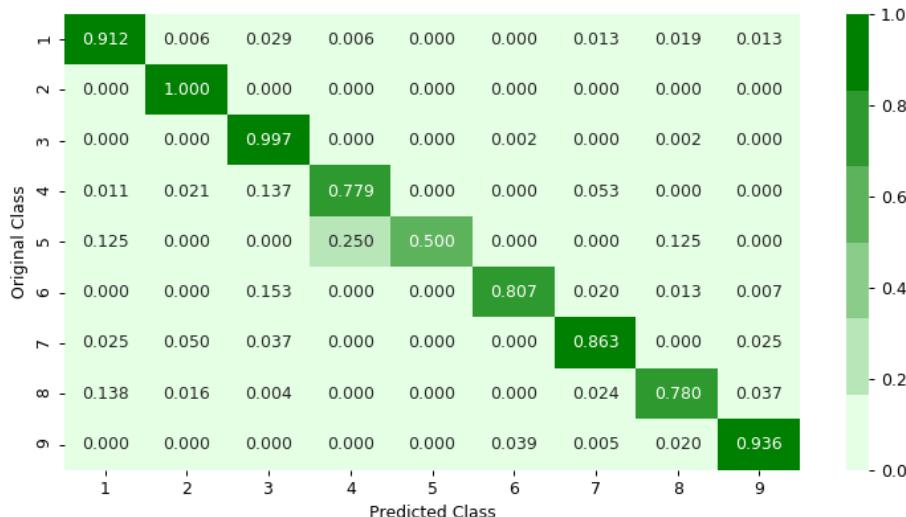
<IPython.core.display.Javascript object>



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.]

#### 4.4.3 Random Forest Classifier

In [37]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

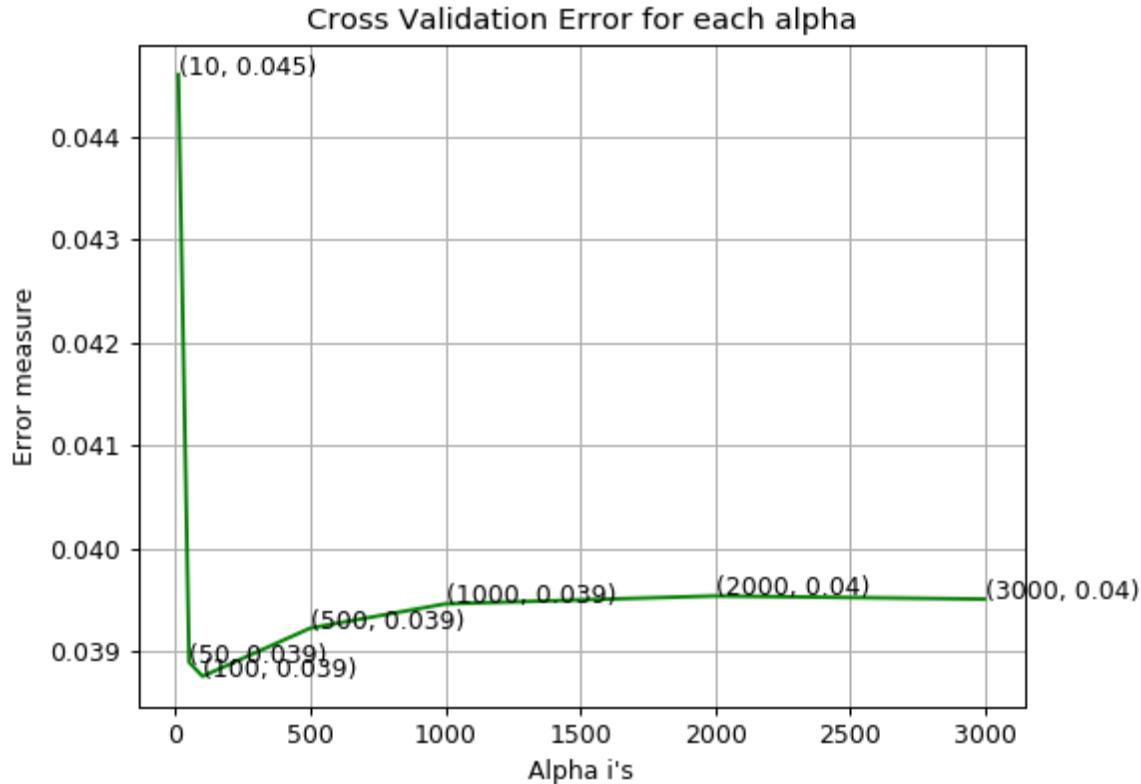
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm,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

```

◀ ▶

log\_loss for c = 10 is 0.044604000720488056  
log\_loss for c = 50 is 0.038892329851189296  
log\_loss for c = 100 is 0.03875524544813011  
log\_loss for c = 500 is 0.039224440805809314  
log\_loss for c = 1000 is 0.03945941790783839  
log\_loss for c = 2000 is 0.03953659123286974  
log\_loss for c = 3000 is 0.03950608587732239

&lt;IPython.core.display.Javascript object&gt;



log loss for train data 0.012379247850927044

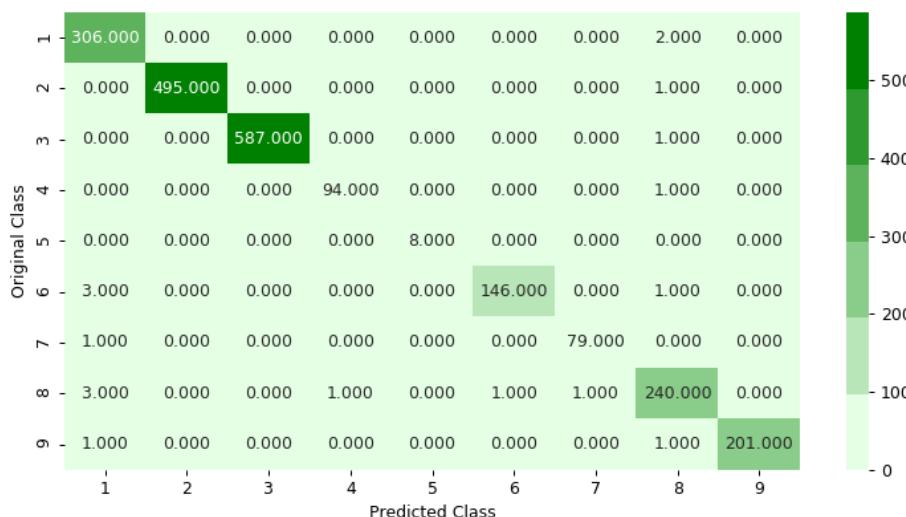
log loss for cv data 0.03875524544813011

log loss for test data 0.039428378936875376

Number of misclassified points 0.8279668813247469

----- Confusion matrix -----

<IPython.core.display.Javascript object>



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

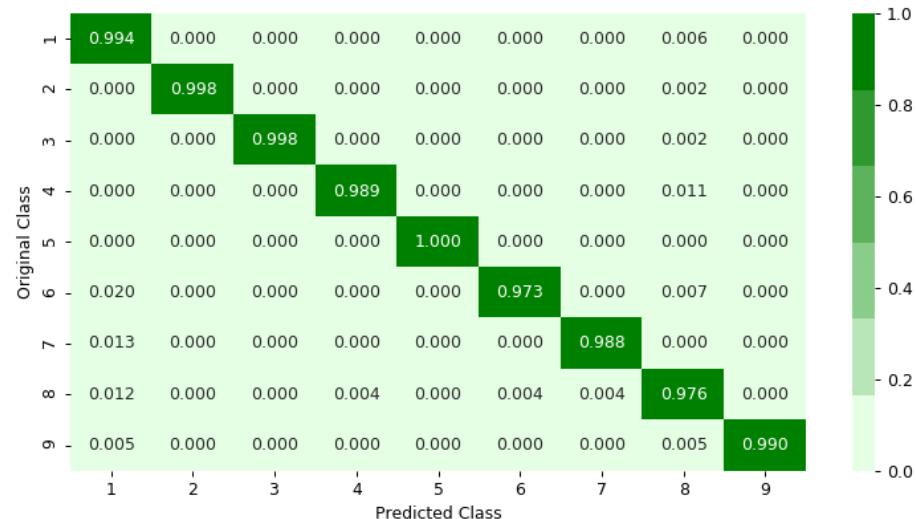
<IPython.core.display.Javascript object>



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.]

#### 4.4.4 XgBoost Classifier

In [38]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

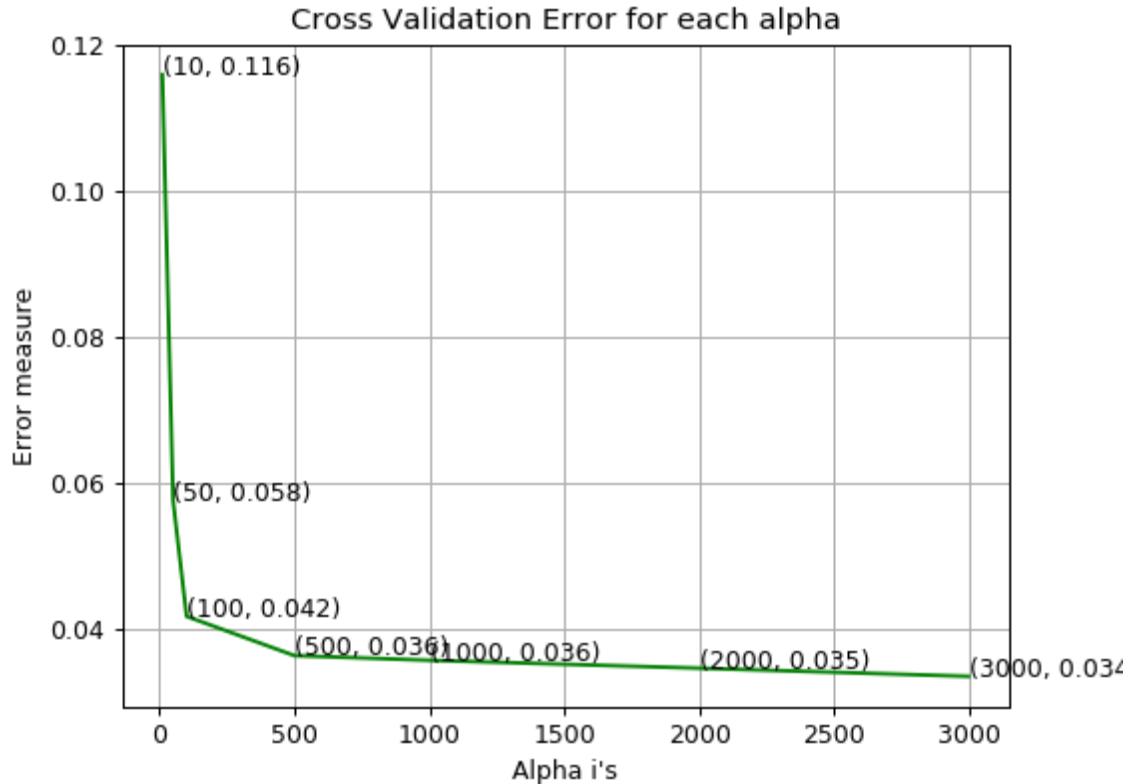
predict_y = sig_clf.predict_proba(X_train_asm)

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

