Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- Source : https://www.kaggle.com/c/malware-classification/data
- For every malware, we have two files
  1. .asm file (read more: https://www.reviversoft.com/file-extensions/asm)
  2. .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)

- Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:
- **Lots of Data for a single-box/computer.**
- There are total 10,868 .bytes files and 10,868 asm files total 21,736 files
- There are 9 types of malwares (9 classes) in our give data
- Types of Malware:
  1. Ramnit
  2. Lollipop
  3. Kelihos_ver3
  4. Vundo
  5. Simda
  6. Tracur
  7. Kelihos_ver1
  8. Obfuscator.ACY
  9. Gatak

2.1.2. Example Data Point

.asm file

```
.text:00401000                                       assume es:nothing, s
s:nothing, ds:_data,    fs:nothing, gs:nothing
.text:00401000 56                                   push    esi
.text:00401001 8D 44 24    08                               lea     eax,
[esp+8]
.text:00401005 50                                   push    eax
.text:00401006 8B F1                                   mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                               call    ??
0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * cons
t &)
.text:0040100D C7 06 08    BB 42 00                           mov     dwo
rd 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 10
h
.text:00401020 C7 01 08    BB 42 00                           mov     dwo
rd ptr [ecx],    offset off_42BB08
.text:00401026 E9 26 1C    00 00                               jmp     su
b_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     dwo
rd ptr [esi],    offset off_42BB08
.text:00401039 E8 13 1C    00 00                               call    su
b_402C51
.text:0040103E F6 44 24    08 01                               test    by
te ptr    [esp+8], 1
.text:00401043 74 09                                   jz      short loc_
40104E
.text:00401045 56                                   push    esi
.text:00401046 E8 6C 1E    00 00                               call    ??
3@YAXPAX@Z    ; operator delete(void *)
.text:0040104B 83 C4 04                                   add     esp, 4
.text:0040104E
.text:0040104E                               loc_40104E:
  ; CODE XREF: .text:00401043□j
.text:0040104E 8B C6                                   mov     eax, esi
.text:00401050 5E                                   pop     esi
.text:00401051 C2 04 00                                   retn    4
.text:00401051                               ; --------------------------
-------------------------------------------------
```

.bytes file

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

3. Exploratory Data Analysis

In [2]:

```python
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 [10]:

```python
#separating byte files and asm files

source = 'train'
destination = 'byteFiles'

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

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

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

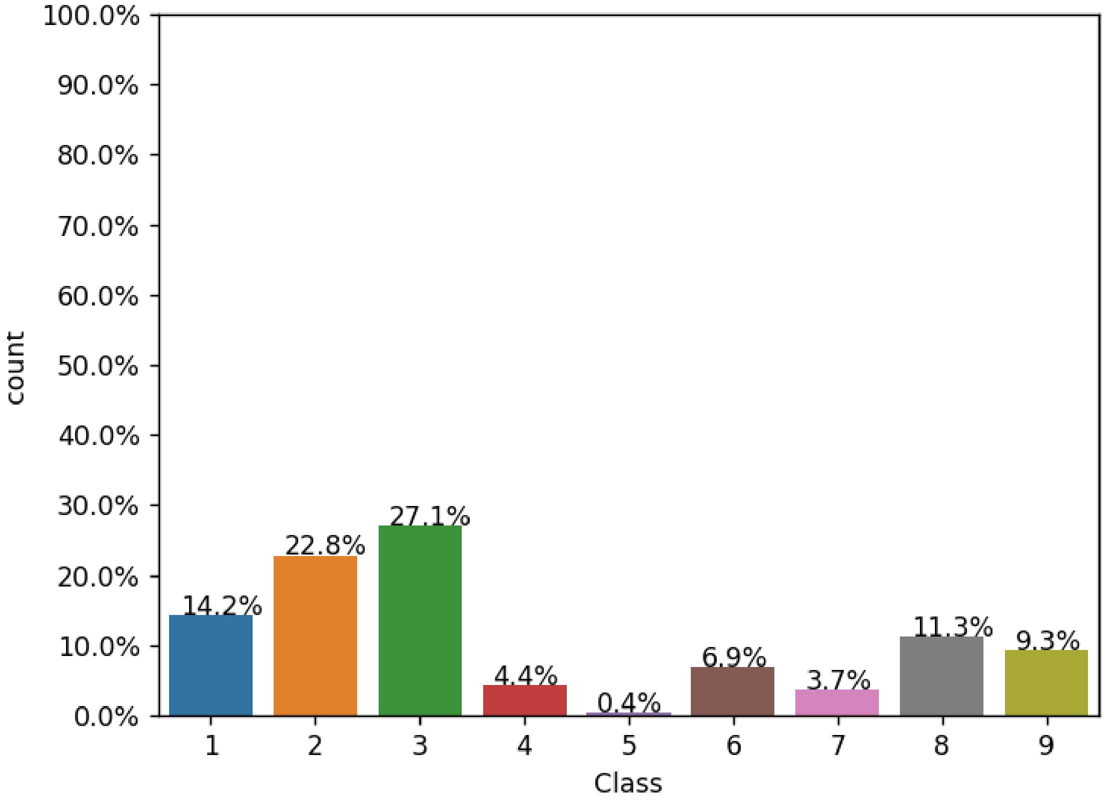
```
01azqd4InC7m9JpocGv5.asm
01azqd4InC7m9JpocGv5.bytes
asmFiles\01azqd4InC7m9JpocGv5.bytes
01IsoiSMh5gxyDYTl4CB.asm
01IsoiSMh5gxyDYTl4CB.bytes
asmFiles\01IsoiSMh5gxyDYTl4CB.bytes
01jsnpXSAlgw6aPeDxrU.asm
01jsnpXSAlgw6aPeDxrU.bytes
asmFiles\01jsnpXSAlgw6aPeDxrU.bytes
01kcPWA9K2BOxQeS5Rju.asm
01kcPWA9K2BOxQeS5Rju.bytes
asmFiles\01kcPWA9K2BOxQeS5Rju.bytes
01SuzwMJEIXsK7A8dQbl.asm
01SuzwMJEIXsK7A8dQbl.bytes
asmFiles\01SuzwMJEIXsK7A8dQbl.bytes
02IOCvYEy8mjiuAQHax3.asm
02IOCvYEy8mjiuAQHax3.bytes
asmFiles\02IOCvYEy8mjiuAQHax3.bytes
02JqQ7H3yEoD8viYWlmS.asm
```

3.1. Distribution of malware classes in whole data set

In [6]:

```
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

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

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [7]:

```python
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_s
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file na
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes}
print (data_size_byte.head())
```

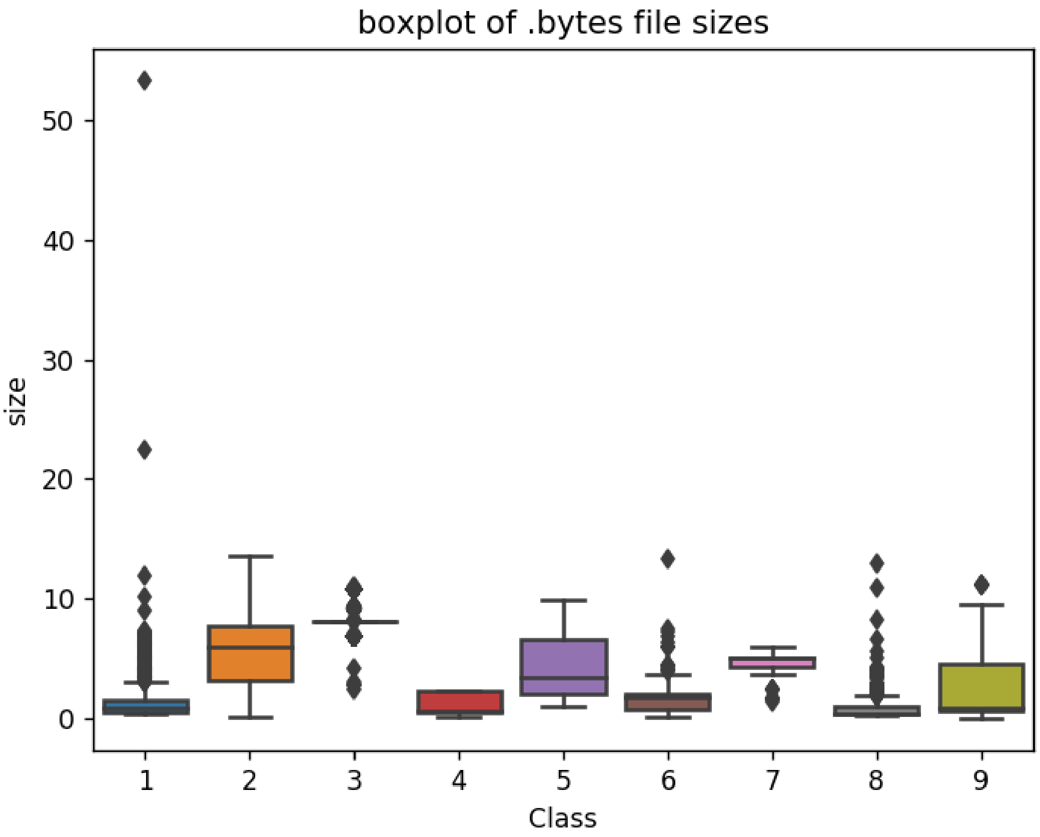
```
   Class                    ID      size
0      9  01azqd4InC7m9JpocGv5  0.000000
1      2  01IsoiSMh5gxyDYTl4CB  5.538818
2      9  01jsnpXSAlgw6aPeDxrU  3.887939
3      1  01kcPWA9K2BOxQeS5Rju  0.574219
4      8  01SuzwMJEIXsK7A8dQbl  0.370850
```

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

In [8]:

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

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



3.2.3 feature extraction from byte files

In [ ]:

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

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        tfile=file.split('.')[0]
        text_file = open('byteFiles/'+tfile+".txt", 'w+')
        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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,
for file in files:
    filenames2.append(f)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i)+",")
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [9]:

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

```
                     ID       0     1     2     3     4     5     6     7  \
0  01azqd4InC7m9JpocGv5  601905  3905  2816  3832  3345  3242  3650  3201   
1  01IsoiSMh5gxyDYTl4CB   39755  8337  7249  7186  8663  6844  8420  7589   
2  01jsnpXSAlgw6aPeDxrU   93506  9542  2568  2438  8925  9330  9007  2342   
3  01kcPWA9K2BOxQeS5Rju   21091  1213   726   817  1257   625   550   523   
4  01SuzwMJEIXsK7A8dQbl   19764   710   302   433   559   410   262   249   

      8  ...     f7    f8    f9    fa    fb    fc    fd     fe     ff     ??  

0  2965  ...   2804  3687  3101  3211  3097  2758  3099   2759   5753   1824  

1  9291  ...    451  6536   439   281   302  7639   518  17001  54902   8588  

2  9107  ...   2325  2358  2242  2885  2863  2471  2786   2680  49144    468  

3  1078  ...    478   873   485   462   516  1133   471    761   7998  13940  

4   422  ...    847   947   350   209   239   653   221    242   2199   9008  


[5 rows x 258 columns]
```

In [10]:

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

Out[10]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	32
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	2
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2242	28
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	485	4
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	350	2

5 rows × 260 columns

In [11]:

```
result.to_csv("bytefiles_size.csv", index=False)
```

In [12]:

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

In [13]:

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

Out[13]:

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

5 rows × 260 columns

In [14]:

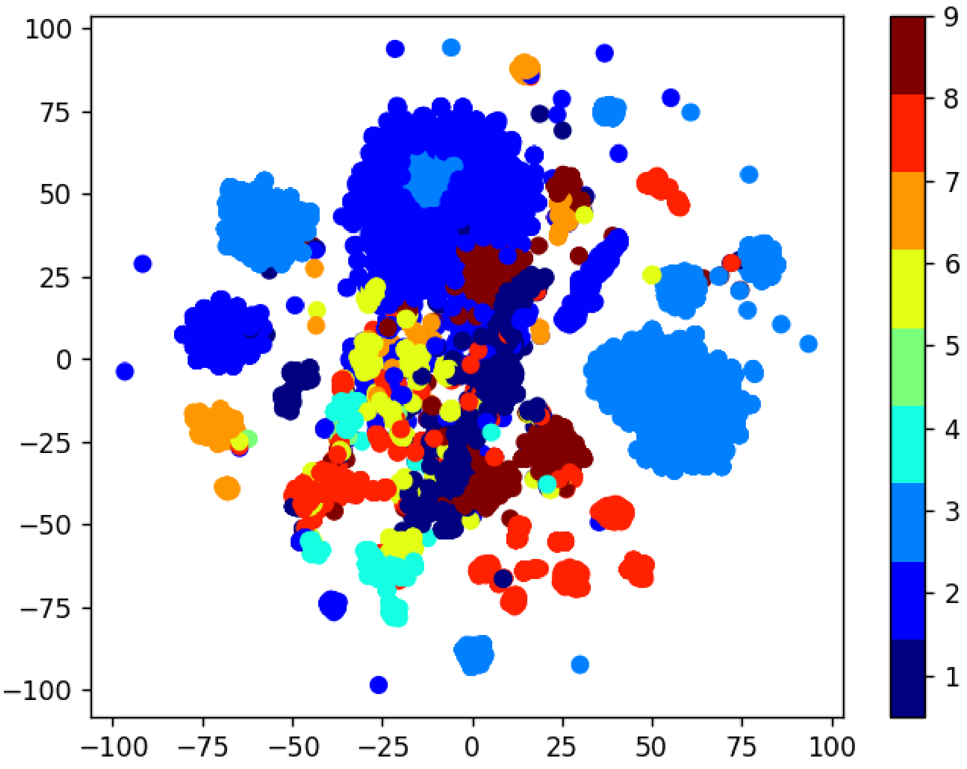
```python
result.to_csv('bytefiles_size_normal.csv', index=False)
```

3.2.4 Multivariate Analysis

In [58]:

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

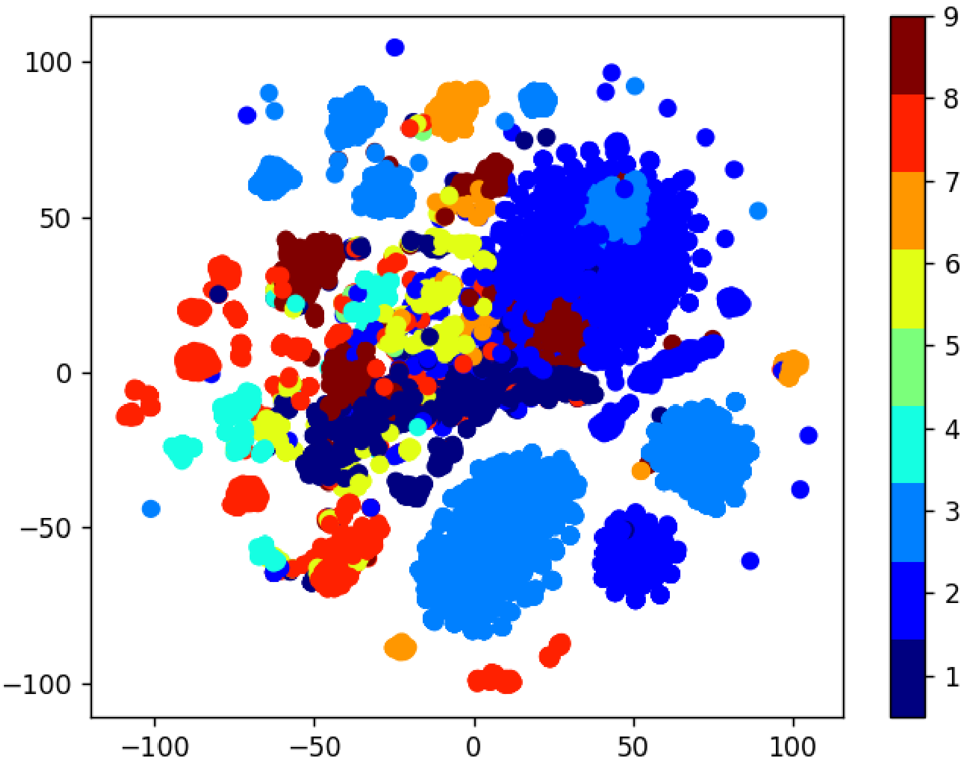
```
<IPython.core.display.Javascript object>
```



In [59]:

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

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



Train Test split

In [60]:

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

In [63]:

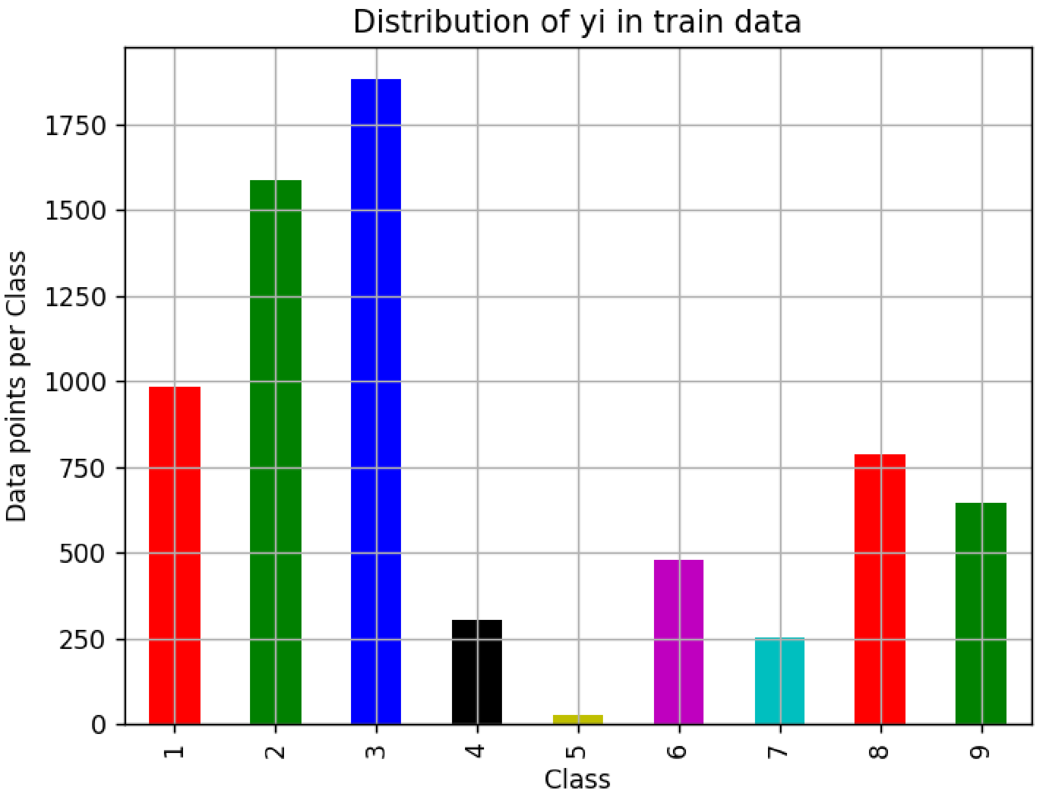
```python
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