```

log\_loss for c = 10 is 0.11588105338340265  
 log\_loss for c = 50 is 0.057658250882591494  
 log\_loss for c = 100 is 0.04186141305711363  
 log\_loss for c = 500 is 0.03649854125696994  
 log\_loss for c = 1000 is 0.035859619519393905  
 log\_loss for c = 2000 is 0.03478236752207586  
 log\_loss for c = 3000 is 0.033667303437409195

&lt;IPython.core.display.Javascript object&gt;



For values of best alpha = 3000 The train log loss is: 0.00982726018742022

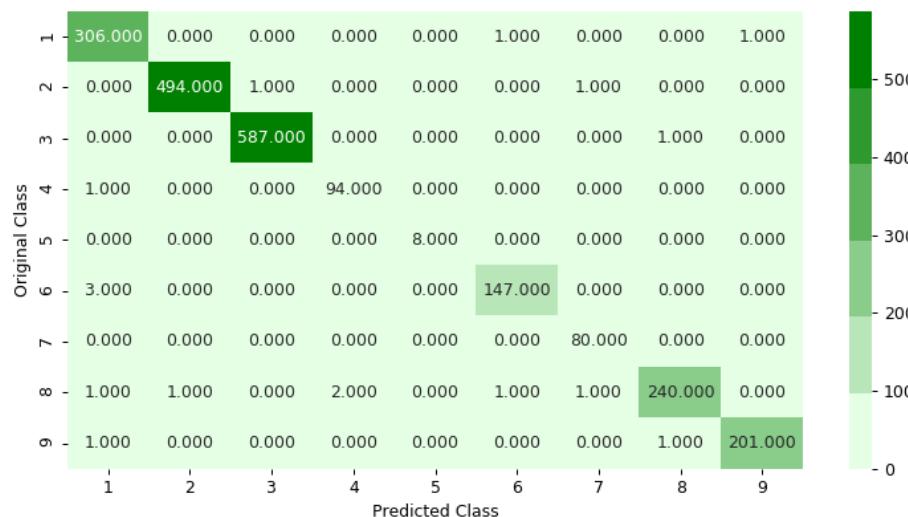
For values of best alpha = 3000 The cross validation log loss is: 0.0336673  
03437409195