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

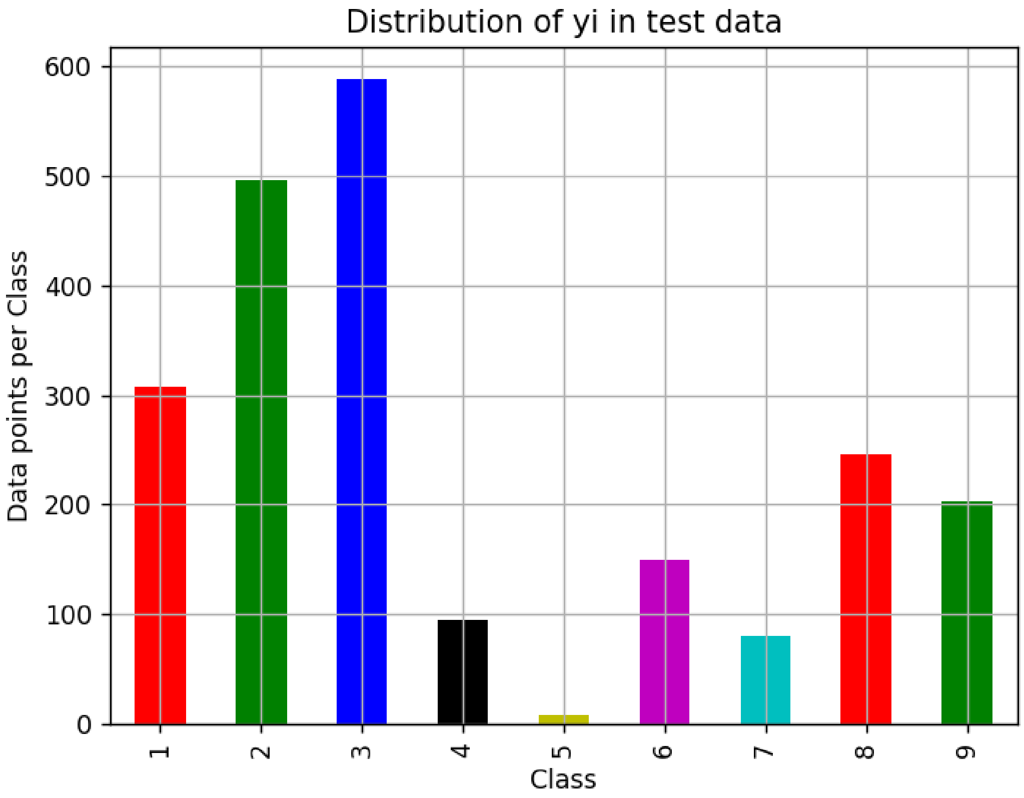
In [67]:

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

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
# -(train_class_distribution.values): the minus sign will give us in decreasir
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.


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
# -(train_class_distribution.values): the minus sign will give us in decreasir
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.v

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
# -(train_class_distribution.values): the minus sign will give us in decreasir
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.val

```

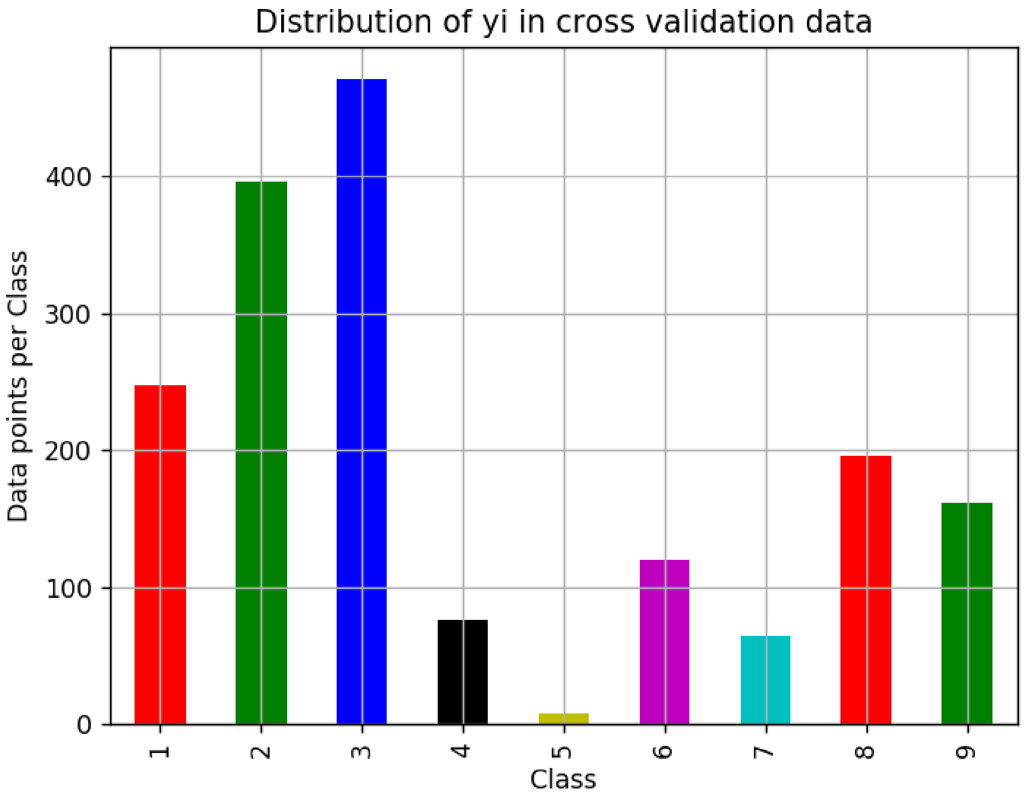
```
<IPython.core.display.Javascript object>
```



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

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



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

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



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

In [61]:

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

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

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

4. Machine Learning Models

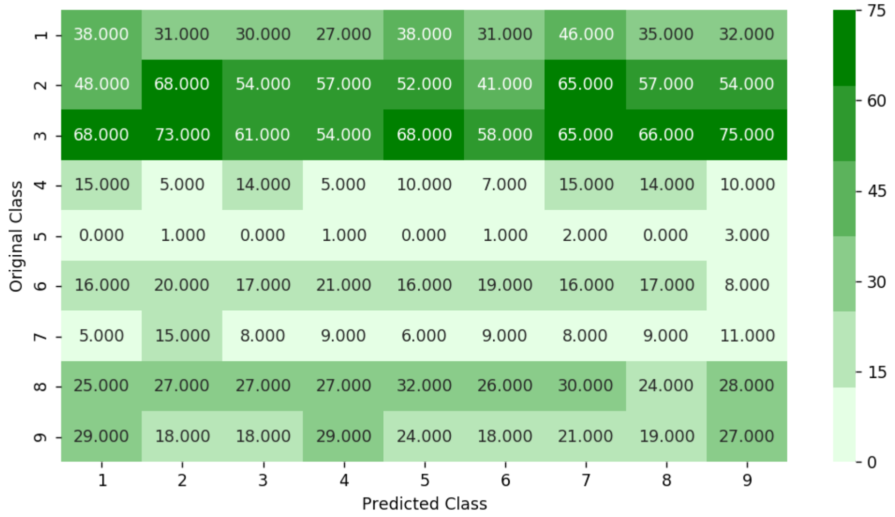
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

In [62]:

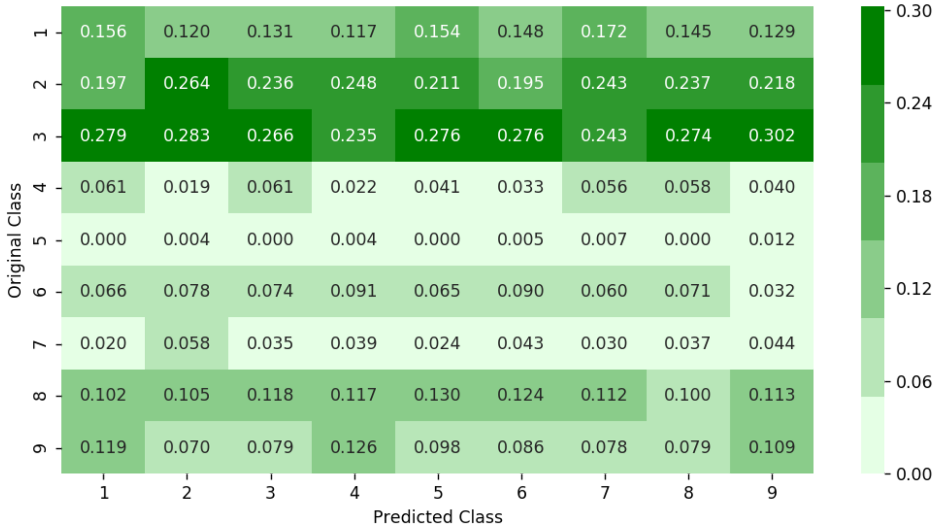
```python
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
# ref: https://stackoverflow.com/a/18662466/4084039

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

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


# 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_predicte

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

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

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

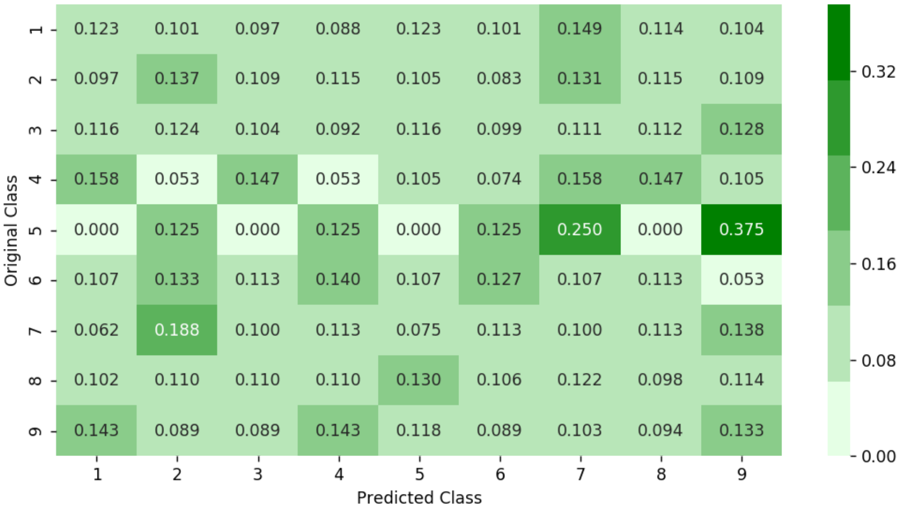


	1	2	3	4	5	6	7	8	9
1	38.000	31.000	30.000	27.000	38.000	31.000	46.000	35.000	32.000
2	48.000	68.000	54.000	57.000	52.000	41.000	65.000	57.000	54.000
3	68.000	73.000	61.000	54.000	68.000	58.000	65.000	66.000	75.000
4	15.000	5.000	14.000	5.000	10.000	7.000	15.000	14.000	10.000
5	0.000	1.000	0.000	1.000	0.000	1.000	2.000	0.000	3.000
6	16.000	20.000	17.000	21.000	16.000	19.000	16.000	17.000	8.000
7	5.000	15.000	8.000	9.000	6.000	9.000	8.000	9.000	11.000
8	25.000	27.000	27.000	27.000	32.000	26.000	30.000	24.000	28.000
9	29.000	18.000	18.000	29.000	24.000	18.000	21.000	19.000	27.000

```
-------------------------------------------------- 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.]
```

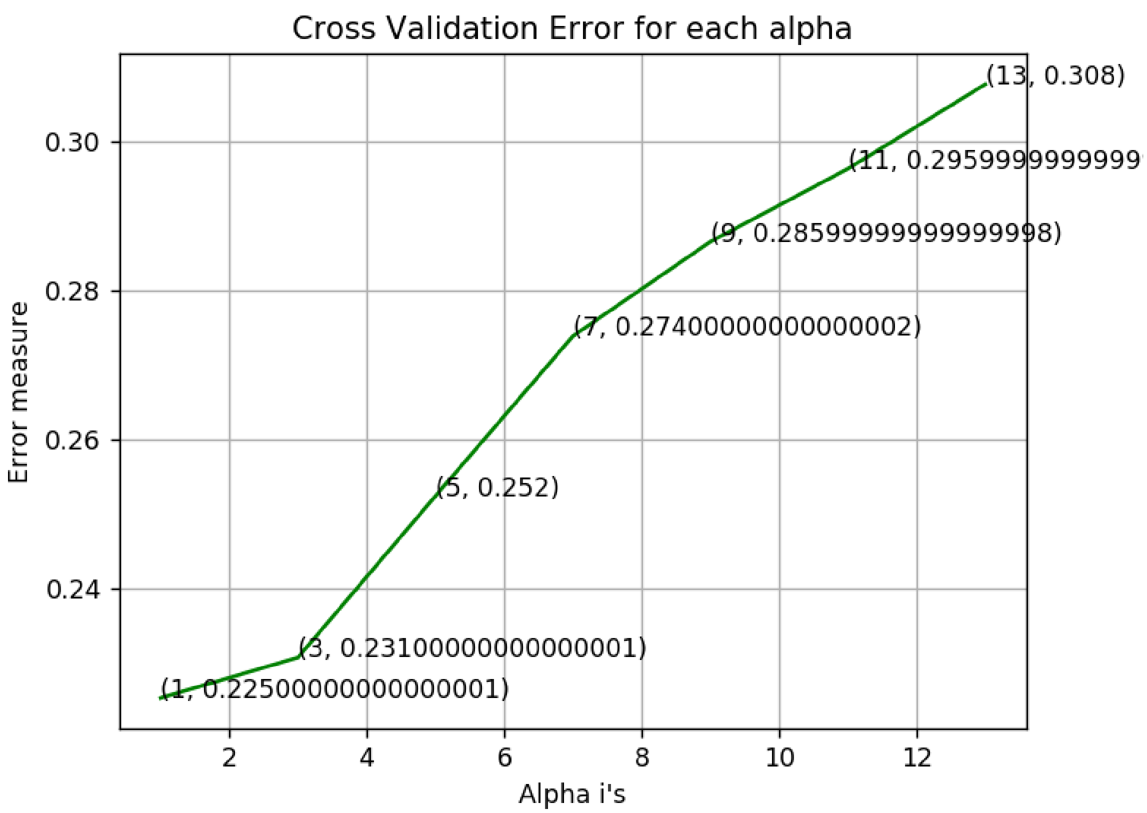
4.1.2. K Nearest Neighbour Classification

In [68]:

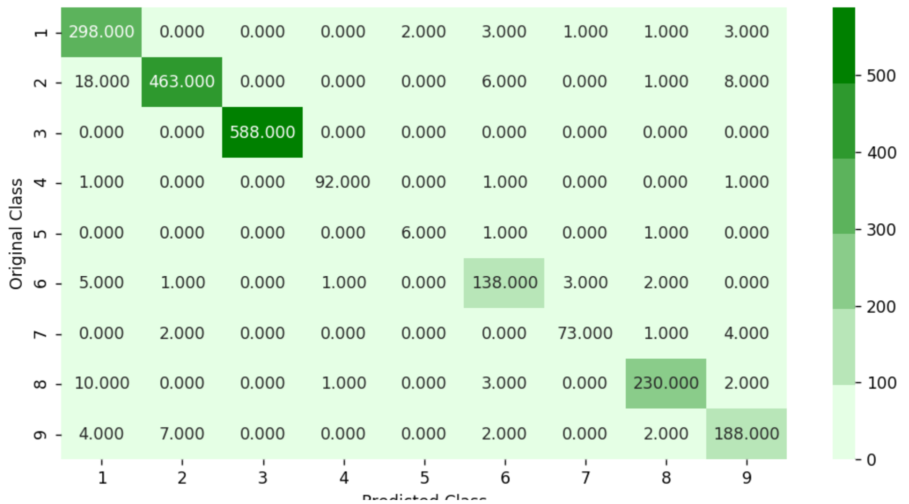
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

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

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

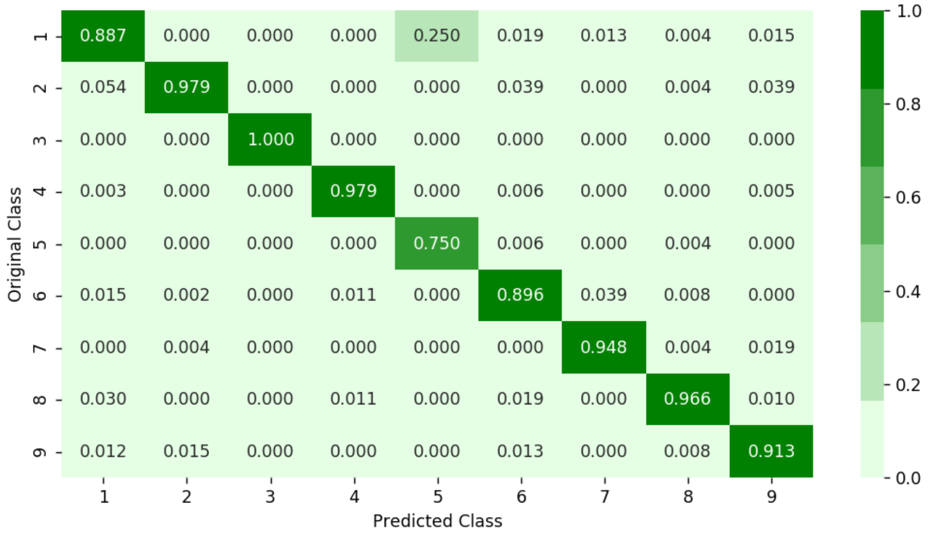
```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154

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



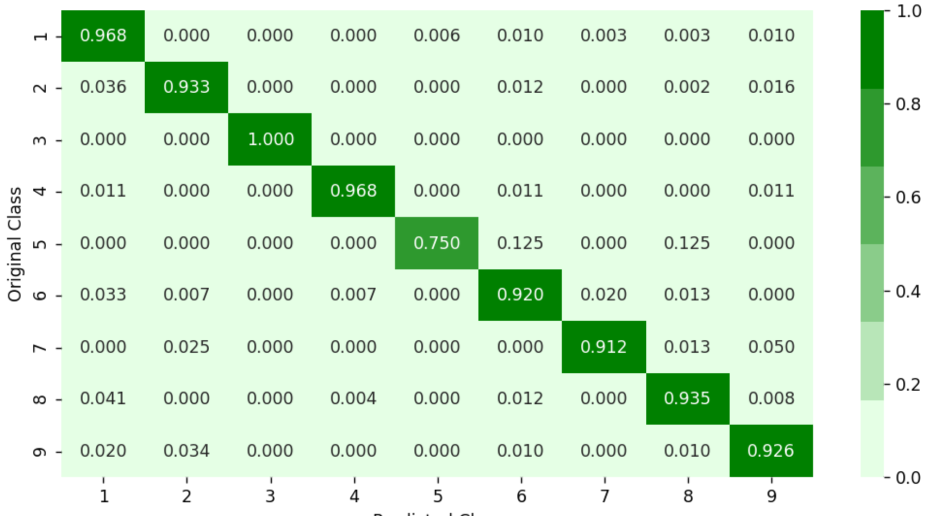
```
For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.225386237304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
--------------------------------------------------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

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



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

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



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

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



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

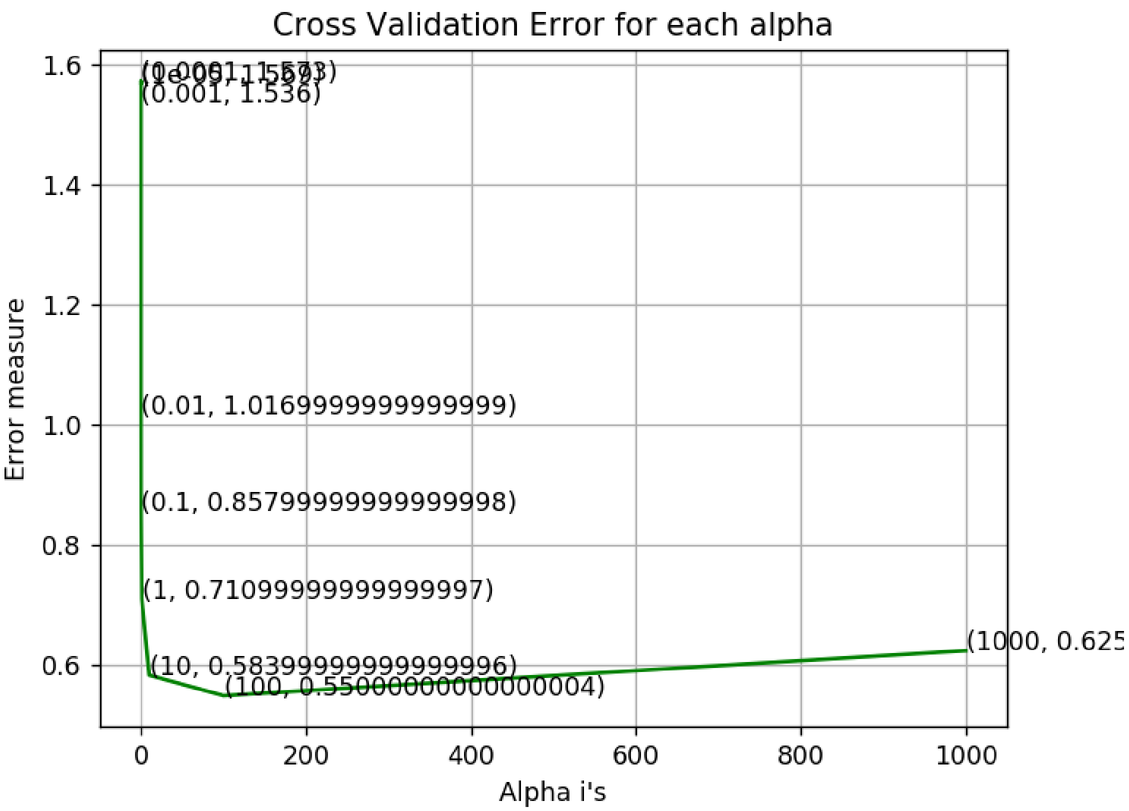
4.1.3. Logistic Regression

In [71]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_i
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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.class

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='ba
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
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.class
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.c
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

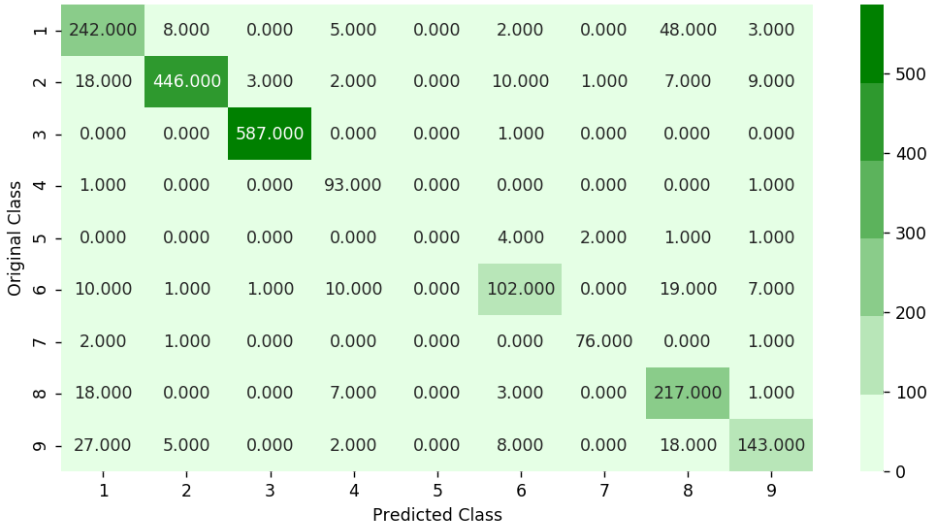
```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =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

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

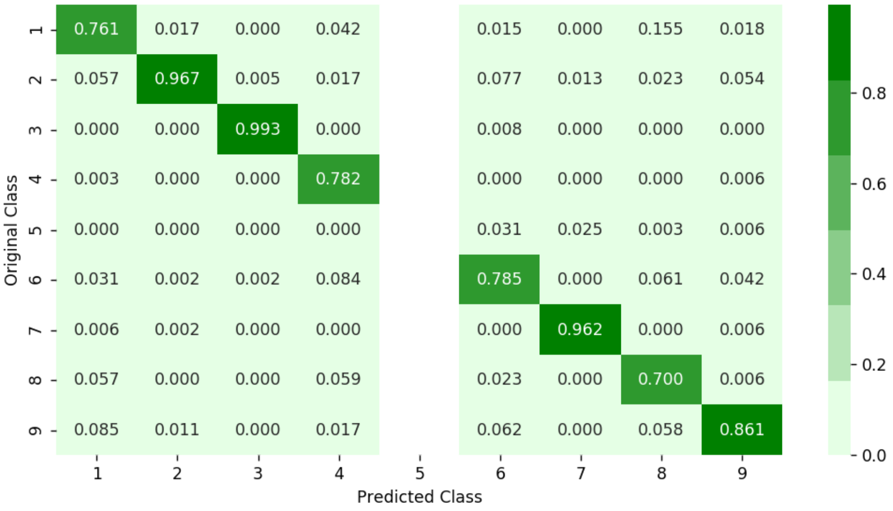


```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
-------------------------------------------------- Confusion matrix ----------
----------------------------------------
```

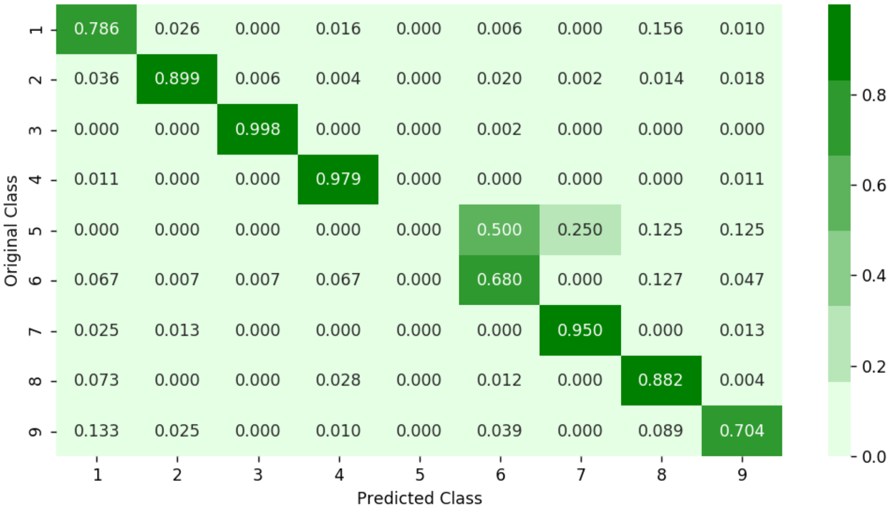
```
<IPython.core.display.Javascript object>
```



```
-------------------------------------------------- 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>
```



	1	2	3	4	5	6	7	8	9
1	0.786	0.026	0.000	0.016	0.000	0.006	0.000	0.156	0.010
2	0.036	0.899	0.006	0.004	0.000	0.020	0.002	0.014	0.018
3	0.000	0.000	0.998	0.000	0.000	0.002	0.000	0.000	0.000
4	0.011	0.000	0.000	0.979	0.000	0.000	0.000	0.000	0.011
5	0.000	0.000	0.000	0.000	0.000	0.500	0.250	0.125	0.125
6	0.067	0.007	0.007	0.067	0.000	0.680	0.000	0.127	0.047
7	0.025	0.013	0.000	0.000	0.000	0.000	0.950	0.000	0.013
8	0.073	0.000	0.000	0.028	0.000	0.012	0.000	0.882	0.004
9	0.133	0.025	0.000	0.010	0.000	0.039	0.000	0.089	0.704

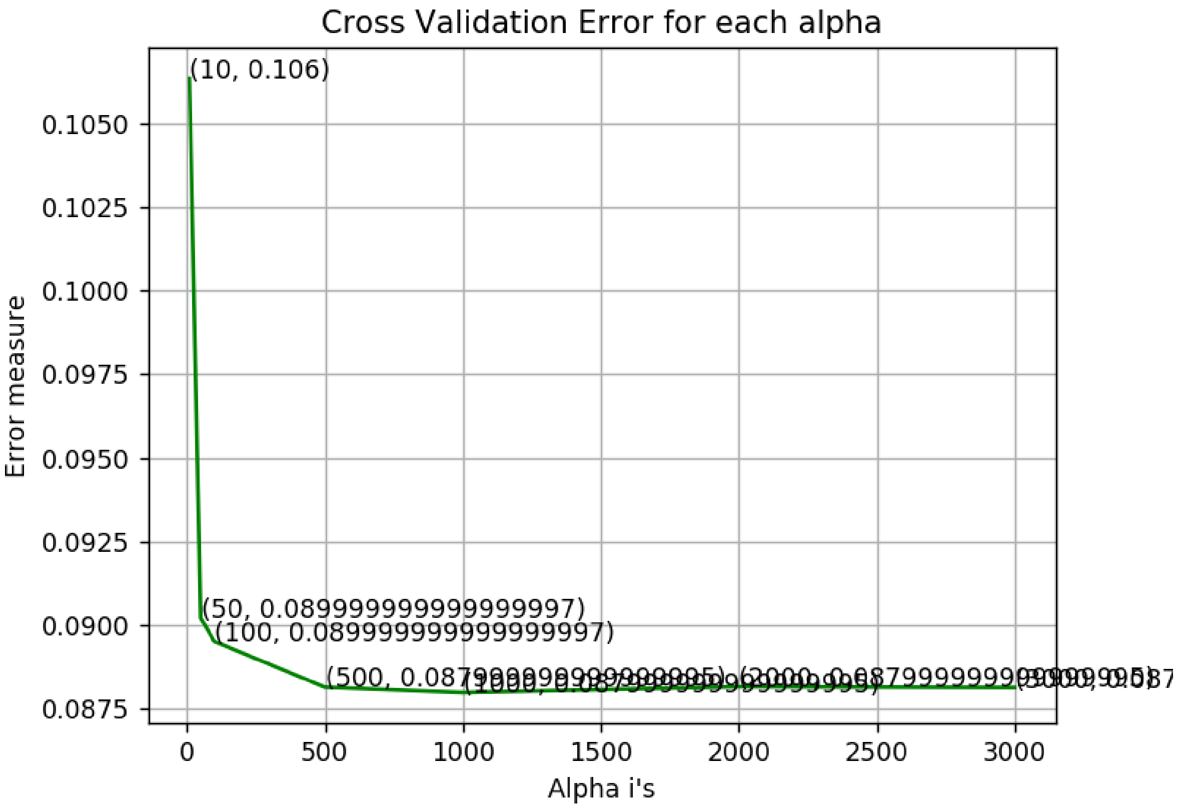
```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.4. Random Forest Classifier

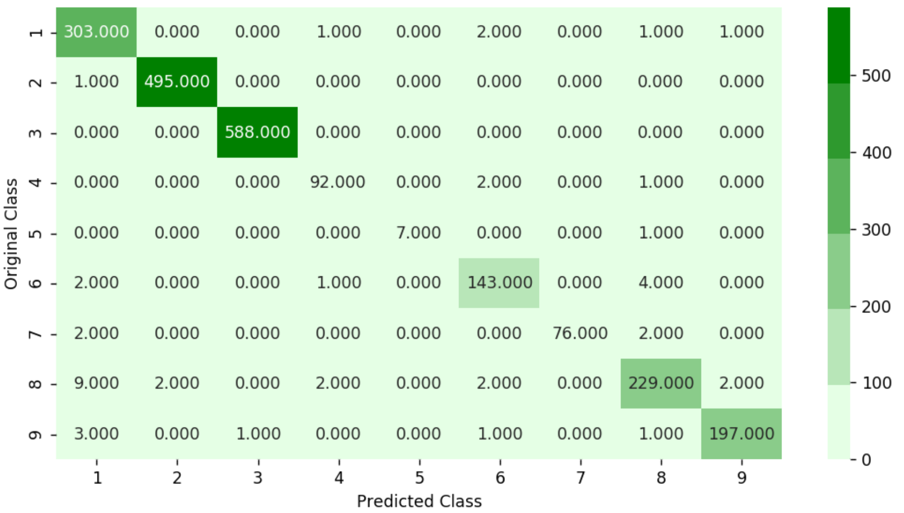
In [72]:

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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