For values of best alpha = 3000 The test log loss is: 0.042877055973511075

Number of misclassified points 0.78196872125115

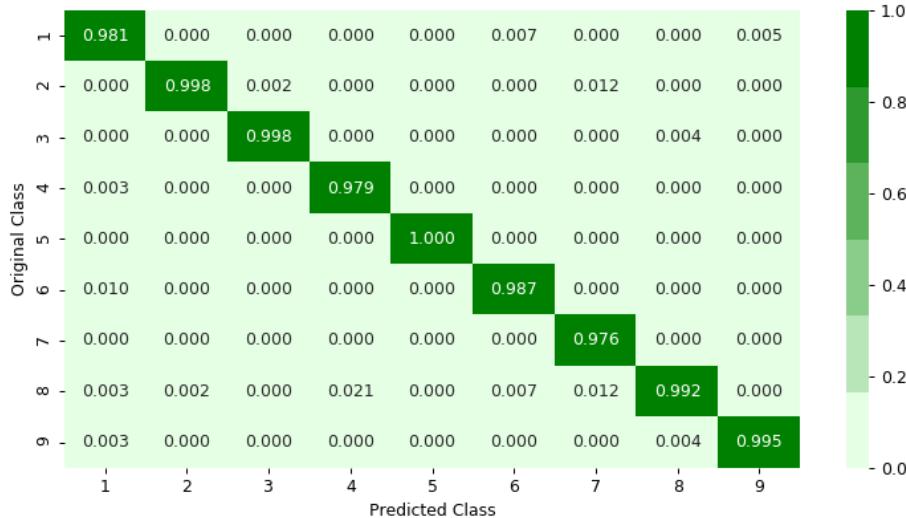
----- Confusion matrix -----

<IPython.core.display.Javascript object>



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

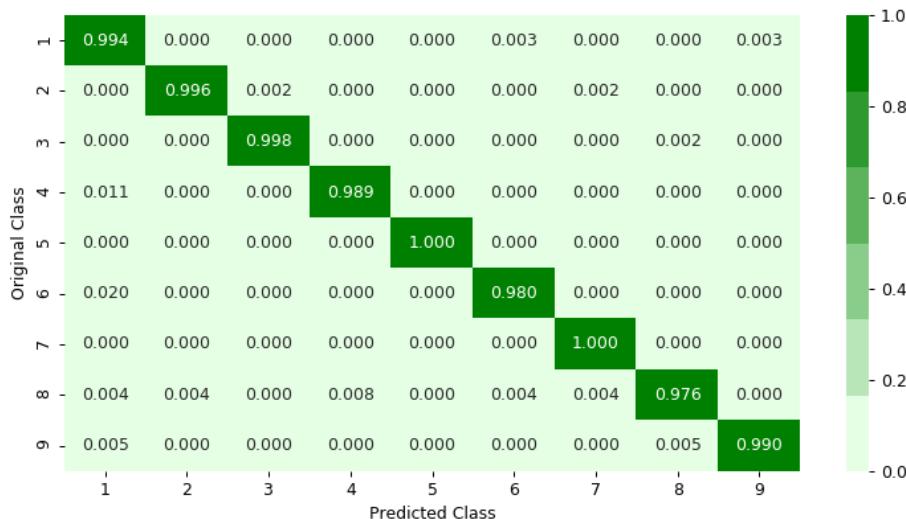
<IPython.core.display.Javascript object>



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 [39]:

```
x_cfl=XGBClassifier()

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  5.6min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  9.3min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 15.6min remaining:  1.7min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 16.7min finished
```

Out[39]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                    estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bytree=1,
                                            colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
                                            max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
                                            n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
                                            reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                                            silent=True, subsample=1),
                    fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
                    param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]}, pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score='warn', scoring=None, verbose=10)
```

In [40]:

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

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 10, 'learning_rate': 0.01, 'colsample_bytree': 0.5}
```

In [42]:

```
x_cfl=XGBClassifier(n_estimators=2000,subsample=0.5,learning_rate=0.01,colsample_bytree=0.5
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 0.010626478719576738  
 cv loss 0.033016089856804966  
 test loss 0.04082419520278345

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

### 4.5.1. Merging both asm and byte file features

In [21]:

result.head()

Out[21]:

	Unnamed: 0	ID	0	1	2	3	4
0	0.000000	01azqd4lnC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048
1	0.000092	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342

5 rows × 261 columns

In [22]:

```
result_asm.head()
```

Out[22]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BoxQeS5Rju		19	744	0	127	57	0	323	0
1	1E93CpP60RHFNiT5Qfvn		17	838	0	103	49	0	0	3
2	3ekVow2ajZHbTnBcsDfX		17	427	0	50	43	0	145	0
3	3X2nY7iQaPB1WDrAZqJe		17	227	0	43	19	0	0	3
4	46OZzdsSKDCFV8h7XWxf		17	402	0	59	170	0	0	3

5 rows × 54 columns

In [23]:

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

(10868, 261)  
(10868, 54)

In [24]:

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

Out[24]:

	Unnamed: 0	0	1	2	3	4	5	6	7
0	0.000000	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946
1	0.000092	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984
2	0.000184	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155
3	0.000276	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481
4	0.000368	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229

5 rows × 308 columns

In [25]:

```
result_y.head()
```

Out[25]:

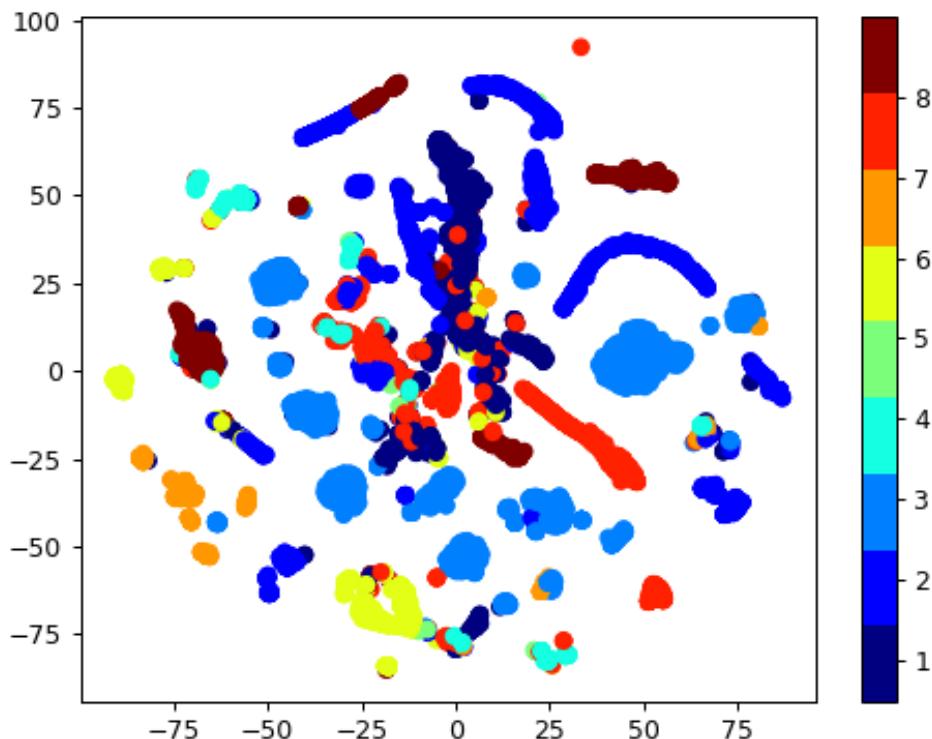
```
0    9  
1    2  
2    9  
3    1  
4    8  
Name: Class, dtype: int64
```

#### 4.5.2. Multivariate Analysis on final features

In [25]:

```
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 [26]:

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

#### 4.5.4. Random Forest Classifier on final features

In [34]:

```

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

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

best_alpha = np.argmin(cv_log_error_array)

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

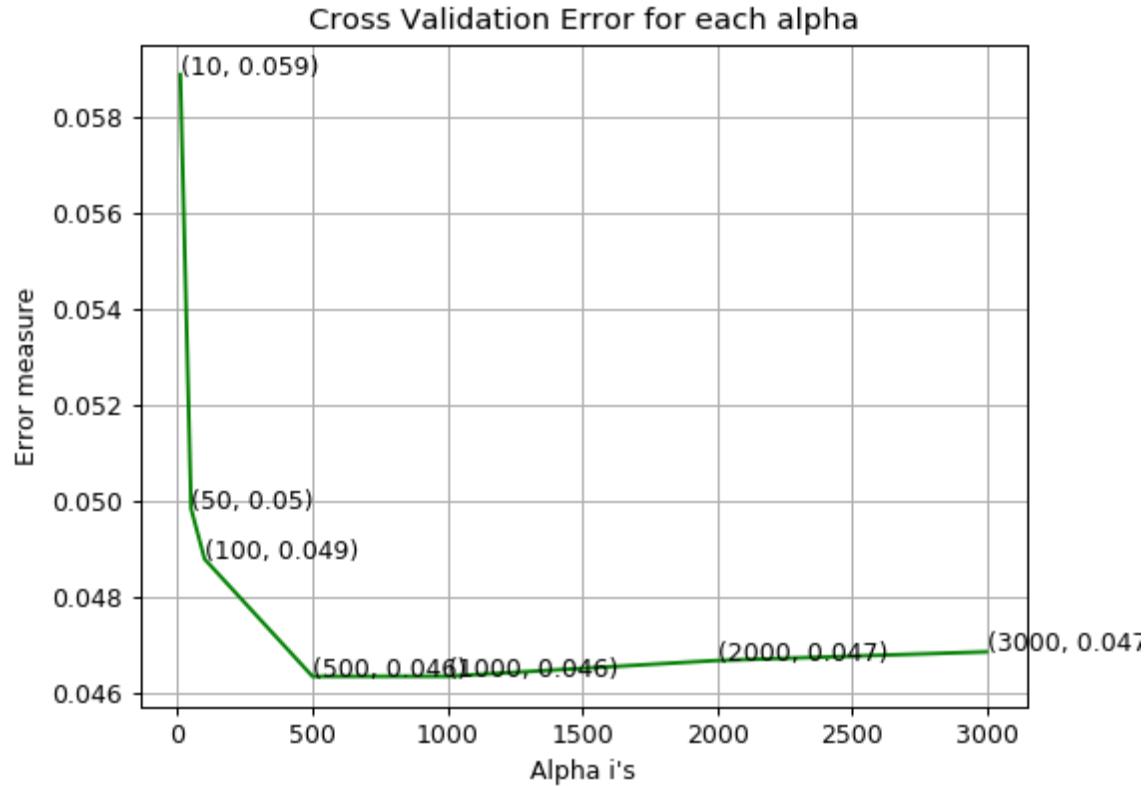
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```

log\_loss for c = 10 is 0.058864988008900165  
log\_loss for c = 50 is 0.04982171352389583  
log\_loss for c = 100 is 0.04877439563993806  
log\_loss for c = 500 is 0.04633136949419593  
log\_loss for c = 1000 is 0.04633282669842955  
log\_loss for c = 2000 is 0.04666148931304081  
log\_loss for c = 3000 is 0.04684161733430787

<IPython.core.display.Javascript object>



For values of best alpha = 500 The train log loss is: 0.015045746557915482

For values of best alpha = 500 The cross validation log loss is: 0.04633136949419593

For values of best alpha = 500 The test log loss is: 0.0419437056294099

#### 4.5.5. XgBoost Classifier on final features

In [35]:

```

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

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

```

best\_alpha = np.argmin(cv\_log\_error\_array)

```

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

```

```

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

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

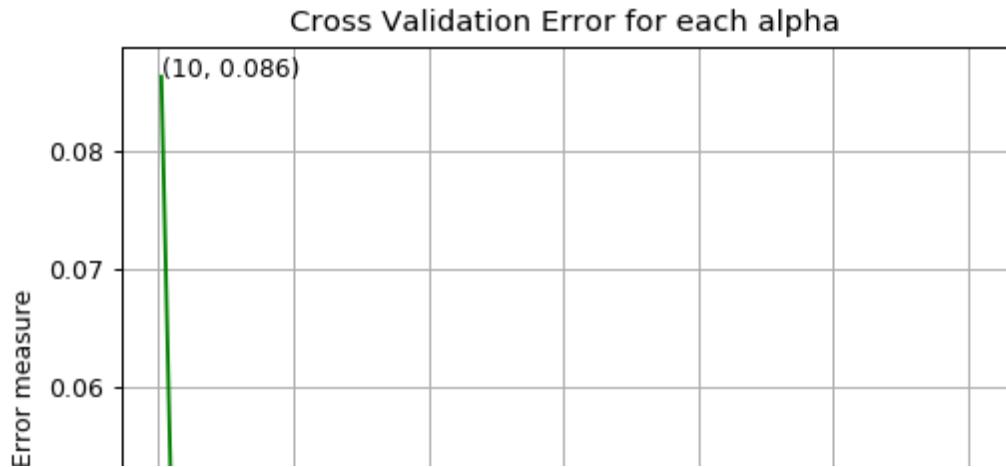
```

```

log_loss for c = 10 is 0.08634410259197668
log_loss for c = 50 is 0.0467962200270487
log_loss for c = 100 is 0.03846464669244138
log_loss for c = 500 is 0.03542509345482663
log_loss for c = 1000 is 0.03524790113745623
log_loss for c = 2000 is 0.03537820448736872
log_loss for c = 3000 is 0.035384159245550155

```

&lt;IPython.core.display.Javascript object&gt;



For values of best alpha = 1000 The train log loss is: 0.010771162453744454

For values of best alpha = 1000 The cross validation log loss is: 0.035384159245550155

For values of best alpha = 1000 The test log loss is: 0.024834218493213808

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

In [36]:

```
x_cfl=XGBClassifier()

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 11.0min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 17.5min
[Parallel(n_jobs=-1)]: Done  27 out of 30 | elapsed: 28.6min remaining:  3.2min
[Parallel(n_jobs=-1)]: Done  30 out of 30 | elapsed: 37.0min finished
```

Out[36]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                    estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                                            colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
                                            max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
                                            n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
                                            reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                                            silent=True, subsample=1),
                    fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
                    param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]}, pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score='warn', scoring=None, verbose=10)
```

In [37]:

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

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 0.5}
```

In [39]:

```
x_cfl=XGBClassifier(n_estimators=1000,max_depth=5,learning_rate=0.1,colsample_bytree=0.5,su
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

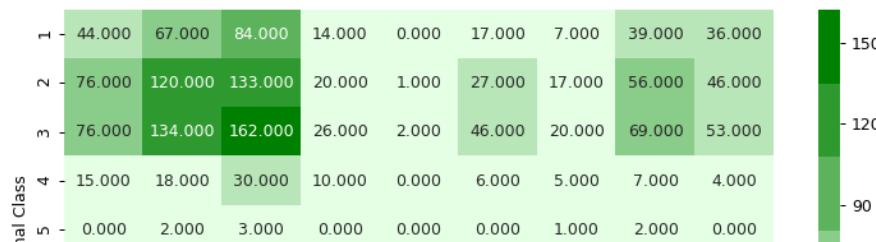
For values of best alpha = 1000 The train log loss is: 0.0108499380402810  
54

For values of best alpha = 1000 The cross validation log loss is: 0.03269  
108838914283

For values of best alpha = 1000 The test log loss is: 0.02814277993749233  
Number of misclassified points 81.73873045078197

----- Confusion matrix -----

<IPython.core.display.Javascript object>



## byte features

In [25]:

```
result_x['ID'] = result.ID
```

In [23]:

```
byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19
```

In [29]:

```
def byte_bigram():
    byte_bigram_vocab = []
    for i, v in enumerate(byte_vocab.split(',')):
        for j in range(0, len(byte_vocab.split(',')-1)):
            byte_bigram_vocab.append(v + ' ' + byte_vocab.split(',')[-1])
len(byte_bigram_vocab)
```

In [26]:

byte\_bigram()

Out[26]:

66049

In [27]:

byte\_bigram\_vocab[:5]

Out[27]:

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

In [30]:

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

In [6]:

byte\_trigram()

Out[6]:

16974593

In [7]:

byte\_trigram\_vocab[:5]

Out[7]:

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

In [28]:

```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
```

In [38]:

```
vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
bytebigram_vect = scipy.sparse.csr_matrix((10868, 66049))
for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
    f = open('byteFiles/' + file)
    a[i:] += scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()])
    f.close()
```