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

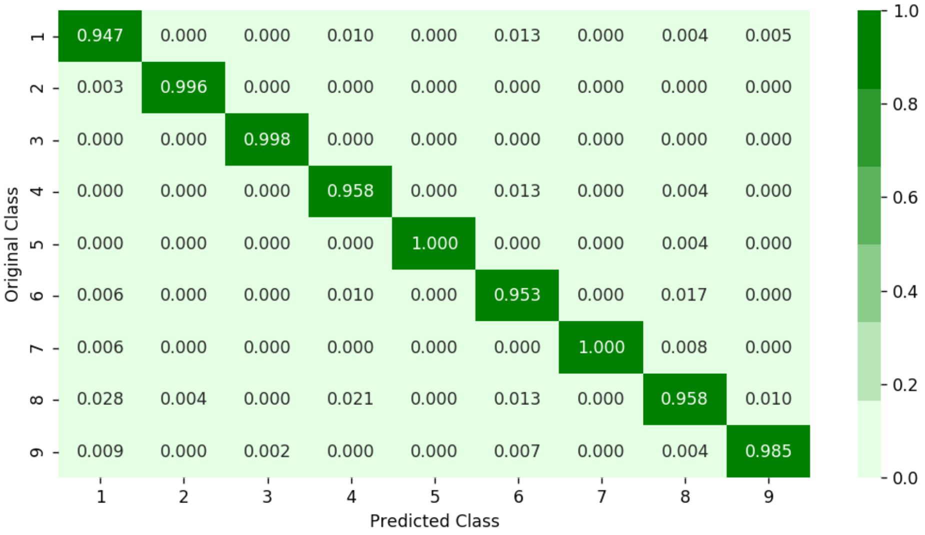


```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.087984952
4621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
-------------------------------------------------- Confusion matrix ----------
----------------------------------------

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



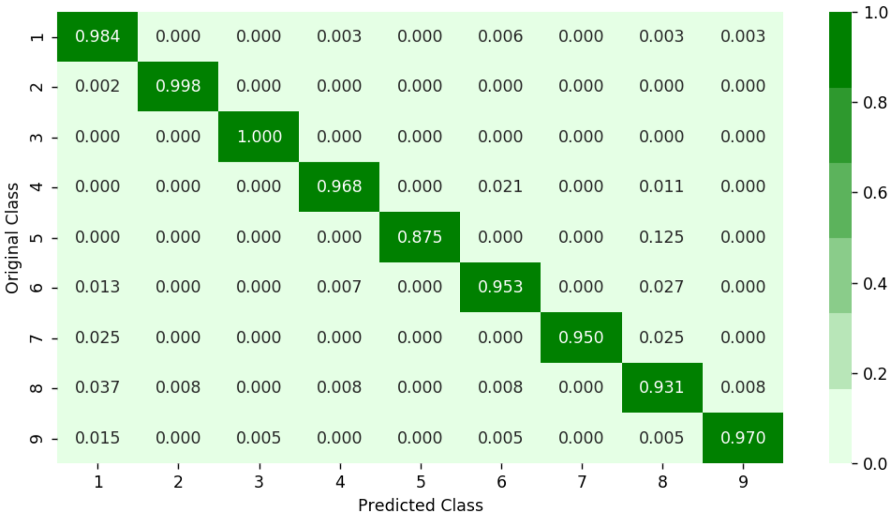
	1	2	3	4	5	6	7	8	9
1	303.000	0.000	0.000	1.000	0.000	2.000	0.000	1.000	1.000
2	1.000	495.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	92.000	0.000	2.000	0.000	1.000	0.000
5	0.000	0.000	0.000	0.000	7.000	0.000	0.000	1.000	0.000
6	2.000	0.000	0.000	1.000	0.000	143.000	0.000	4.000	0.000
7	2.000	0.000	0.000	0.000	0.000	0.000	76.000	2.000	0.000
8	9.000	2.000	0.000	2.000	0.000	2.000	0.000	229.000	2.000
9	3.000	0.000	1.000	0.000	0.000	1.000	0.000	1.000	197.000

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

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



	1	2	3	4	5	6	7	8	9
1	0.947	0.000	0.000	0.010	0.000	0.013	0.000	0.004	0.005
2	0.003	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.998	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.958	0.000	0.013	0.000	0.004	0.000
5	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.004	0.000
6	0.006	0.000	0.000	0.010	0.000	0.953	0.000	0.017	0.000
7	0.006	0.000	0.000	0.000	0.000	0.000	1.000	0.008	0.000
8	0.028	0.004	0.000	0.021	0.000	0.013	0.000	0.958	0.010
9	0.009	0.000	0.002	0.000	0.000	0.007	0.000	0.004	0.985

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

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



	1	2	3	4	5	6	7	8	9
1	0.984	0.000	0.000	0.003	0.000	0.006	0.000	0.003	0.003
2	0.002	0.998	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.968	0.000	0.021	0.000	0.011	0.000
5	0.000	0.000	0.000	0.000	0.875	0.000	0.000	0.125	0.000
6	0.013	0.000	0.000	0.007	0.000	0.953	0.000	0.027	0.000
7	0.025	0.000	0.000	0.000	0.000	0.000	0.950	0.025	0.000
8	0.037	0.008	0.000	0.008	0.000	0.008	0.000	0.931	0.008
9	0.015	0.000	0.005	0.000	0.000	0.005	0.000	0.005	0.970

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

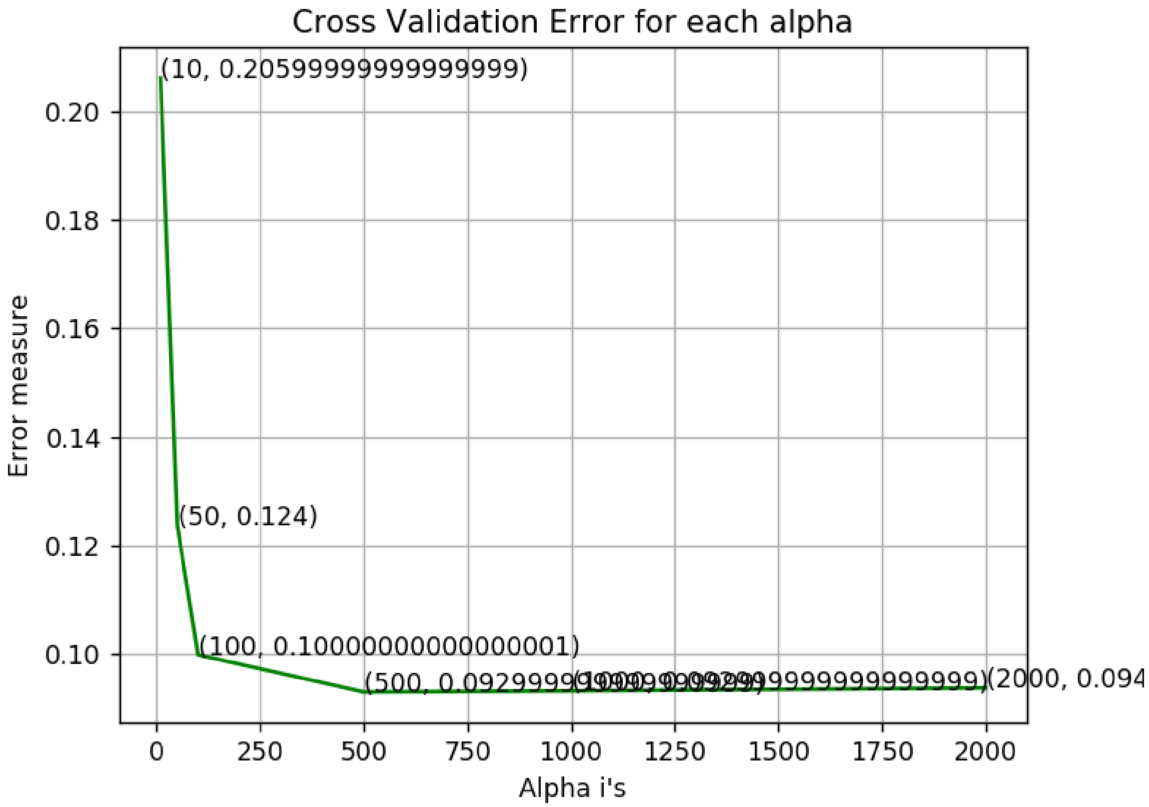
4.1.5. XgBoost Classification

In [74]:

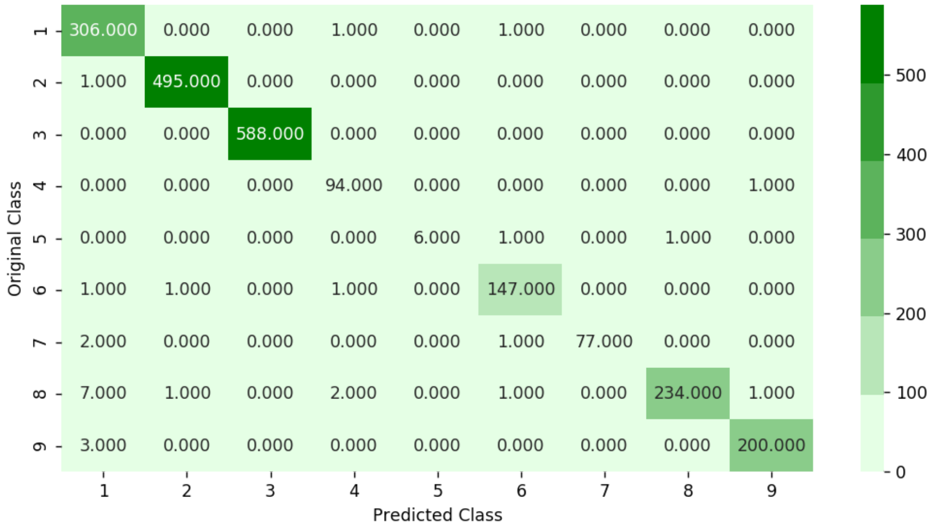
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

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_,

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

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

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

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

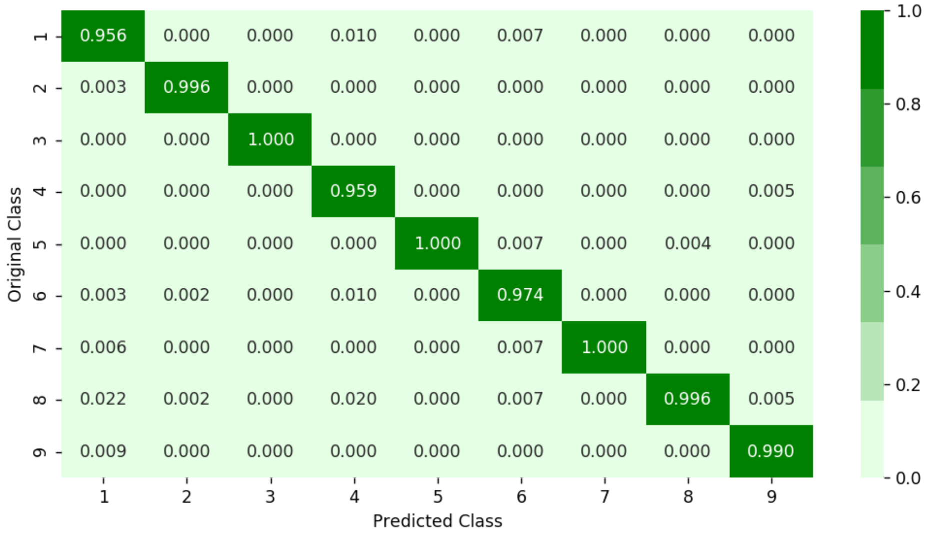


```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.0931035681
289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
-------------------------------------------------- Confusion matrix ----------
----------------------------------------

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



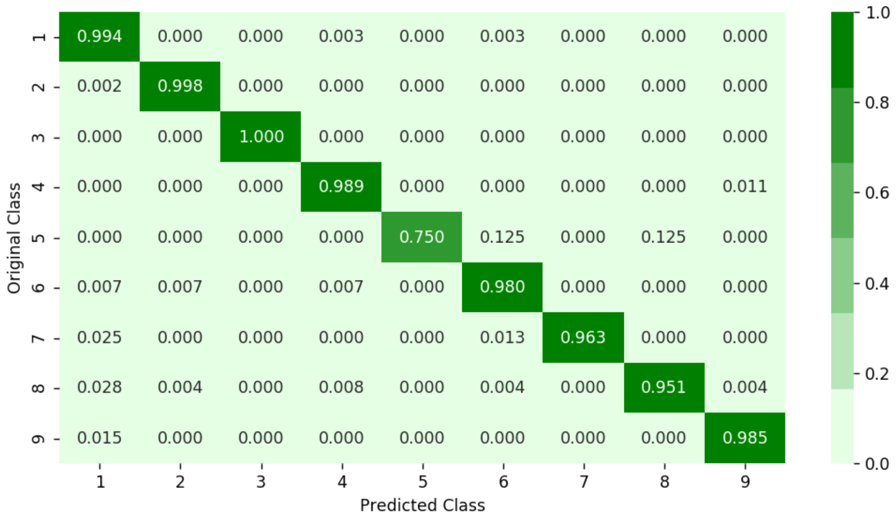
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	306.000	0.000	0.000	1.000	0.000	1.000	0.000	0.000	0.000
2	1.000	495.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	94.000	0.000	0.000	0.000	0.000	1.000
5	0.000	0.000	0.000	0.000	6.000	1.000	0.000	1.000	0.000
6	1.000	1.000	0.000	1.000	0.000	147.000	0.000	0.000	0.000
7	2.000	0.000	0.000	0.000	0.000	1.000	77.000	0.000	0.000
8	7.000	1.000	0.000	2.000	0.000	1.000	0.000	234.000	1.000
9	3.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	200.000

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

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.956	0.000	0.000	0.010	0.000	0.007	0.000	0.000	0.000
2	0.003	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.959	0.000	0.000	0.000	0.000	0.005
5	0.000	0.000	0.000	0.000	1.000	0.007	0.000	0.004	0.000
6	0.003	0.002	0.000	0.010	0.000	0.974	0.000	0.000	0.000
7	0.006	0.000	0.000	0.000	0.000	0.007	1.000	0.000	0.000
8	0.022	0.002	0.000	0.020	0.000	0.007	0.000	0.996	0.005
9	0.009	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.990

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

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



	1	2	3	4	5	6	7	8	9
1	0.994	0.000	0.000	0.003	0.000	0.003	0.000	0.000	0.000
2	0.002	0.998	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.989	0.000	0.000	0.000	0.000	0.011
5	0.000	0.000	0.000	0.000	0.750	0.125	0.000	0.125	0.000
6	0.007	0.007	0.000	0.007	0.000	0.980	0.000	0.000	0.000
7	0.025	0.000	0.000	0.000	0.000	0.013	0.963	0.000	0.000
8	0.028	0.004	0.000	0.008	0.000	0.004	0.000	0.951	0.004
9	0.015	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.985

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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [75]:

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning
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_jo
random_cfl1.fit(X_train,y_train)
```

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

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

Out[75]:

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

In [76]:

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

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

In [80]:

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

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

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

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

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

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

4.2 Modeling with .asm files

```
There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In par
allel 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 [ ]:

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

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

In [ ]:

```

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

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

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


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

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

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
    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 f
            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
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in 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
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
    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 f
            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 [137]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain a
# 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[137]:

```
                     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 [138]:
#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=356157170
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file n
    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())
```

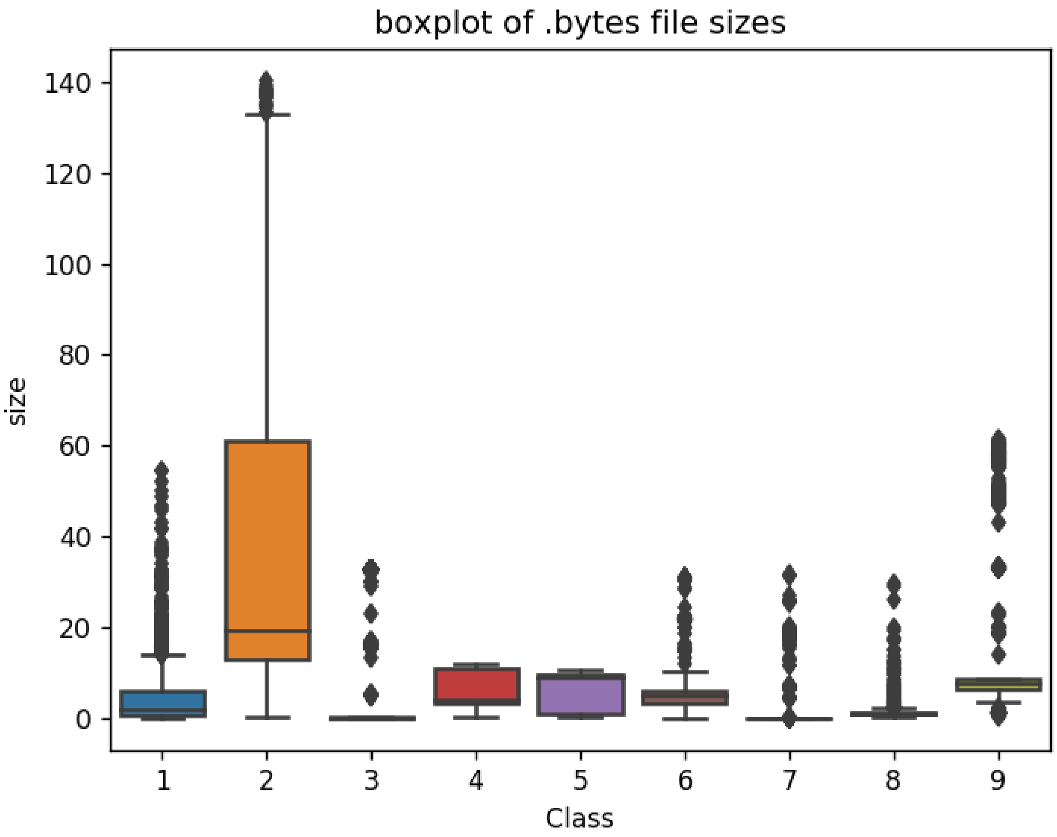
```
   Class                    ID       size
0      9  01azqd4InC7m9JpocGv5  56.229886
1      2  01IsoiSMh5gxyDYTl4CB  13.999378
2      9  01jsnpXSAlgw6aPeDxrU   8.507785
3      1  01kcPWA9K2BOxQeS5Rju   0.078190
4      8  01SuzwMJEIXsK7A8dQbl   0.996723
```

4.2.1.2 Distribution of .asm file sizes

In [139]:

```python
#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 [140]:

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

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

Out[140]:

	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 × 54 columns

In [145]:

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

Out[145]:

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

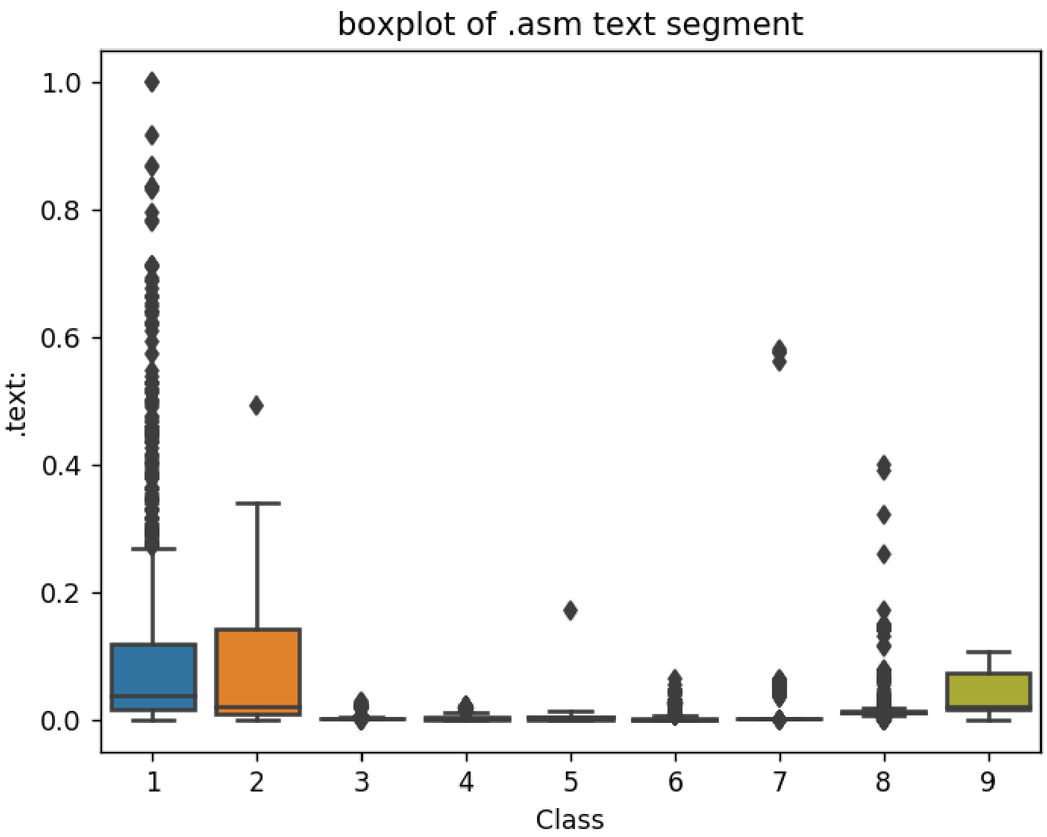
5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [146]:

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

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

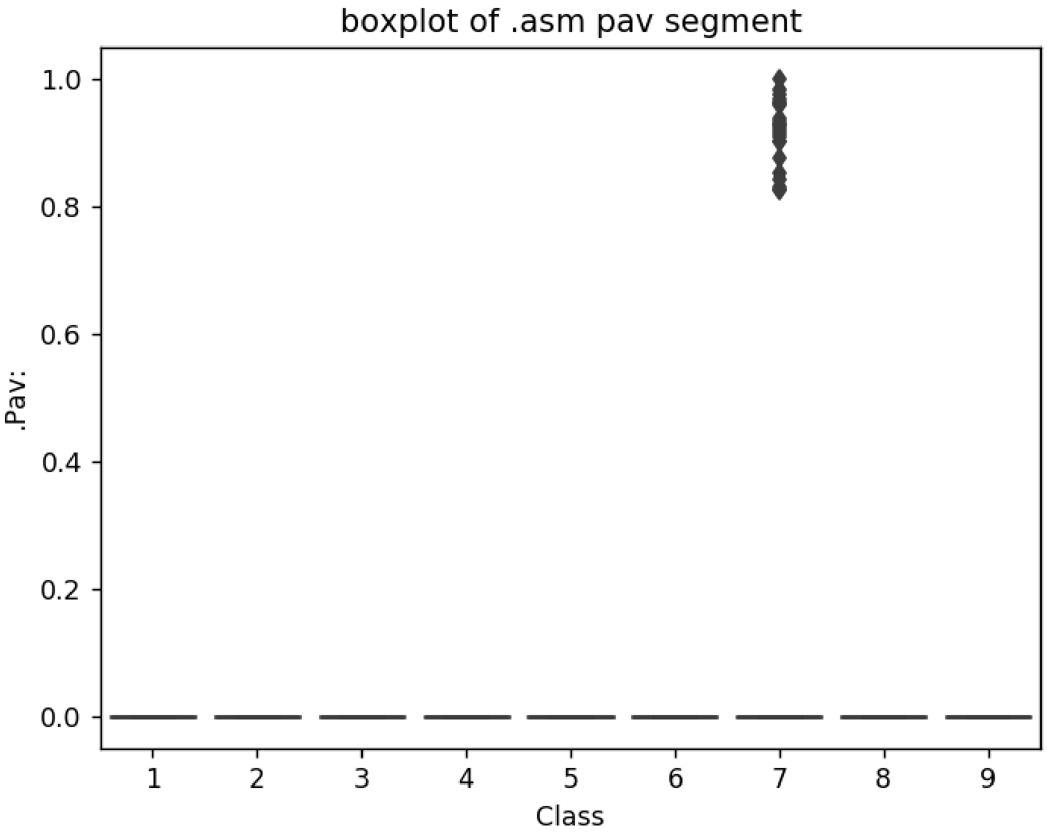


```
The plot is between Text and class
Class 1,2 and 9 can be easly separated
```