10868it [3:49:23, 2.10it/s]

In [39]:

```
bytebigram_vect
```

Out[39]:

```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'  
with 0 stored elements in Compressed Sparse Row format>
```

In [40]:

```
scipy.sparse.save_npz('bytebigram.npz', bytebigram_vect)
```

In [30]:

```
from sklearn.preprocessing import normalize  
byte_bigram_vect = normalize(scipy.sparse.load_npz('bytebigram.npz'), axis = 0)
```

◀ ▶

## N-Gram(2-Gram, 3-Gram, 4-Gram) Opcode Vectorization

In [31]:

```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
```

◀ ▶

In [31]:

```
def asmopcodebigram():  
    asmopcodebigram = []  
    for i, v in enumerate(opcodes):  
        for j in range(0, len(opcodes)):  
            asmopcodebigram.append(v + ' ' + opcodes[j])  
    len(asmopcodebigram)
```

In [32]:

```
asmopcodebigram
```

Out[32]:

676

In [33]:

```
def asmopcodetrigram():
    asmopcodetrigram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            for k in range(0, len(opcodes)):
                asmopcodetrigram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k])
len(asmopcodetrigram)
```

In [33]:

asmopcodetrigram

Out[33]:

17576

In [34]:

```
def asmopcodetetragram():
    asmopcodetetragram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            for k in range(0, len(opcodes)):
                for l in range(0, len(opcodes)):
                    asmopcodetetragram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k] + ' ')
len(asmopcodetetragram)
```

In [34]:

asmopcodetetragram

Out[34]:

456976

In [ ]:

```
def opcode_collect():
    op_file = open("opcode_file.txt", "w+")
    for asmfile in os.listdir('asmFiles'):
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
opcode_collect()
```

In [47]:

```
vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asmopcodebigram)
opcodebivect = scipy.sparse.csr_matrix((10868, len(asmopcodebigram)))
raw_opcode = open('opcode_file.txt').read().split('\n')

for indx in range(10868):
    opcodebivect[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
```

In [48]:

```
opcodebivect
```

Out[48]:

```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'  
with 1877309 stored elements in Compressed Sparse Row format>
```

In [49]:

```
scipy.sparse.save_npz('opcodebigram.npz', opcodebivect)
```

In [51]:

```
vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asmopcodetrigram)
opcodetrivect = scipy.sparse.csr_matrix((10868, len(asmopcodetrigram)))

for indx in range(10868):
    opcodetrivect[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
```

In [52]:

```
opcodetrivect
```

Out[52]:

```
<10868x17576 sparse matrix of type '<class 'numpy.float64'>'  
with 7332672 stored elements in Compressed Sparse Row format>
```

In [53]:

```
scipy.sparse.save_npz('opcodetrigram.npz', opcodetrivect)
```

In [54]:

```
vect = CountVectorizer(ngram_range=(4, 4), vocabulary = asmopcodetetragram)
opcodetetraVect = scipy.sparse.csr_matrix((10868, len(asmopcodetetragram)))

for indx in range(10868):
    opcodetetraVect[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
```

In [55]:

opcodedetraVect

Out[55]:

```
<10868x456976 sparse matrix of type '<class 'numpy.float64'>'  
with 16605229 stored elements in Compressed Sparse Row format>
```

In [56]:

scipy.sparse.save\_npz('opcodedetragram.npz', opcodedetraVect)

In [35]:

opcodedetraVect = scipy.sparse.load\_npz('opcodedetragram.npz')

In [36]:

opcodedetraVect = scipy.sparse.load\_npz('opcodedetragram.npz')

In [37]:

opcodebiVect = scipy.sparse.load\_npz('opcodebigram.npz')

## Image Feature Extraction From ASM Files

In [35]:

import array

In [64]:

```
def collect_img_asm():
    for asmfile in os.listdir("asmFiles"):
        filename = asmfile.split('.')[0]
        file = codecs.open("asmFiles/" + asmfile, 'rb')
        filelen = os.path.getsize("asmFiles/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        scipy.misc.imsave('asm_image/' + filename + '.png', reshaped)
```

In [65]:

collect\_img\_asm()

In [ ]:

```
from IPython.display import Image  
Image(filename='asm_image/deTXH9Zau7qmM0yfYsRS.png')
```

# First 200 Image Pixels

In [38]:

```
import cv2
imagefeatures = np.zeros((10868, 200))
```

In [67]:

```
for i, asmfile in enumerate(os.listdir("asmFiles")):
    img = cv2.imread("asm_image/" + asmfile.split('.')[0] + '.png')
    img_arr = img.flatten()[:200]
    imagefeatures[i, :] += img_arr
```

In [68]:

```
imgfeatures_name = []
for i in range(200):
    img_features_name.append('pix' + str(i))
imgdf = pd.DataFrame(normalize(imagefeatures, axis = 0), columns = imgfeatures_name)
```

In [69]:

```
imgdf['ID'] = result.ID
```

In [70]:

```
imgdf.head()
```

Out[70]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	
0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C
1	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	C
2	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C
3	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C
4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C

5 rows × 201 columns

In [71]:

```
joblib.dump(imgdf, 'img_df')
```

Out[71]:

```
['img_df']
```

In [72]:

```
img_df=joblib.load('img_df')
```

In [73]:

img\_df.head()

Out[73]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	
0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C
1	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	C
2	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C
3	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C
4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	C

5 rows × 201 columns

## Important Feature Selection Using Random Forest

In [38]:

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

## Important Feature Among Opcode Bi-Gram

In [44]:

op\_bi\_indexes = imp\_features(normalize(opcodebivect, axis = 0), asmopcodebigram, 200)

In [45]:

```
op_bi_df = pd.SparseDataFrame(normalize(opcodebivect, axis = 0), columns = asmopcodebigram)
for col in op_bi_df.columns:
    if col not in np.take(asmopcodebigram, op_bi_indexes):
        op_bi_df.drop(col, axis = 1, inplace = True)
```

In [46]:

```
op_bi_df.to_dense().to_csv('op_bi.csv')
```

In [47]:

```
op_bi_df = pd.read_csv('op_bi.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [48]:

```
op_bi_df['ID'] = result.ID
op_bi_df.head()
```

Out[48]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp sub	jmp dec	jmp add	jmp
0	0.031815	0.003894	0.000000	0.00042	0.000000	0.002374	0.00895	0.001268	0.016752	0.00
1	0.000000	0.000649	0.000000	0.00021	0.000374	0.000419	0.00000	0.000000	0.001971	0.00
2	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00
3	0.000000	0.000101	0.000000	0.00007	0.000000	0.000279	0.00000	0.000000	0.000000	0.00
4	0.000362	0.001156	0.001467	0.00028	0.000374	0.000140	0.00000	0.000000	0.000000	0.00

5 rows × 201 columns

## Important Feature Among Opcode 3-Gram

In [39]:

```
op_tri_indexes = imp_features(normalize(opcodetrivect, axis = 0), asmopcodetrigram, 200)
```

In [40]:

```
op_tri_df = pd.SparseDataFrame(normalize(opcodetrivect, axis = 0), columns = asmopcodetrigram)
op_tri_df = op_tri_df.loc[:, np.intersect1d(op_tri_df.columns, np.take(asmopcodetrigram, op
```

In [41]:

```
op_tri_df.to_dense().to_csv('op_tri.csv')
```

In [42]:

```
op_tri_df = pd.read_csv('op_tri.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [43]:

```
op_tri_df['ID'] = result.ID
op_tri_df.head()
```

Out[43]:

	add cmp jmp	add mov add	add mov cmp	add mov jmp	add mov mov	add pop call	add pop mov	add pop pop	add pop push	ad pc ret
0	0.000000	0.002183	0.001340	0.001563	0.003593	0.0	0.005354	0.000342	0.000000	0.000000
1	0.000000	0.001364	0.000670	0.000625	0.002705	0.0	0.001785	0.000000	0.000000	0.000002
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000
4	0.001292	0.001091	0.004914	0.002814	0.014009	0.0	0.000000	0.000000	0.000441	0.000000

5 rows × 201 columns

## Important Feature Among Opcode 4-Gram

In [49]:

```
op_tetra_indexes = imp_features(normalize(opcodetetravect, axis = 0), asmopcodetetragram, 20)
```

In [50]:

```
op_tetra_df = pd.SparseDataFrame(normalize(opcodetetravect, axis = 0), columns = asmopcodetetragram)
op_tetra_df = op_tetra_df.loc[:, np.intersect1d(op_tetra_df.columns, np.take(asmopcodetetragram, op_tetra_indexes))]
```

In [51]:

```
op_tetra_df.to_dense().to_csv('op_tetra.csv')
```

In [52]:

```
op_tetra_df = pd.read_csv('op_tetra.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [53]:

```
op_tetra_df['ID'] = result.ID
op_tetra_df.head()
```

Out[53]:

	add mov add mov	add mov add pop	add mov cmp jnb	add mov mov add	add mov mov mov	add pop push	add pop pop	add pop push	add push	add retn push	call add mov sub	...
0	0.001593	0.007668	0.000000	0.002031	0.002517	0.0	0.0	0.0	0.00116	0.000000	...	
1	0.000000	0.007668	0.000000	0.001625	0.002760	0.0	0.0	0.0	0.00000	0.000000	...	
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.00000	0.000000	...	
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.00000	0.000000	...	
4	0.002125	0.000000	0.023352	0.023558	0.006657	0.0	0.0	0.0	0.00000	0.009682	...	

5 rows × 201 columns

## Important Feature Among Byte Bi-Gram

In [54]:

```
byte_bi_idxes = imp_features(normalize(bytebigram_vect, axis = 0), byte_bigram_vocab, 300)
```

In [55]:

```
np.save('byte_bi_idx', byte_bi_idxes)
```

In [56]:

```
byte_bi_idxes = np.load('byte_bi_idx.npy')
```

In [57]:

```
top_byte_bi = np.zeros((10868, 0))
for i in byte_bi_idxes:
    sliced = bytebigram_vect[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

In [58]:

```
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns = np.take(byte_bigram_vocab, byte_bi_i
```

In [59]:

```
byte_bi_df.to_dense().to_csv('byte_bi.csv')
```

In [60]:

```
byte_bi_df = pd.read_csv('byte_bi.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [61]:

```
byte_bi_df['ID'] = result.ID
```

In [62]:

```
byte_bi_df.head()
```

Out[62]:

	??	55	55	55	55	55	55	55	55	55	55	55	55	55	55	54	54	54	54	54	54	54	54	54
	??	95	b3	b2	b1	b0	af	ae	ad	ac	...	b3	b4	c4	d1	d0	cf	ce	cd	cd	cc	cc	cc	
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 301 columns

## Advanced features

**Adding 300 bytebigram,200 opcode bigram,200 opcode trigram,200 opcode tetragram ,first 200 image pixels**

In [74]:

```
final_data = pd.concat([result_x, op_bi_df, op_tri_df, op_tetra_df, byte_bi_df, img_df], axis=1)
```

In [75]:

```
final_data = final_data.drop('ID', axis = 1)
```

In [76]:

```
final_data.head()
```

Out[76]:

Unnamed: 0	0	1	2	3	4	5	6	7	
0	0.000000	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946
1	0.000092	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984
2	0.000184	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155
3	0.000276	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481
4	0.000368	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229

5 rows × 1408 columns

In [77]:

```
final_data.to_csv('final_data.csv')
```

In [27]:

```
final_data = pd.read_csv('final_data.csv')
```

In [37]:

```
x_train_final, x_test_final, y_train_final, y_test_final = train_test_split(final_data, res_x_trn_final, x_cv_final, y_trn_final, y_cv_final = train_test_split(x_train_final, y_train_
```

## Machine Learning Models on ASM Features + Byte Features + Advanced Features

In [80]:

```

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_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(x_trn_final,y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=logisticR.classes_, eps=0.001))

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

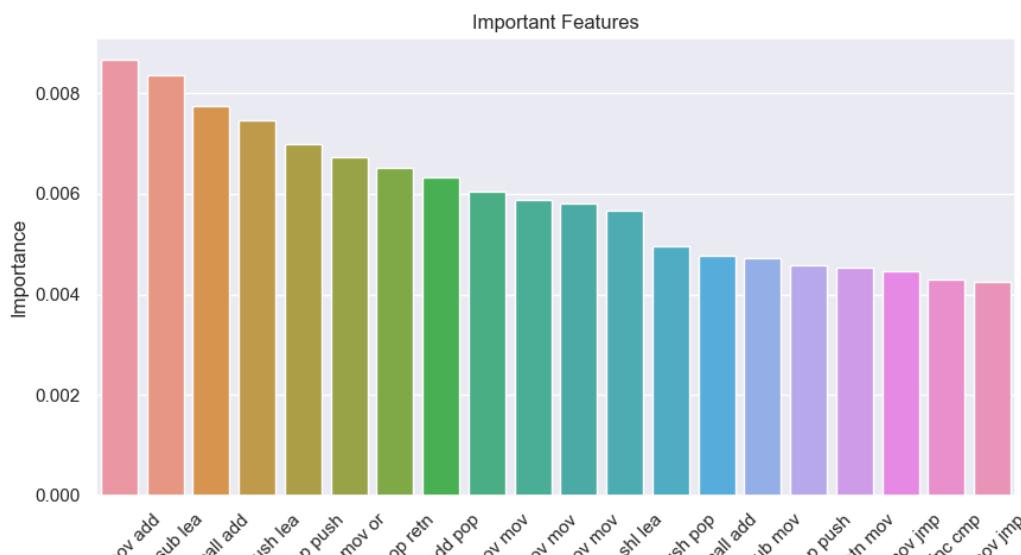
best_alpha = np.argmin(cv_log_error_array)

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()