In [115]:

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

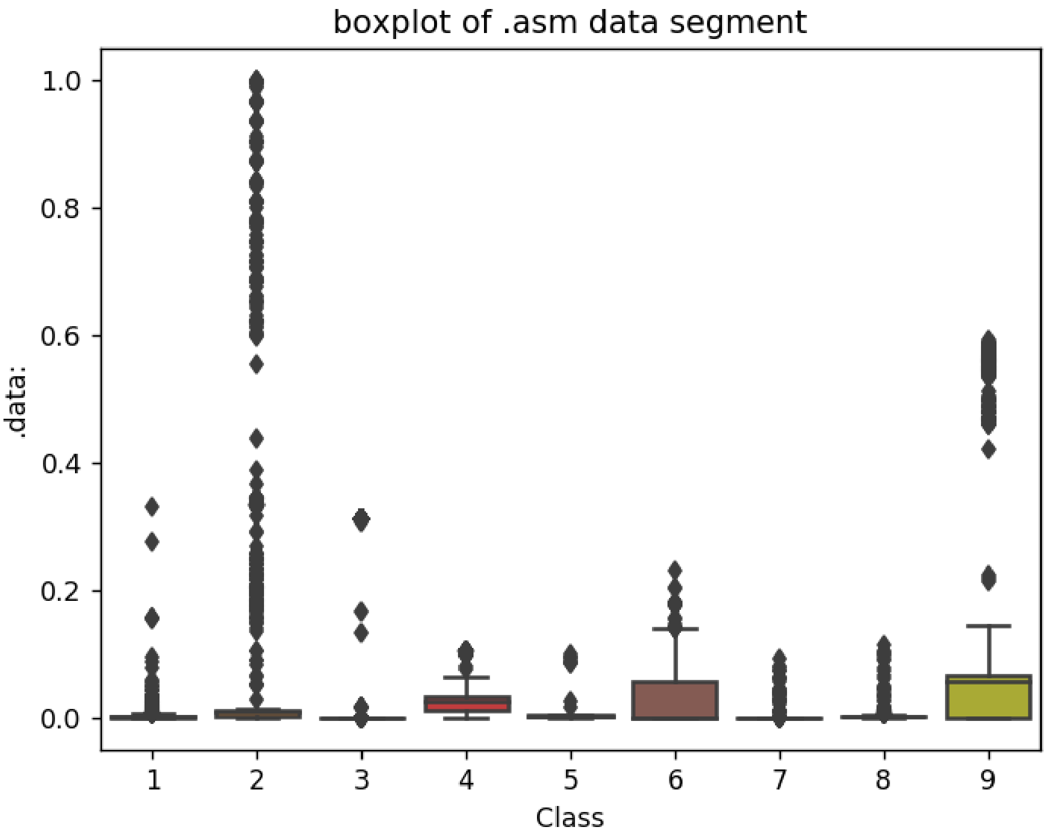
```
<IPython.core.display.Javascript object>
```



In [116]:

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

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

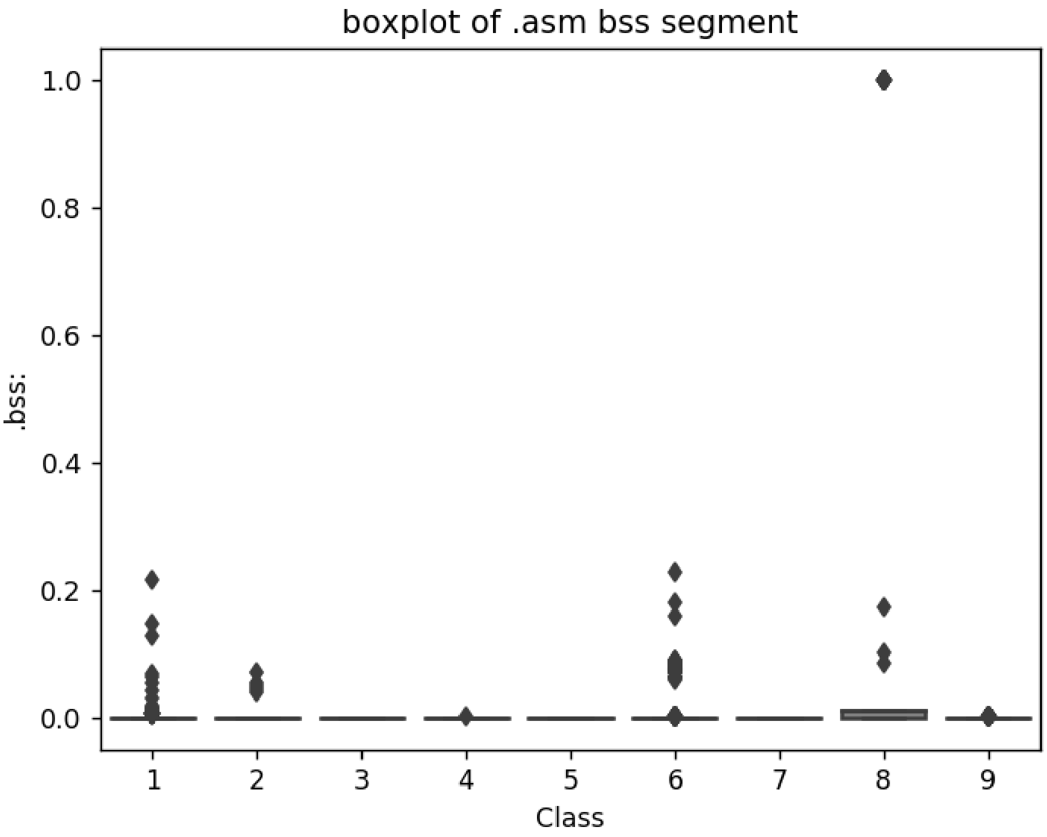


```
The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points
```

In [117]:

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

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

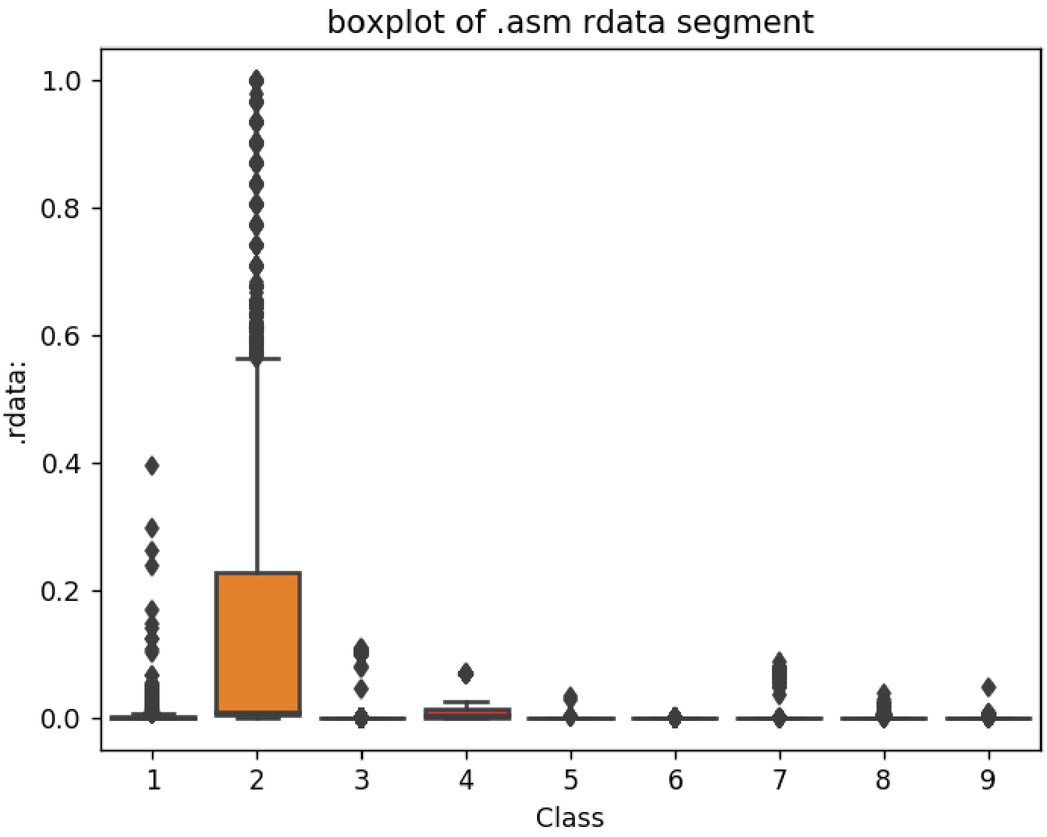


plot between bss segment and class label
very less number of files are having bss segment

In [118]:

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

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

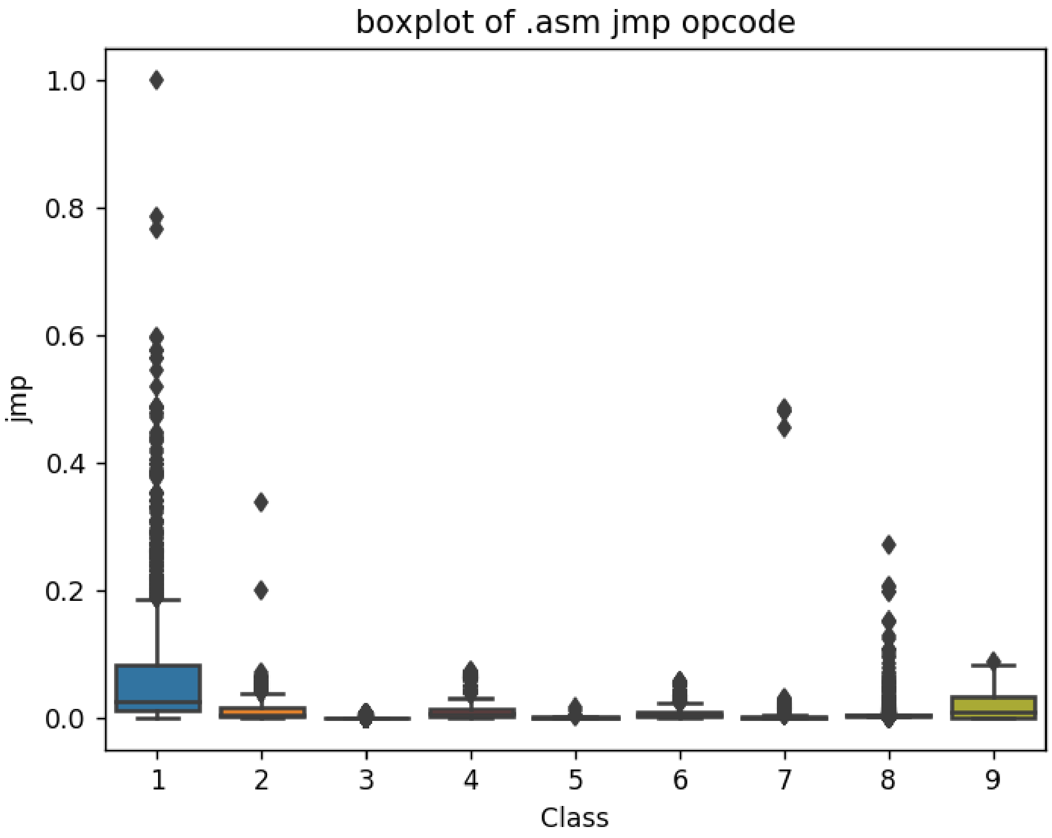


Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [119]:

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

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

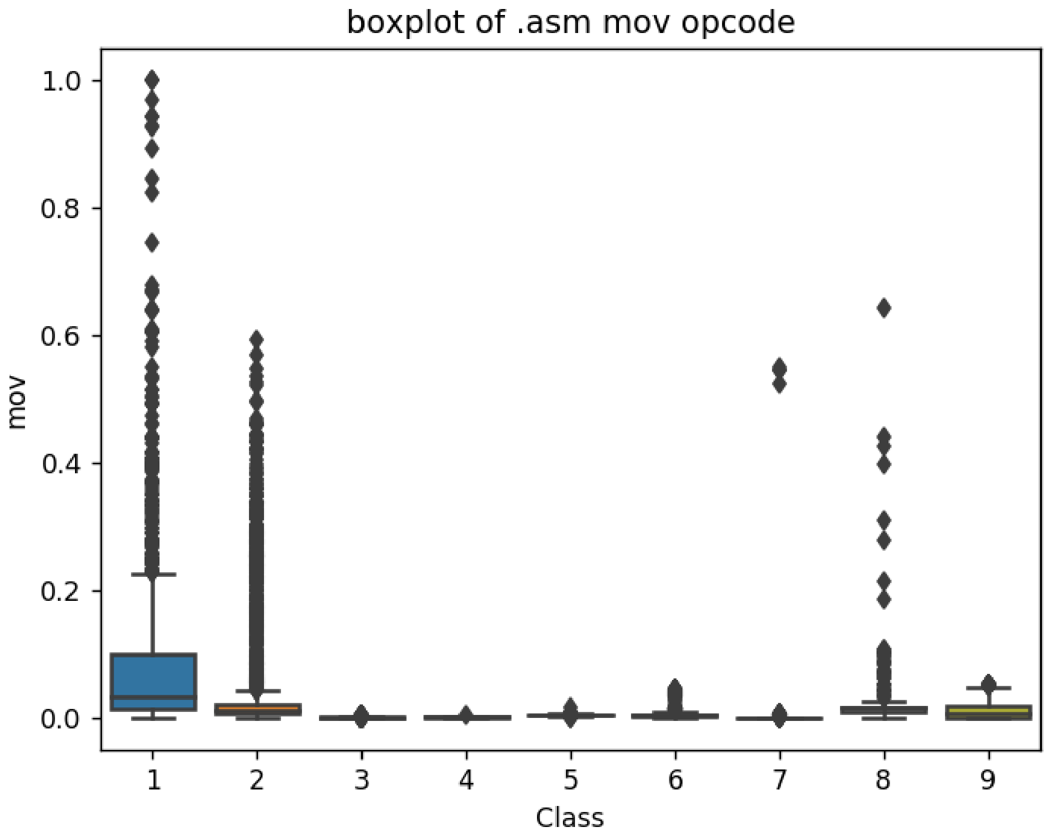


plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [120]:

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

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

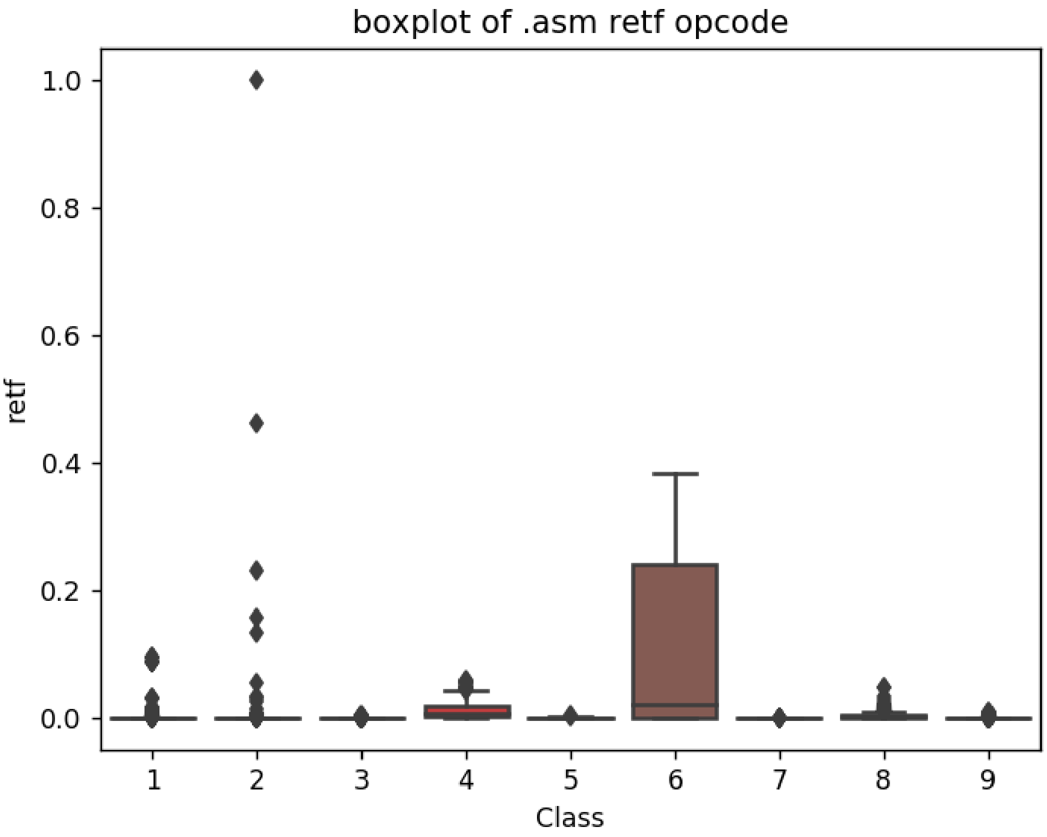


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [121]:

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

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

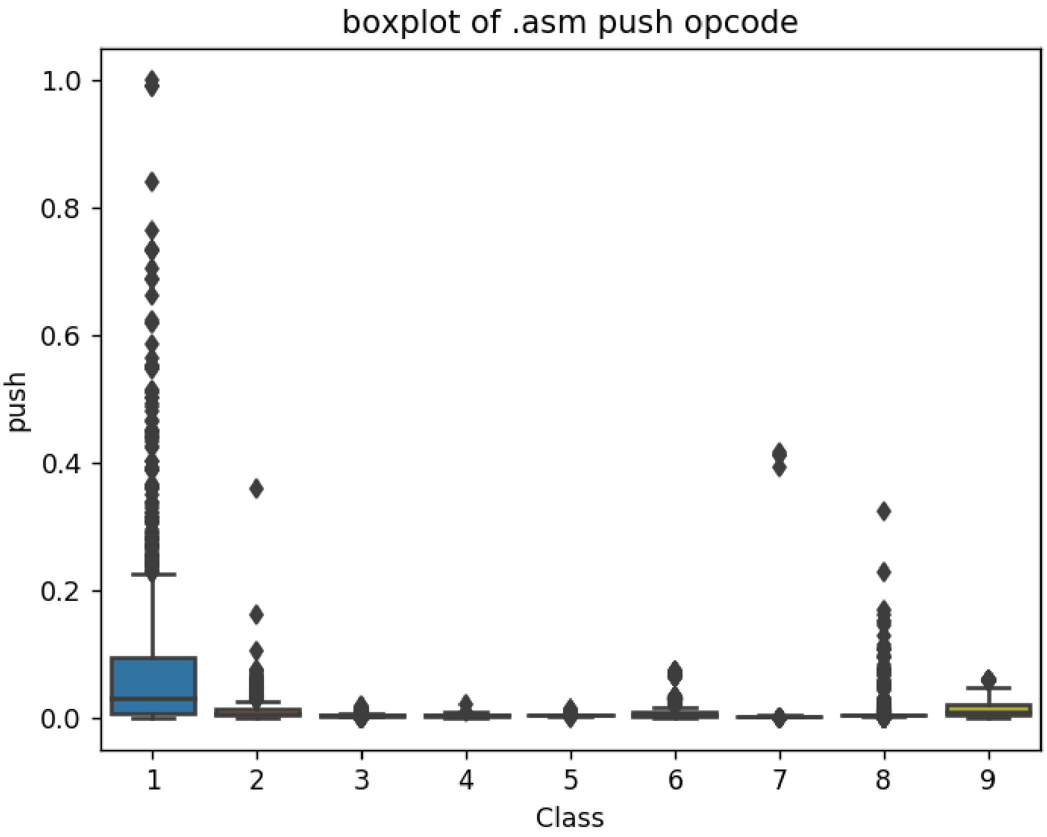


plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [122]:

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

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



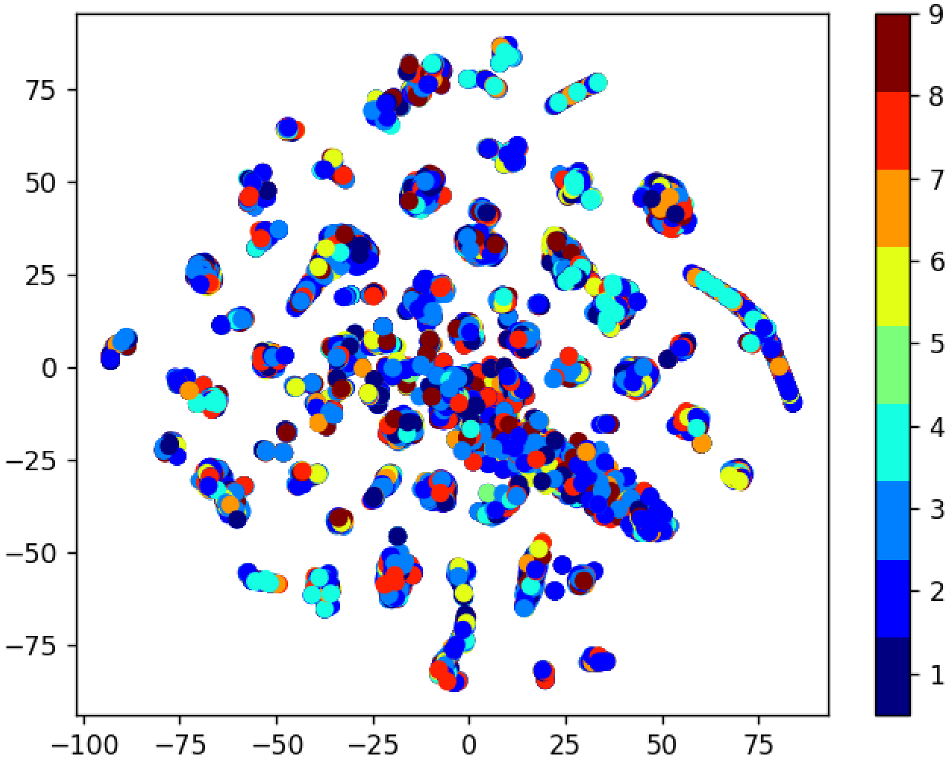
```
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000
```

4.2.2 Multivariate Analysis on .asm file features

In [129]:

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

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

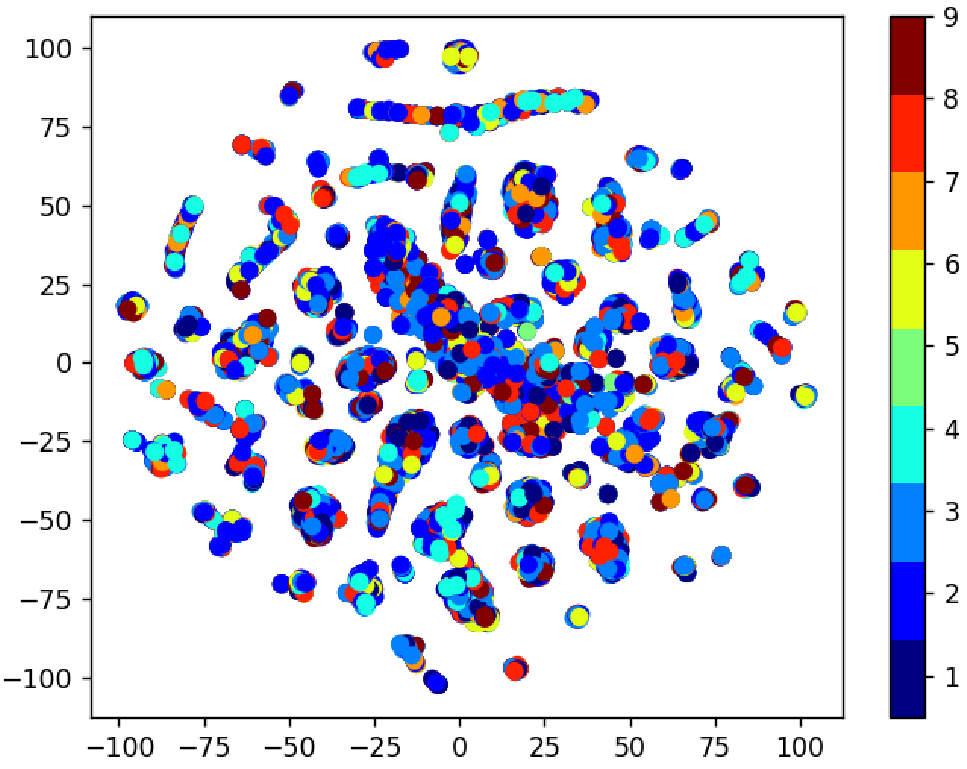
```
<IPython.core.display.Javascript object>
```



In [147]:

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

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

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



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

4.2.3 Conclusion on EDA

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

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

4.3 Train and test split

In [149]:

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

In [150]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm,
In [153]: print( X_cv_asm.isnull().all())
```

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

4.4. Machine Learning models on features of .asm files

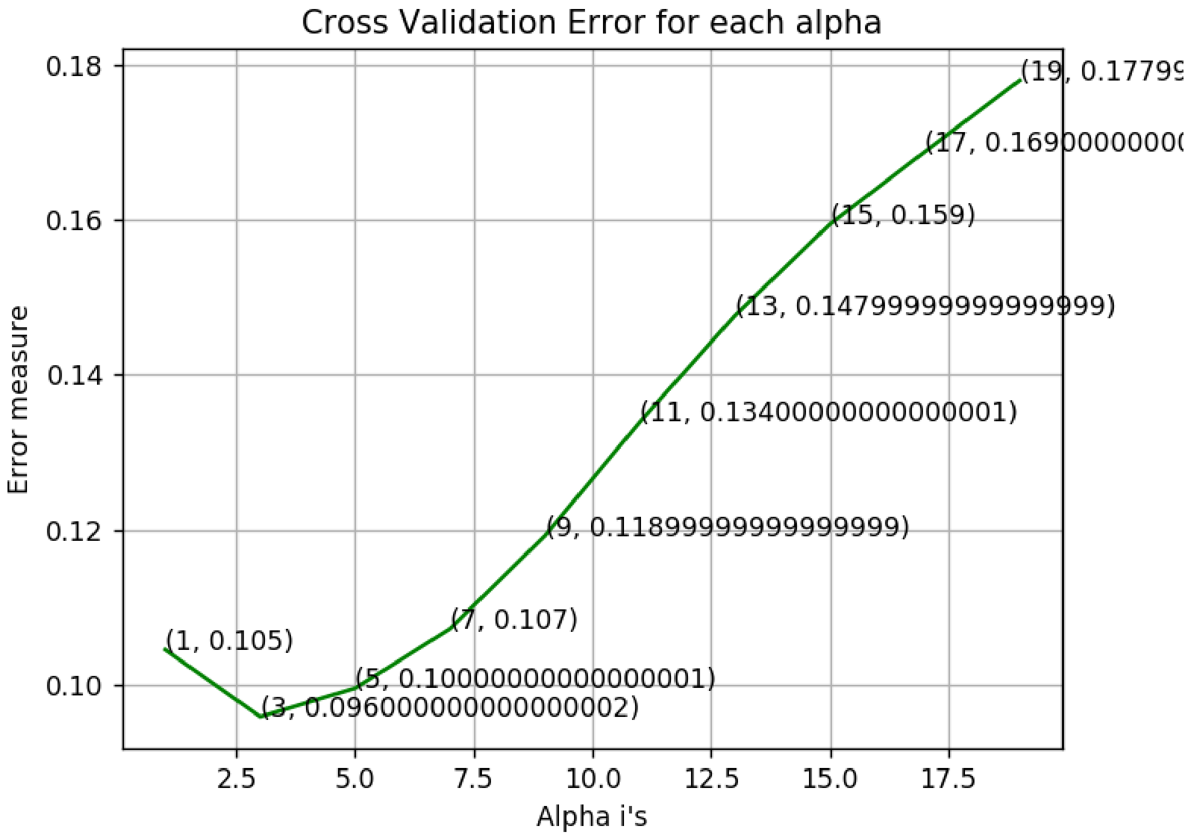
4.4.1 K-Nearest Neigbors

In [159]:

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

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


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

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

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)
```

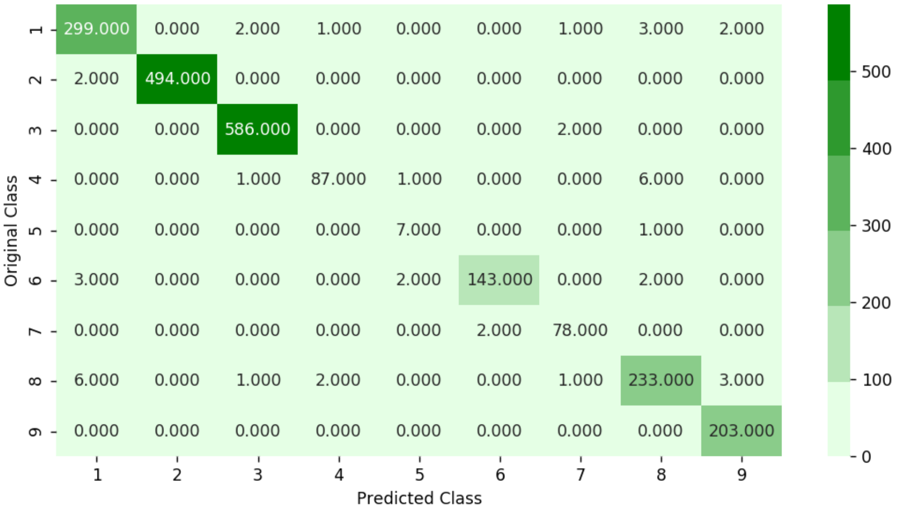
```python
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


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

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

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



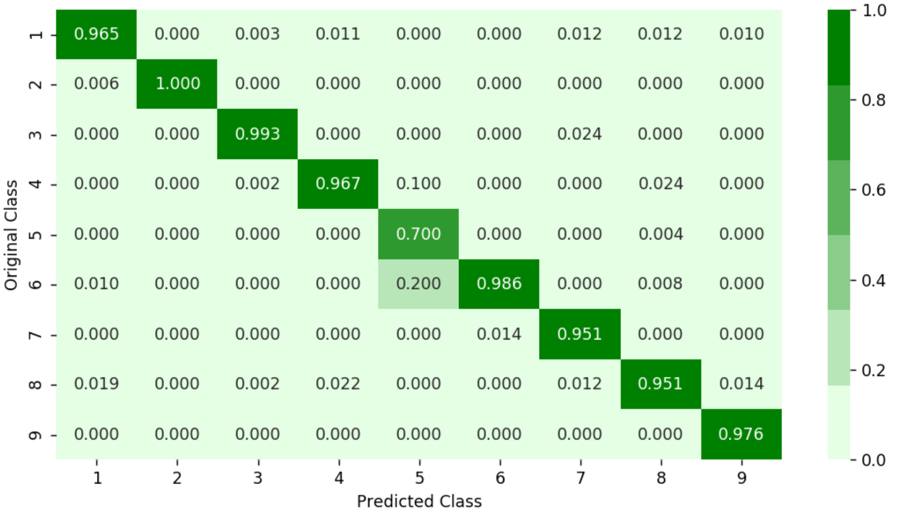
```
log loss for train data 0.0476773462198
log loss for cv data 0.0958800580948
log loss for test data 0.0894810720832
```

```
Number of misclassified points  2.02391904324
-------------------------------------------------- Confusion matrix --------
------------------------------------------

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



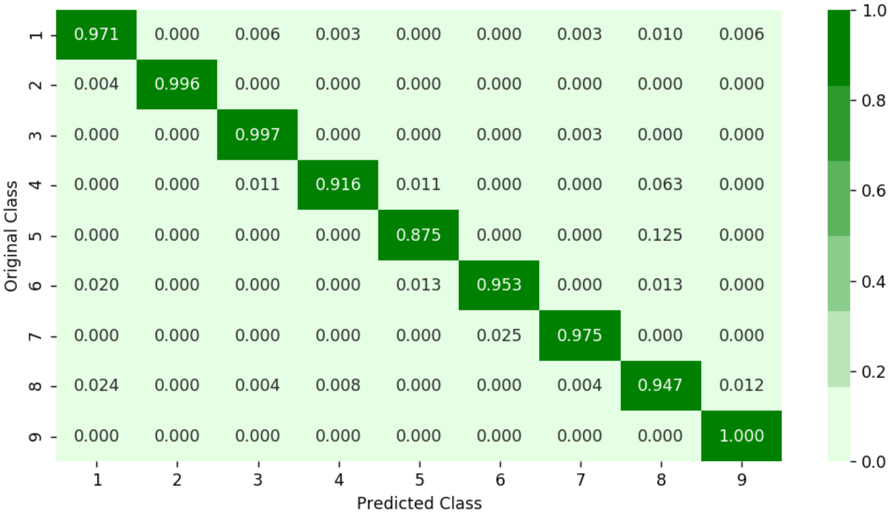
	1	2	3	4	5	6	7	8	9
1	299.000	0.000	2.000	1.000	0.000	0.000	1.000	3.000	2.000
2	2.000	494.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	586.000	0.000	0.000	0.000	2.000	0.000	0.000
4	0.000	0.000	1.000	87.000	1.000	0.000	0.000	6.000	0.000
5	0.000	0.000	0.000	0.000	7.000	0.000	0.000	1.000	0.000
6	3.000	0.000	0.000	0.000	2.000	143.000	0.000	2.000	0.000
7	0.000	0.000	0.000	0.000	0.000	2.000	78.000	0.000	0.000
8	6.000	0.000	1.000	2.000	0.000	0.000	1.000	233.000	3.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	203.000

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

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



	1	2	3	4	5	6	7	8	9
1	0.965	0.000	0.003	0.011	0.000	0.000	0.012	0.012	0.010
2	0.006	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.993	0.000	0.000	0.000	0.024	0.000	0.000
4	0.000	0.000	0.002	0.967	0.100	0.000	0.000	0.024	0.000
5	0.000	0.000	0.000	0.000	0.700	0.000	0.000	0.004	0.000
6	0.010	0.000	0.000	0.000	0.200	0.986	0.000	0.008	0.000
7	0.000	0.000	0.000	0.000	0.000	0.014	0.951	0.000	0.000
8	0.019	0.000	0.002	0.022	0.000	0.000	0.012	0.951	0.014
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.976

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

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



	1	2	3	4	5	6	7	8	9
1	0.971	0.000	0.006	0.003	0.000	0.000	0.003	0.010	0.006
2	0.004	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.997	0.000	0.000	0.000	0.003	0.000	0.000
4	0.000	0.000	0.011	0.916	0.011	0.000	0.000	0.063	0.000
5	0.000	0.000	0.000	0.000	0.875	0.000	0.000	0.125	0.000
6	0.020	0.000	0.000	0.000	0.013	0.953	0.000	0.013	0.000
7	0.000	0.000	0.000	0.000	0.000	0.025	0.975	0.000	0.000
8	0.024	0.000	0.004	0.008	0.000	0.000	0.004	0.947	0.012
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

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

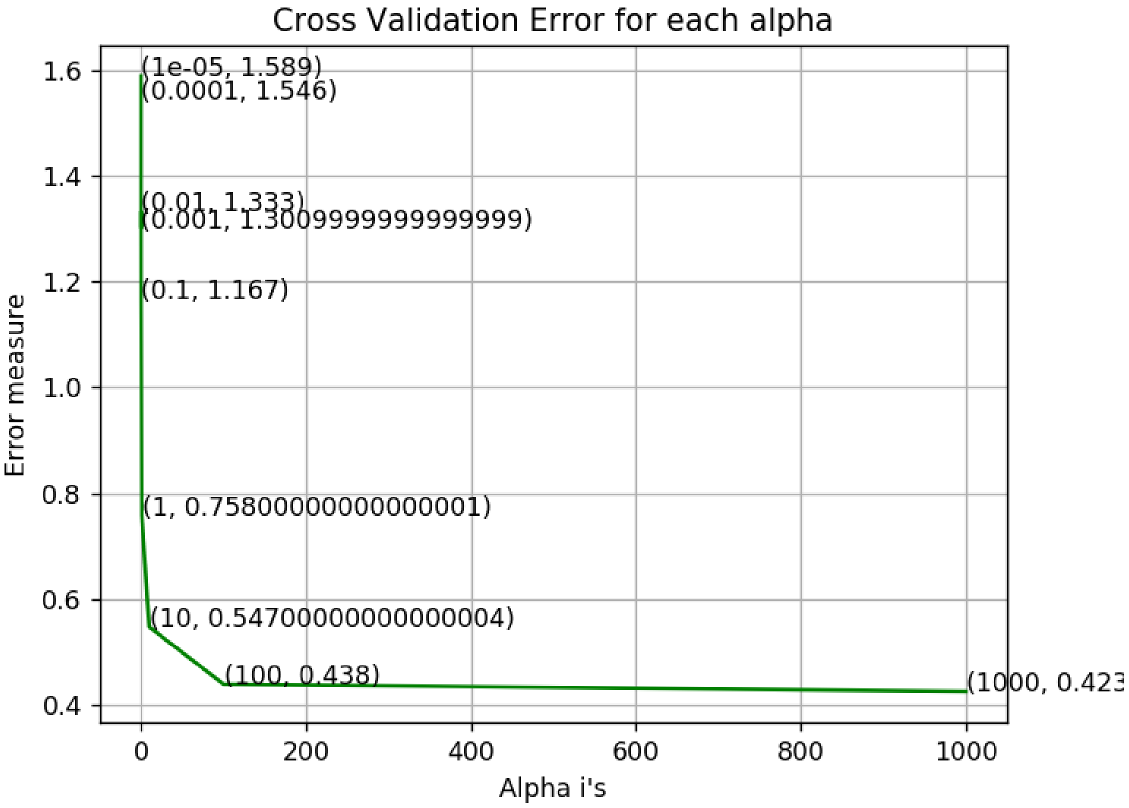
4.4.2 Logistic Regression

In [160]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/g
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

best_alpha = np.argmin(cv_log_error_array)

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

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='b
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=log
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logis
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

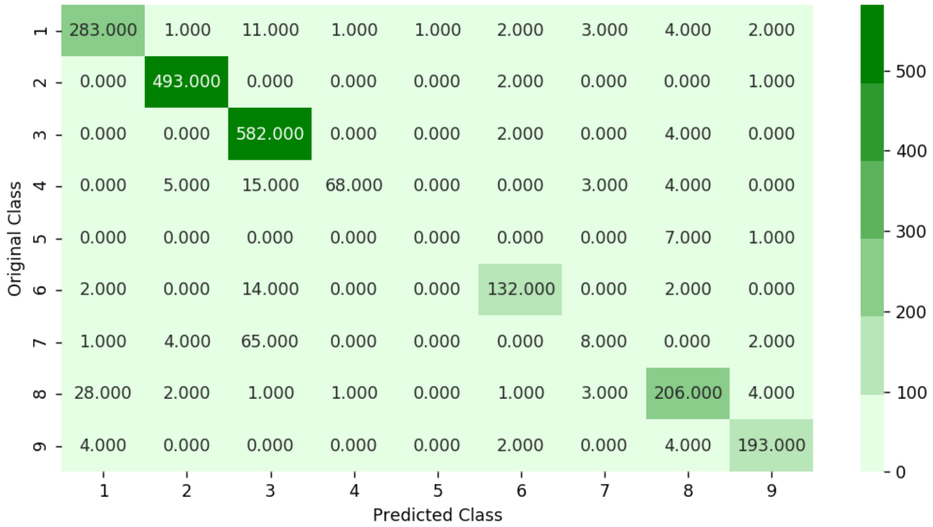
```
log_loss for c =  1e-05 is 1.58867274165
log_loss for c =  0.0001 is 1.54560797884
log_loss for c =  0.001 is 1.30137786807
log_loss for c =  0.01 is 1.33317456931
log_loss for c =  0.1 is 1.16705751378
log_loss for c =  1 is 0.757667807779
log_loss for c =  10 is 0.546533939819
log_loss for c =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

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



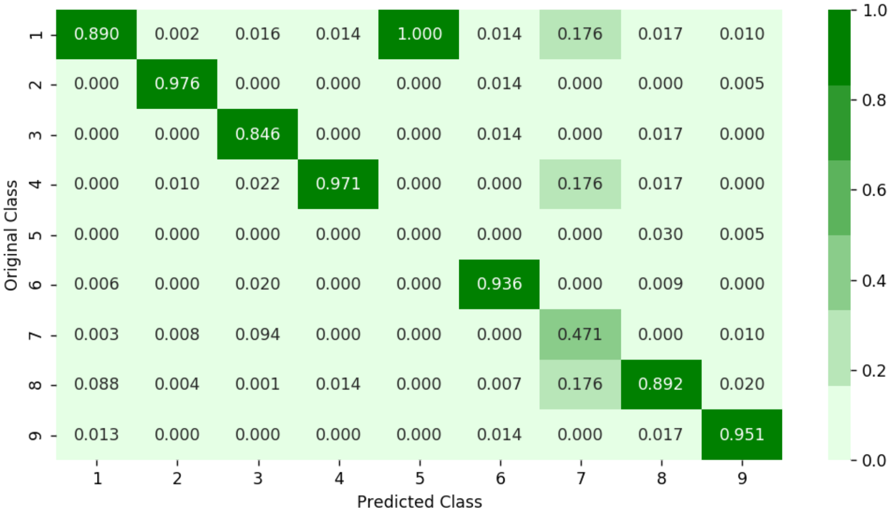
```
log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points  9.61361545538
-------------------------------------------------- Confusion matrix ----------
----------------------------------------
```

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



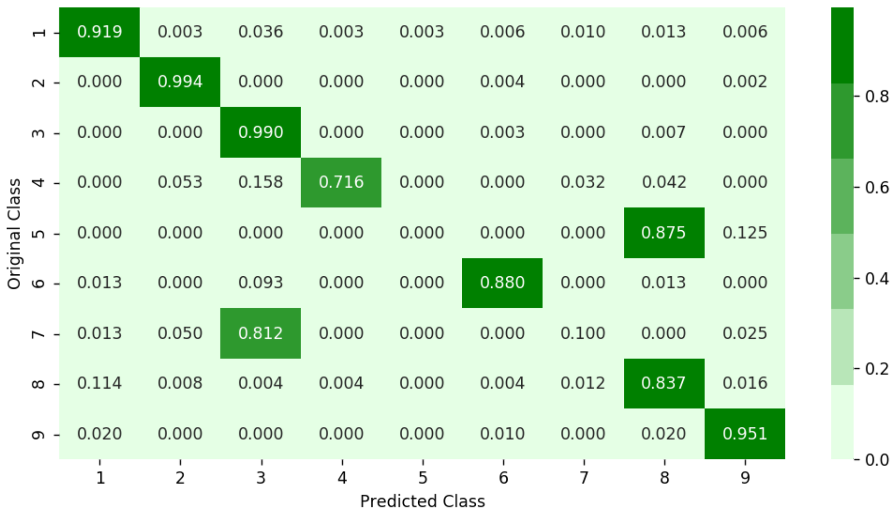
	1	2	3	4	5	6	7	8	9
1	283.000	1.000	11.000	1.000	1.000	2.000	3.000	4.000	2.000
2	0.000	493.000	0.000	0.000	0.000	2.000	0.000	0.000	1.000
3	0.000	0.000	582.000	0.000	0.000	2.000	0.000	4.000	0.000
4	0.000	5.000	15.000	68.000	0.000	0.000	3.000	4.000	0.000
5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	7.000	1.000
6	2.000	0.000	14.000	0.000	0.000	132.000	0.000	2.000	0.000
7	1.000	4.000	65.000	0.000	0.000	0.000	8.000	0.000	2.000
8	28.000	2.000	1.000	1.000	0.000	1.000	3.000	206.000	4.000
9	4.000	0.000	0.000	0.000	0.000	2.000	0.000	4.000	193.000

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

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



	1	2	3	4	5	6	7	8	9
1	0.890	0.002	0.016	0.014	1.000	0.014	0.176	0.017	0.010
2	0.000	0.976	0.000	0.000	0.000	0.014	0.000	0.000	0.005
3	0.000	0.000	0.846	0.000	0.000	0.014	0.000	0.017	0.000
4	0.000	0.010	0.022	0.971	0.000	0.000	0.176	0.017	0.000
5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.005
6	0.006	0.000	0.020	0.000	0.000	0.936	0.000	0.009	0.000
7	0.003	0.008	0.094	0.000	0.000	0.000	0.471	0.000	0.010
8	0.088	0.004	0.001	0.014	0.000	0.007	0.176	0.892	0.020
9	0.013	0.000	0.000	0.000	0.000	0.014	0.000	0.017	0.951

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

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



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

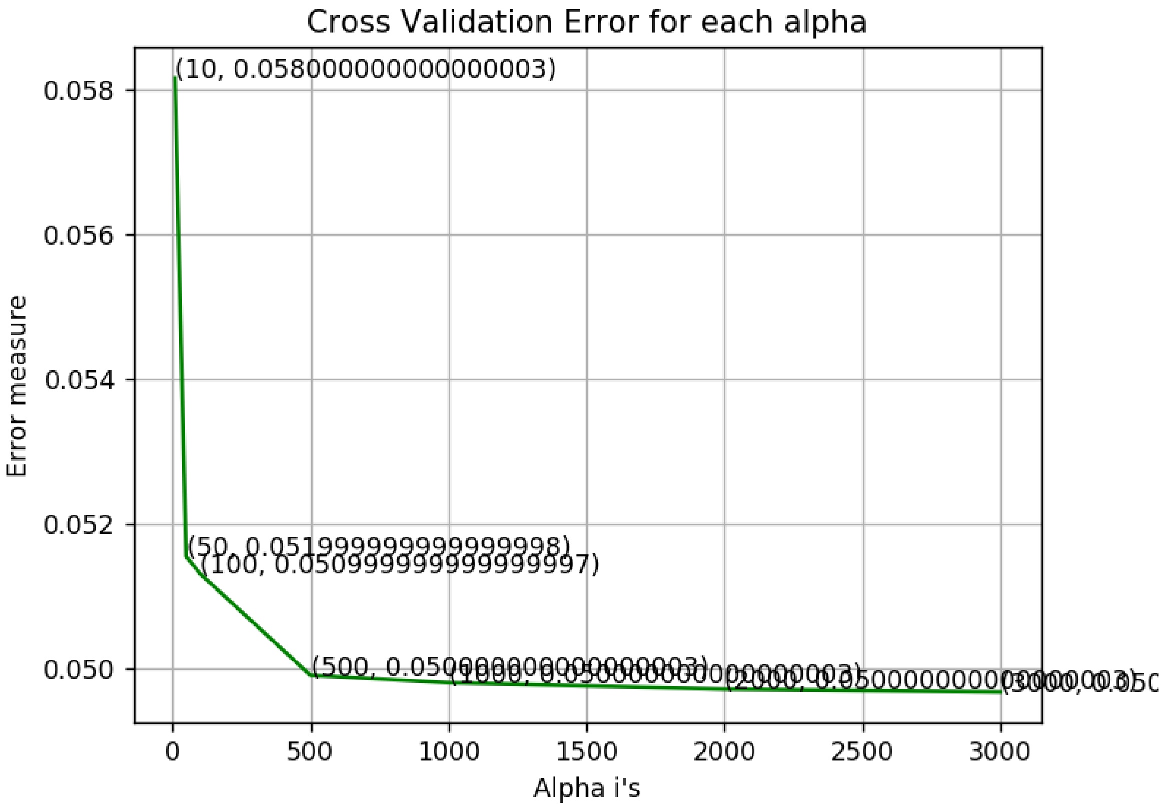
```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.4.3 Random Forest Classifier

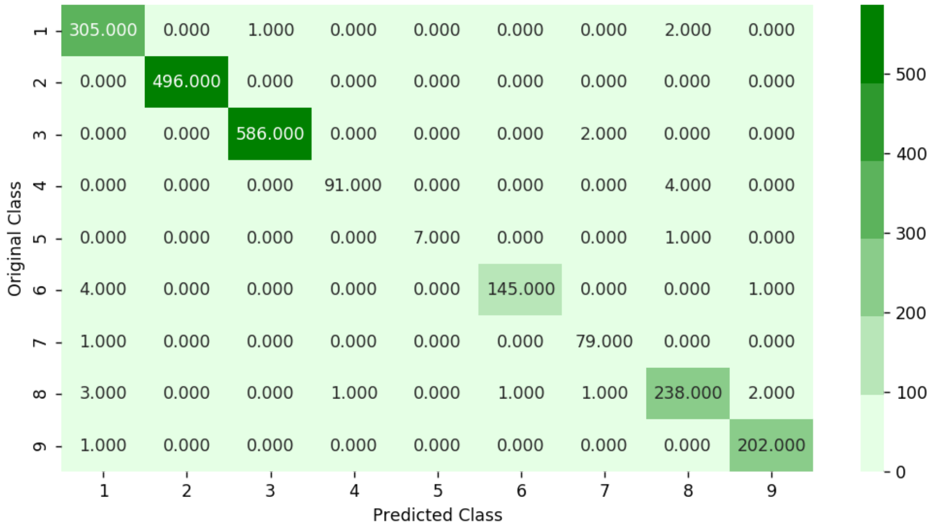
In [161]:

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

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

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

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

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.clas

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
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
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.c
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_c
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

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

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

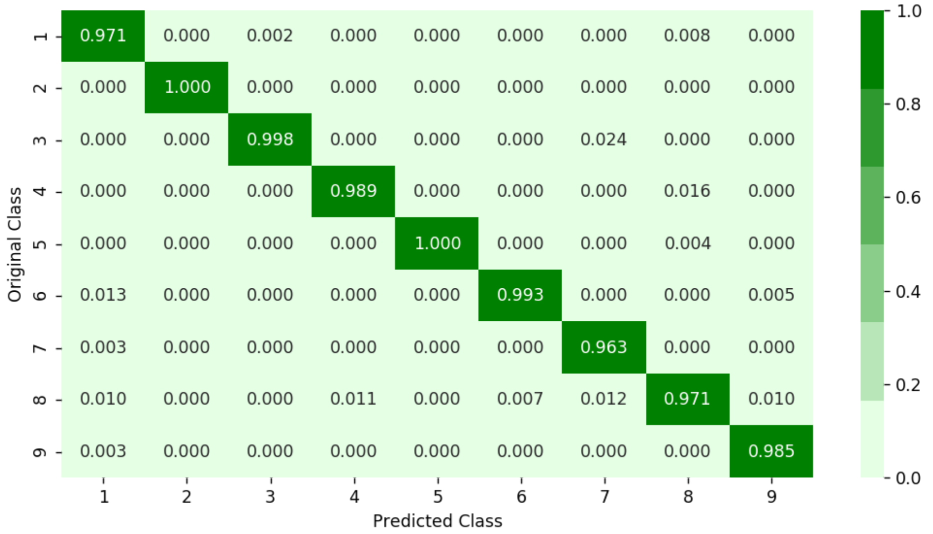


```
log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points  1.14995400184
-------------------------------------------------- Confusion matrix ----------
----------------------------------------

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



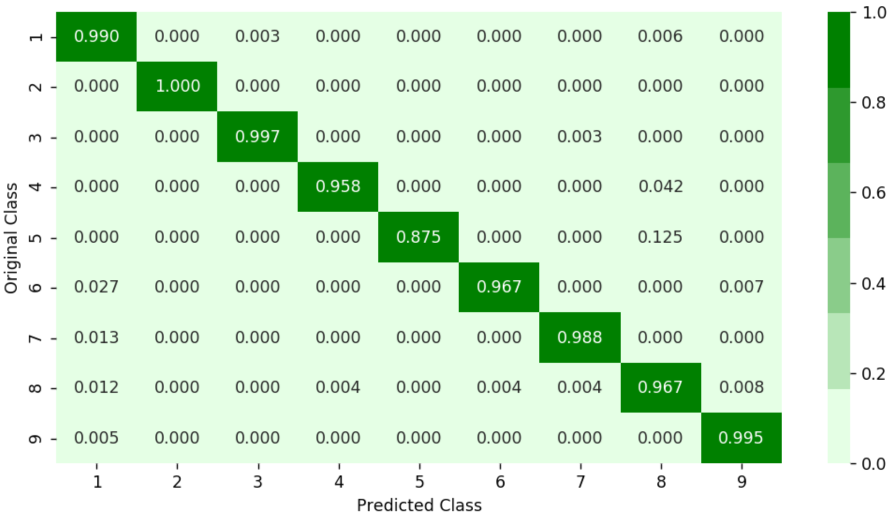
	1	2	3	4	5	6	7	8	9
1	305.000	0.000	1.000	0.000	0.000	0.000	0.000	2.000	0.000
2	0.000	496.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	586.000	0.000	0.000	0.000	2.000	0.000	0.000
4	0.000	0.000	0.000	91.000	0.000	0.000	0.000	4.000	0.000
5	0.000	0.000	0.000	0.000	7.000	0.000	0.000	1.000	0.000
6	4.000	0.000	0.000	0.000	0.000	145.000	0.000	0.000	1.000
7	1.000	0.000	0.000	0.000	0.000	0.000	79.000	0.000	0.000
8	3.000	0.000	0.000	1.000	0.000	1.000	1.000	238.000	2.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	202.000

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

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



	1	2	3	4	5	6	7	8	9
1	0.971	0.000	0.002	0.000	0.000	0.000	0.000	0.008	0.000
2	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.998	0.000	0.000	0.000	0.024	0.000	0.000
4	0.000	0.000	0.000	0.989	0.000	0.000	0.000	0.016	0.000
5	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.004	0.000
6	0.013	0.000	0.000	0.000	0.000	0.993	0.000	0.000	0.005
7	0.003	0.000	0.000	0.000	0.000	0.000	0.963	0.000	0.000
8	0.010	0.000	0.000	0.011	0.000	0.007	0.012	0.971	0.010
9	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.985

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

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.990	0.000	0.003	0.000	0.000	0.000	0.000	0.006	0.000
2	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.997	0.000	0.000	0.000	0.003	0.000	0.000
4	0.000	0.000	0.000	0.958	0.000	0.000	0.000	0.042	0.000
5	0.000	0.000	0.000	0.000	0.875	0.000	0.000	0.125	0.000
6	0.027	0.000	0.000	0.000	0.000	0.967	0.000	0.000	0.007
7	0.013	0.000	0.000	0.000	0.000	0.000	0.988	0.000	0.000
8	0.012	0.000	0.000	0.004	0.000	0.004	0.004	0.967	0.008
9	0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.995

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

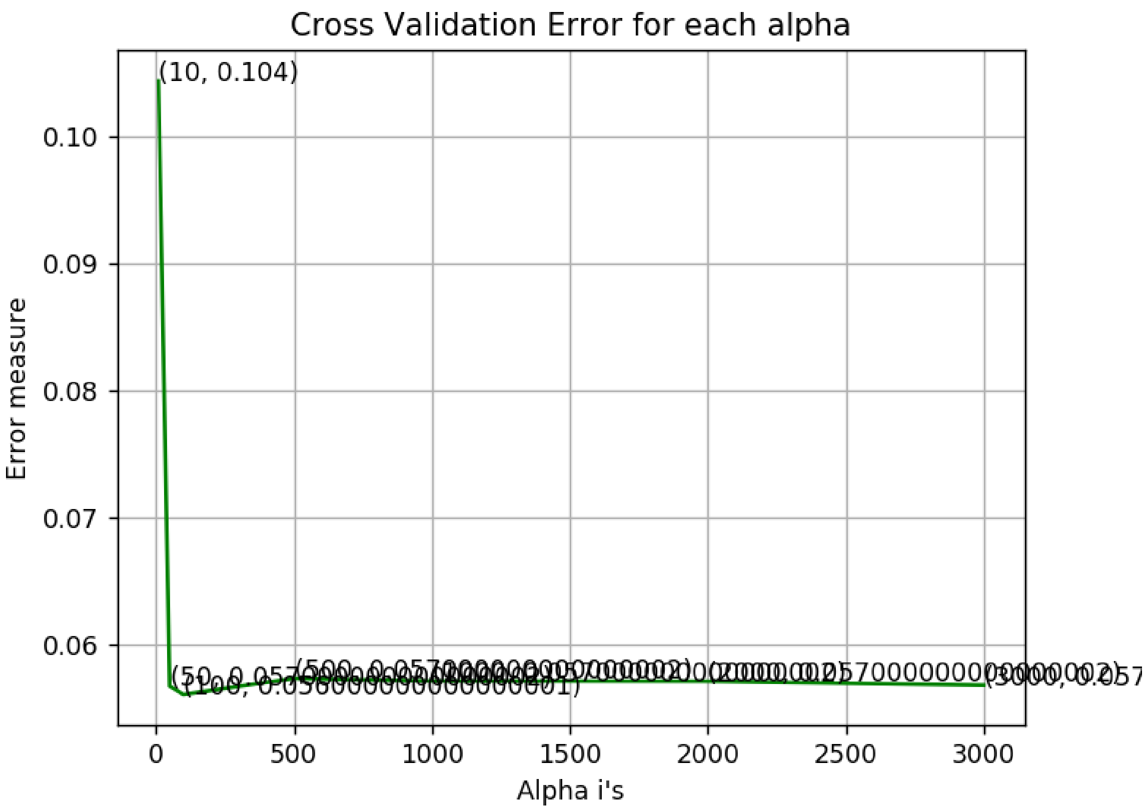
4.4.4 XgBoost Classifier

In [162]:

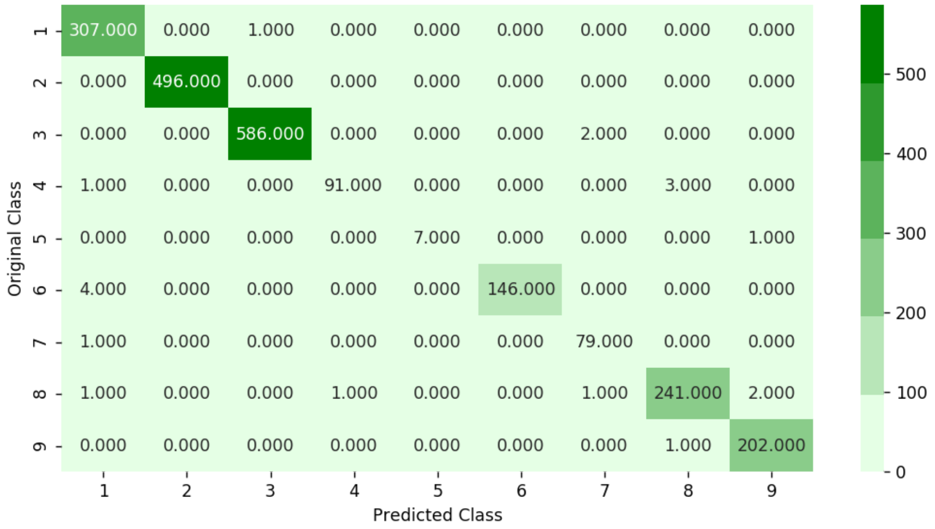
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    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.clas

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
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
predict_y = sig_clf.predict_proba(X_test_asm)
```