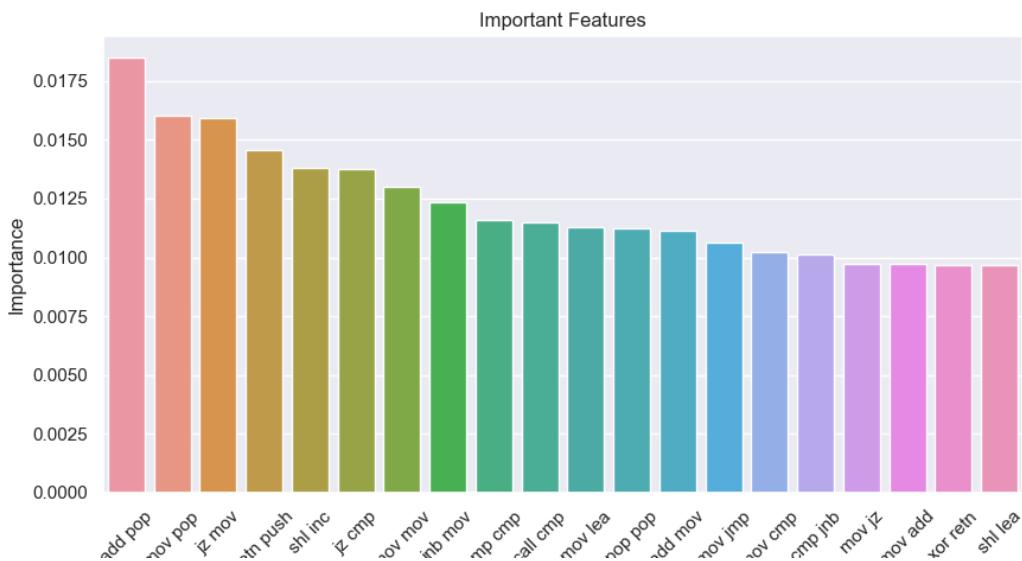
```

log\_loss for c = 1e-05 is 1.1840039867727614  
 log\_loss for c = 0.0001 is 1.1217881098745714  
 log\_loss for c = 0.001 is 1.174936322460997  
 log\_loss for c = 0.01 is 1.0741224260174453  
 log\_loss for c = 0.1 is 1.1761396828975654  
 log\_loss for c = 1 is 1.2362570810343723  
 log\_loss for c = 10 is 1.1804717850739066  
 log\_loss for c = 100 is 1.1684083137157295  
 log\_loss for c = 1000 is 1.1061521197568476

&lt;IPython.core.display.Javascript object&gt;



&lt;IPython.core.display.Javascript object&gt;



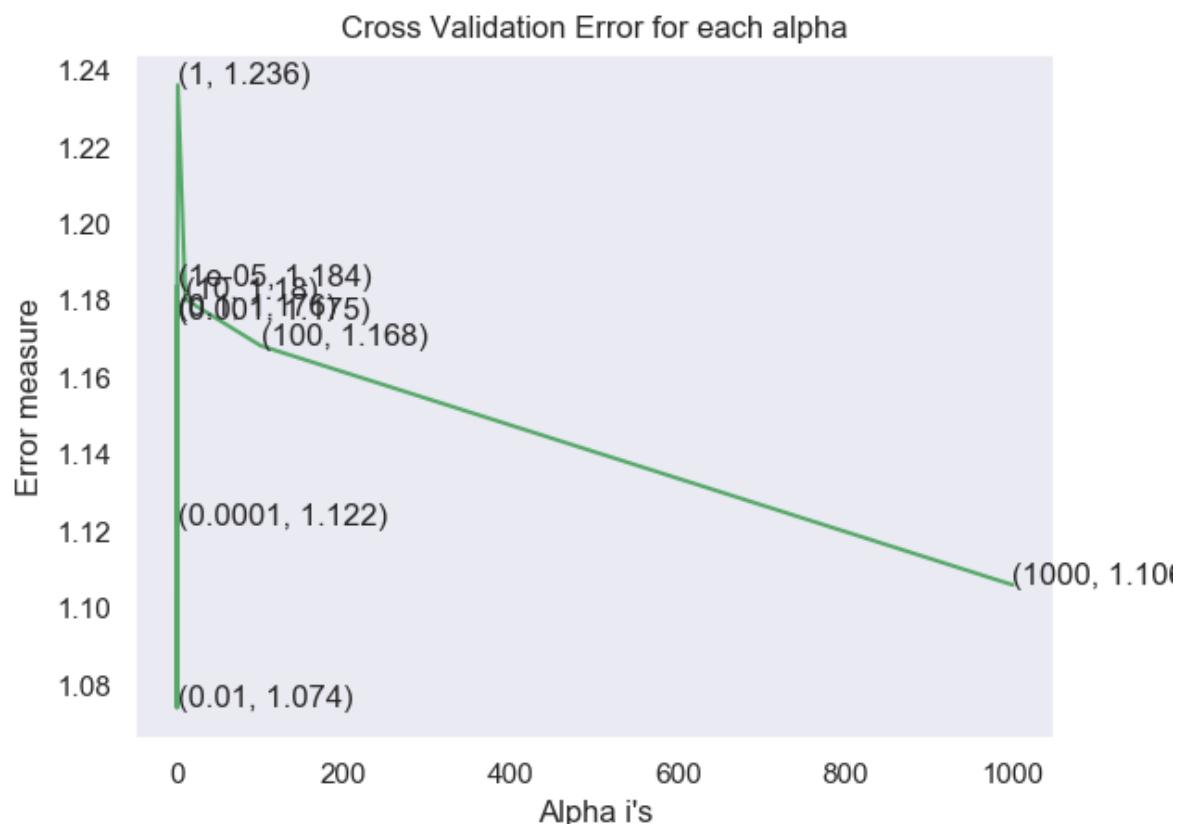
&lt;IPython.core.display.Javascript object&gt;



&lt;IPython.core.display.Javascript object&gt;



&lt;IPython.core.display.Javascript object&gt;



In [35]:

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

predict_y = sig_clf.predict_proba(x_trn_final)
print ('log loss for train data',(log_loss(y_trn_final, predict_y, labels=logisticR.classes_
predict_y = sig_clf.predict_proba(x_cv_final)
print ('log loss for cv data',(log_loss(y_cv_final, predict_y, labels=logisticR.classes_, e
predict_y = sig_clf.predict_proba(x_test_final)
print ('log loss for test data',(log_loss(y_test_final, predict_y, labels=logisticR.classes_
```

C:\Users\Sai charan\Anaconda3\lib\site-packages\sklearn\svm\base.py:922: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.

"the number of iterations.", ConvergenceWarning)

C:\Users\Sai charan\Anaconda3\lib\site-packages\sklearn\svm\base.py:922: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.

"the number of iterations.", ConvergenceWarning)

C:\Users\Sai charan\Anaconda3\lib\site-packages\sklearn\svm\base.py:922: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.

"the number of iterations.", ConvergenceWarning)

C:\Users\Sai charan\Anaconda3\lib\site-packages\sklearn\svm\base.py:922: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.

"the number of iterations.", ConvergenceWarning)

```
log loss for train data 1.193174530266704
log loss for cv data 1.1785070578048291
log loss for test data 1.2060464393477006
```

In [39]:

```
plot_confusion_matrix(y_test_final,sig_clf.predict(x_test_final))
```