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

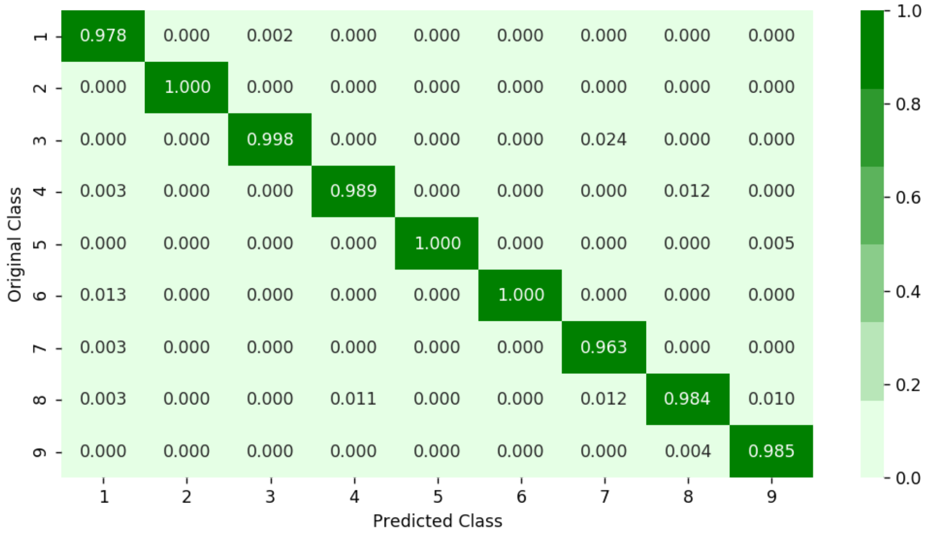
```
log_loss for c =  10 is 0.104344888454
log_loss for c =  50 is 0.0567190635611
log_loss for c =  100 is 0.056075038646
log_loss for c =  500 is 0.057336051683
log_loss for c =  1000 is 0.0571265109903
log_loss for c =  2000 is 0.057103406781
log_loss for c =  3000 is 0.0567993215778

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



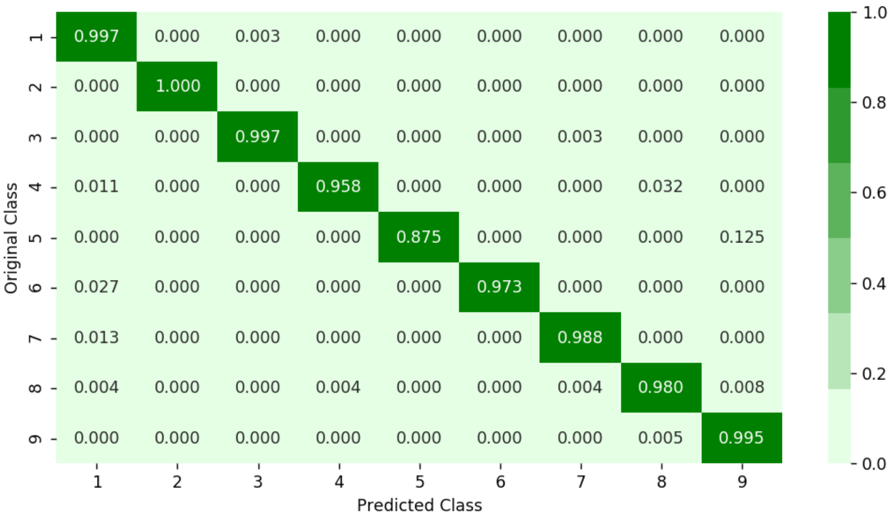
```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.0560750386
46
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
-------------------------------------------------- Confusion matrix ----------
----------------------------------------

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



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

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



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

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



	1	2	3	4	5	6	7	8	9
1	0.997	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.997	0.000	0.000	0.000	0.003	0.000	0.000
4	0.011	0.000	0.000	0.958	0.000	0.000	0.000	0.032	0.000
5	0.000	0.000	0.000	0.000	0.875	0.000	0.000	0.000	0.125
6	0.027	0.000	0.000	0.000	0.000	0.973	0.000	0.000	0.000
7	0.013	0.000	0.000	0.000	0.000	0.000	0.988	0.000	0.000
8	0.004	0.000	0.000	0.004	0.000	0.000	0.004	0.980	0.008
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.995

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

4.4.5 Xgboost Classifier with best hyperparameters

In [163]:

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

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  1.1min remaining:   39.
3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.3min remaining:   23.
0s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  1.4min remaining:    9.
2s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.3min finished
```

Out[163]:

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

In [164]:

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

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

In [170]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

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

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

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

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

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

4.5.1. Merging both asm and byte file features

In [171]:

```python
result.head()
```

Out[171]:

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

5 rows × 260 columns

In [174]:

```python
result_asm.head()
```

Out[174]:

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

5 rows × 54 columns

In [173]:

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

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

In [182]:

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

Out[182]:

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

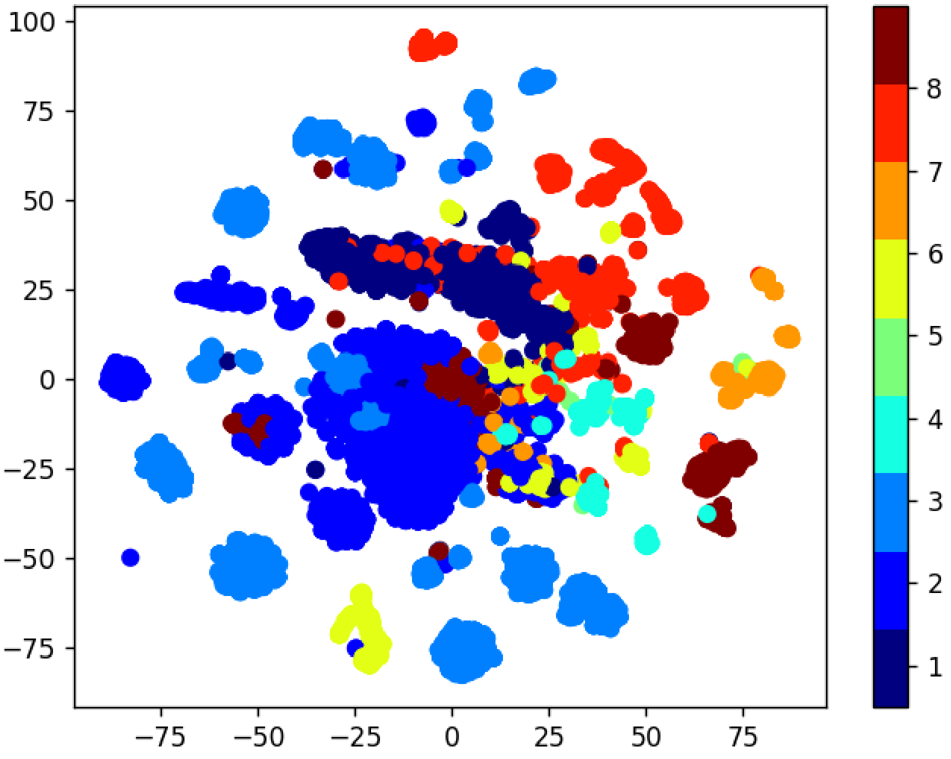
5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

In [181]:

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

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



4.5.3. Train and Test split

In [183]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, res
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tra
```

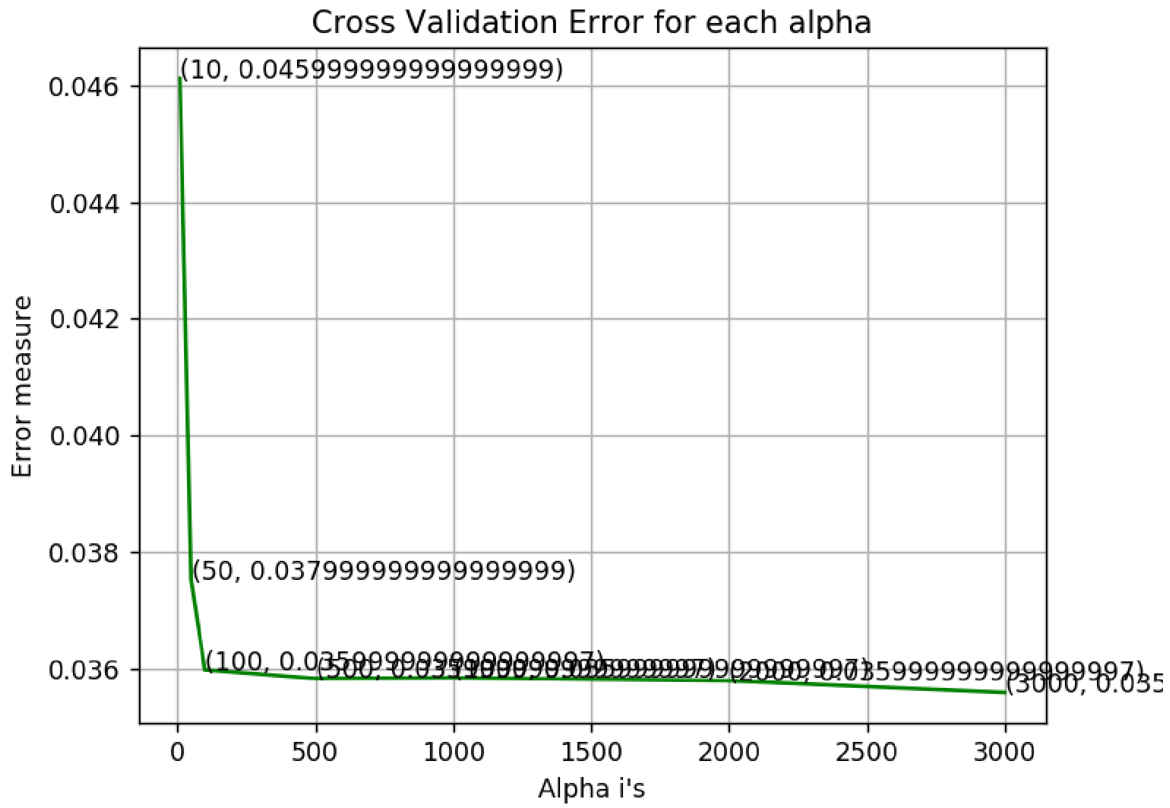
4.5.4. Random Forest Classifier on final features

In [185]:

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

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

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

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

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.cl

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
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
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 for c =  10 is 0.0461221662017
log_loss for c =  50 is 0.0375229563452
log_loss for c =  100 is 0.0359765822455
log_loss for c =  500 is 0.0358291883873
log_loss for c =  1000 is 0.0358403093496
log_loss for c =  2000 is 0.0357908022178
log_loss for c =  3000 is 0.0355909487962

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

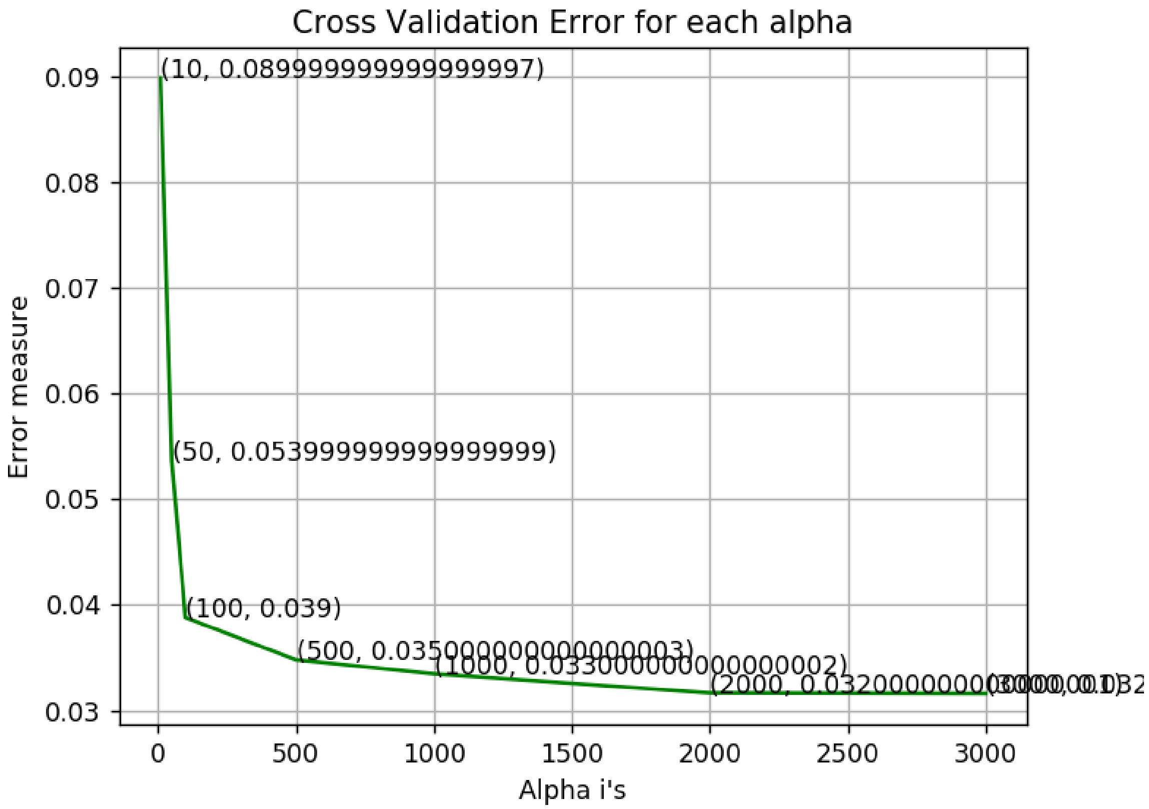


```
For values of best alpha =  3000 The train log loss is: 0.0166267614753
For values of best alpha =  3000 The cross validation log loss is: 0.035590948
7962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

4.5.5. XgBoost Classifier on final features

In [186]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    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.cl

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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
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 for c =  10 is 0.0898979446265
log_loss for c =  50 is 0.0536946658041
log_loss for c =  100 is 0.0387968186177
log_loss for c =  500 is 0.0347960327293
log_loss for c =  1000 is 0.0334668083237
log_loss for c =  2000 is 0.0316569078846
log_loss for c =  3000 is 0.0315972694477

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



```
For values of best alpha =  3000 The train log loss is: 0.0111918809342
For values of best alpha =  3000 The cross validation log loss is: 0.031597269
4477
For values of best alpha =  3000 The test log loss is: 0.0323978515915
```

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

In [187]:

```
x_cfl=XGBClassifier()

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

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  4.5min remaining:  2.6m
in
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remaining:  1.8m
in
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remaining:   44.
5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

Out[187]:

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

In [188]:

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

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

In [189]:

```

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

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

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsamp
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  3000 The train log loss is: 0.0121922832297
For values of best alpha =  3000 The cross validation log loss is: 0.034495548
7471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```