Number of misclassified points 31.324747010119598

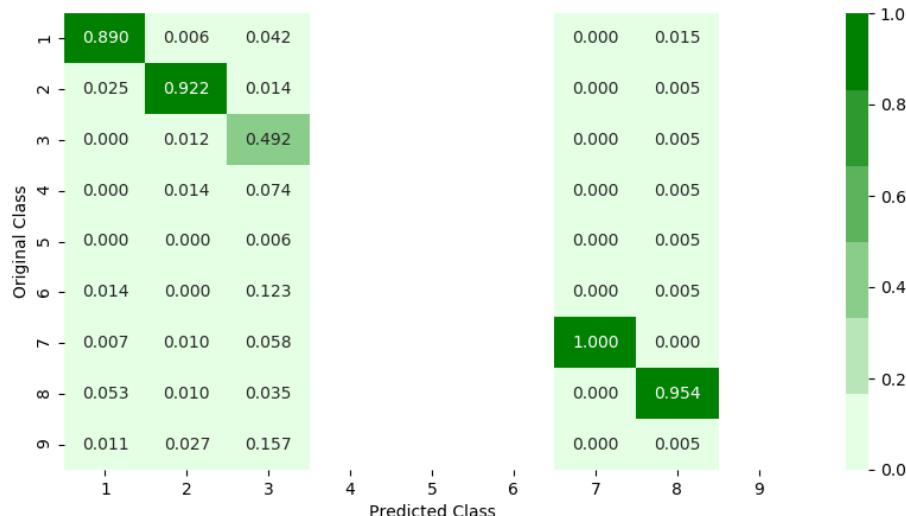
----- Confusion matrix -----

<IPython.core.display.Javascript object>



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

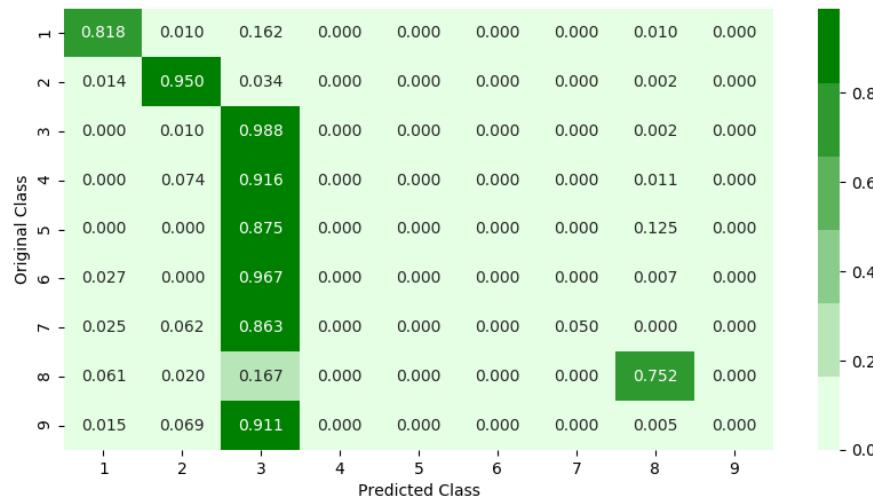
<IPython.core.display.Javascript object>



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

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

<IPython.core.display.Javascript object>



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

In [29]:

```
alpha=[10,100,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(x_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(x_trn_final, y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=x_cfl.classes_, eps=1e-15))

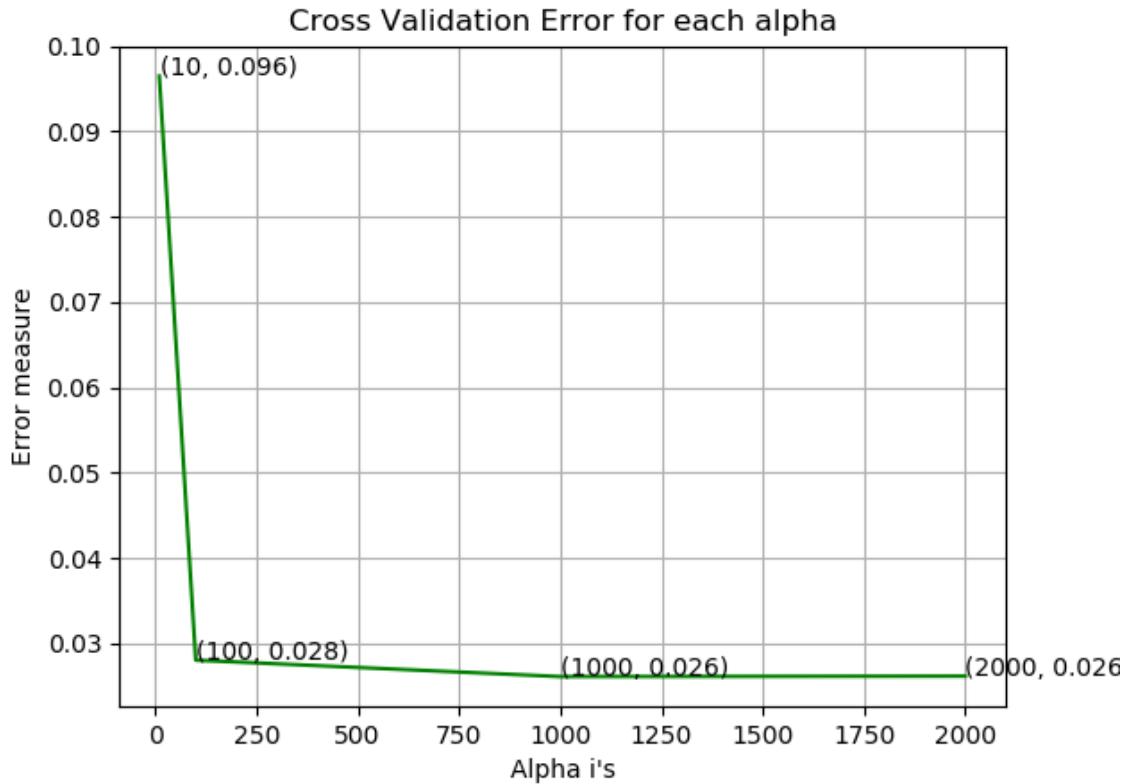
for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c = 10 is 0.09649648467635132
log_loss for c = 100 is 0.028026994875892948
log_loss for c = 1000 is 0.02610102301724636
log_loss for c = 2000 is 0.026155764643162237
```

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



In [84]:

```
x_cfl=XGBClassifier(n_estimators=2000,nthread=-1)
x_cfl.fit(x_trn_final,y_trn_final,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_trn_final, y_trn_final)

predict_y = sig_clf.predict_proba(x_trn_final)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
```

```
For values of best alpha =  0.01 The train log loss is: 0.010187974436441512
For values of best alpha =  0.01 The cross validation log loss is: 0.0239576
2856614576
For values of best alpha =  0.01 The test log loss is: 0.018309505637434106
```

## Results

In [18]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','log loss']
ptable.add_row(["random","Byte files","2.45"])
ptable.add_row(["knn","Byte files","0.48"])
ptable.add_row(["Logistic Regression","Byte files","0.52"])
ptable.add_row(["Random Forest Classifier ","Byte files","0.06"])
ptable.add_row(["XgBoost Classification","Byte files","0.07"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["knn","asmfiles","0.21"])
ptable.add_row(["Logistic Regression","asmfiles","0.38"])
ptable.add_row(["Random Forest Classifier ","asmfiles","0.03"])
ptable.add_row(["XgBoost Classification","asmfiles","0.04"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Random Forest Classifier ","Byte files+asmfiles","0.04"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles","0.02"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Logistic Regression","Byte files+asmfiles+advanced features","1.12"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles+advanced features","0.01"])
print(ptable)
```

loss	Model	Features	log
45	random	Byte files	2.
48	knn	Byte files	0.
52	Logistic Regression	Byte files	0.
06	Random Forest Classifier	Byte files	0.
07	XgBoost Classification	Byte files	0.
21	knn	asmfiles	0.
38	Logistic Regression	asmfiles	0.
03	Random Forest Classifier	asmfiles	0.
04	XgBoost Classification	asmfiles	0.
04	Random Forest Classifier	Byte files+asmfiles	0.
	XgBoost Classification	Byte files+asmfiles	0.

02				
12				
01	XgBoost Classification	Byte files+asmfiles+advanced features	0.	
+-----+	+-----+	+-----+	+-----+	
-+-----+				

## Conclusion

- First I focused the bytefiles and asmfiles from train.7z
- then I explored and extracted features from the data and applied ML models on bytes file.
- Similarly explored and extracted features from the data and applied ML models on asm file.
- And then combined the bytes files and asm files and applied ML models on the combined features.
- Finally extracted byte bigram,,opcode bi gram,opcode trigram,opcode tetra gram and 200 pixels of asm image and applied ML models on them.

In [ ]: