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 (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**

In [ ]:

```
import sys
import os
from math import log
import numpy as np
import scipy as sp
from PIL import Image
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook as tqdm
def saveimg(array,name):
#     print (name)
    if array.shape[1]!=16:
        assert(False)
    b=int((array.shape[0]*16)**(0.5))
    b=2**(int(log(b)/log(2))+1)
    a=int(array.shape[0]*16/b)
    #print a,b,array.shape
    array=array[:int(a*b/16),:]
    array=np.reshape(array,(a,b))
    #print array.shape
    im = Image.fromarray(np.uint8(array))
    im.save('train/trainImageBytes/'+name+'.jpg', "BMP")
files=os.listdir('Malware/train')
c=0
for x in tqdm(files):
    if '.bytes' != x[-6:]:
        continue
    f=open('Malware/train/'+x)
    array=[]
    c+=1
    for line in f:
        xx=line.split()
        if len(xx)!=17:
            continue
        #if xx[1]=='??':
        #    break
        array.append([int(i,16) if i!='??' else 0 for i in xx[1:] ])
    saveimg(np.array(array),x)
    del array
    f.close()
```

2.1.2. Example Data Point

.asm file

```
.text:00401000                                  assume es:nothing, ss:nothin
g, ds:_data, fs:nothing, gs:nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24 08                          lea     eax, [esp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                                mov     esi, ecx
.text:00401008 E8 1C 1B 00 00                           call    ??0exception@
std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08 BB 42 00                        mov     dword ptr [es
i], offset off_42BB08
.text:00401013 8B C6                                mov     eax, esi
.text:00401015 5E                               pop     esi
.text:00401016 C2 04 00                             retn    4
.text:00401016                          ; ---------------------------------
---------------------------------------
.text:00401019 CC CC CC CC CC CC CC                     align 10h
.text:00401020 C7 01 08 BB 42 00                        mov     dword ptr [ec
x], offset off_42BB08
.text:00401026 E9 26 1C 00 00                           jmp     sub_402C51
.text:00401026                          ; ---------------------------------
---------------------------------------
.text:0040102B CC CC CC CC CC                           align 10h
.text:00401030 56                               push    esi
.text:00401031 8B F1                                mov     esi, ecx
.text:00401033 C7 06 08 BB 42 00                        mov     dword ptr [es
i], offset off_42BB08
.text:00401039 E8 13 1C 00 00                           call    sub_402C51
.text:0040103E F6 44 24 08 01                           test    byte ptr [esp
+8], 1
.text:00401043 74 09                                jz      short loc_40104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E 00 00                           call    ??3@YAXPAX@Z
; operator delete(void *)
.text:0040104B 83 C4 04                             add     esp, 4
.text:0040104E
.text:0040104E                          loc_40104E:                 ; CODE XR
EF: .text:00401043□j
.text:0040104E 8B C6                                mov     eax, esi
.text:00401050 5E                               pop     esi
.text:00401051 C2 04 00                             retn    4
.text:00401051                          ; ---------------------------------
---------------------------------------
```

.bytes file

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- **Multi class log-loss**
- **Confusion matrix**

2.2.3. Machine 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=VLQTRILGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_plB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import networkx as nx
import matplotlib
import random
from tqdm import tqdm_notebook as tqdm
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 [40]:

```
from prettytable import PrettyTable as pt
conclusion_table = pt()

conclusion_table.field_names = ["feature_name(Xgboost)", "train_Logloss", "Test_Loglos
s"]
```

Don't run if already sepearated

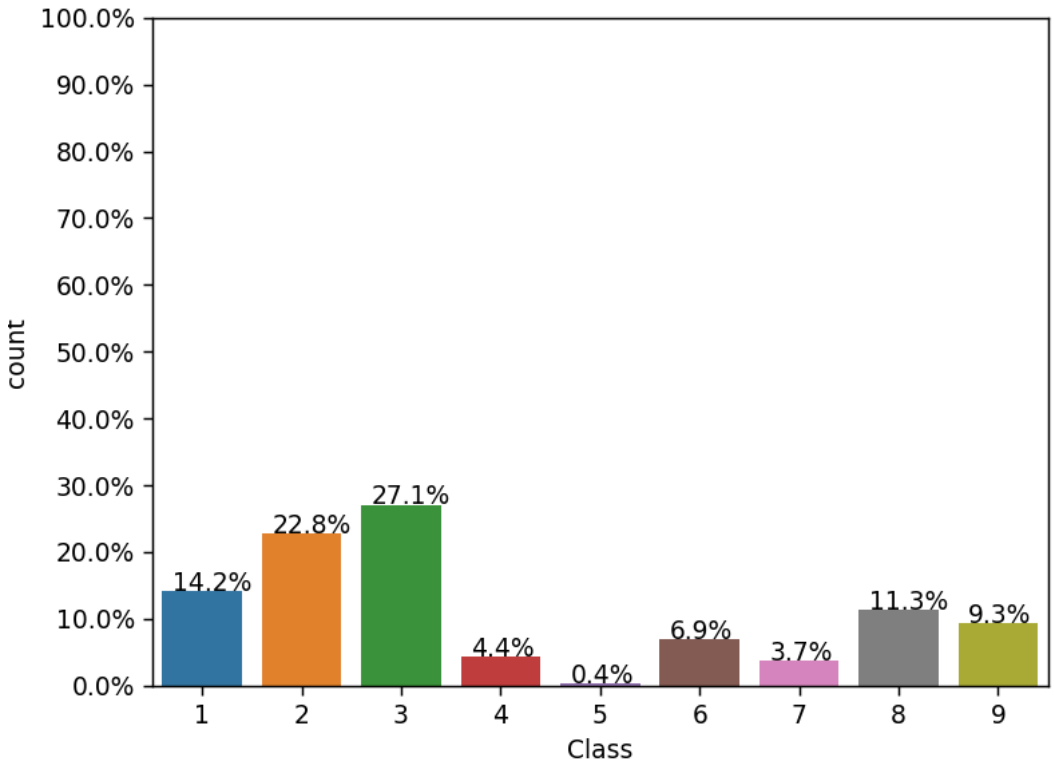
In [11]:

```
#separating byte files and asm files

source = 'Malware/train'
destination_bytes = 'train/byteFiles'
# destination_asm = 'train/asmFiles'

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

# if not os.path.isdir(destination_asm):
#     os.makedirs(destination_asm)

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

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

3.1. Distribution of malware classes in whole data set

In [3]:

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

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [4]:

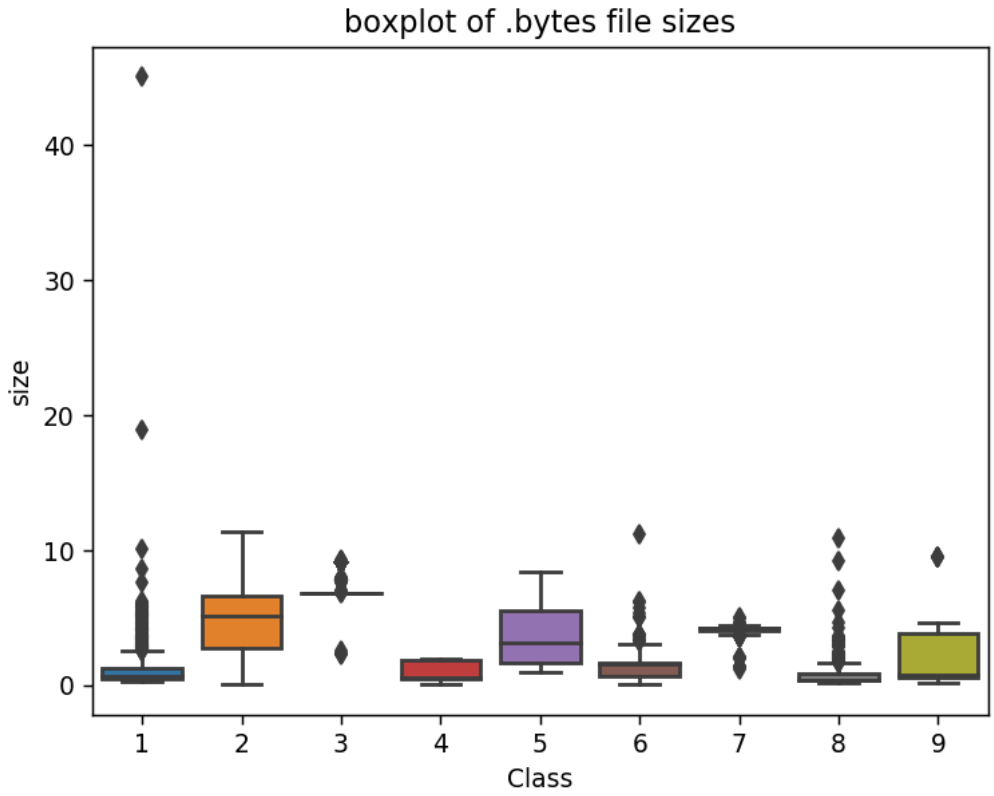
```
#file sizes of byte files

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

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

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

In [5]:

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



3.2.3 feature extraction from byte files

In [238]:

```
from tqdm import tqdm_notebook as tqdm
#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

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

In [237]:

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

print("Creating BOW from 256+?? feaure")
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,
17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,
34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,
51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,
6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,87,88,89,8a,
8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,
a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,
c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,
e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,
ff,??")
for file in tqdm(files):
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('train/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 [239]:

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

In [240]:

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

In [6]:

```
result=pd.read_csv("result_with_size.csv")
result = result.drop(['Unnamed: 0'], axis=1)
result.head()
```

Out[6]:

	ID	0	1	2	3	4	5	6	7	8	...	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	

5 rows × 260 columns

In [7]:

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

In [8]:

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

Out[8]:

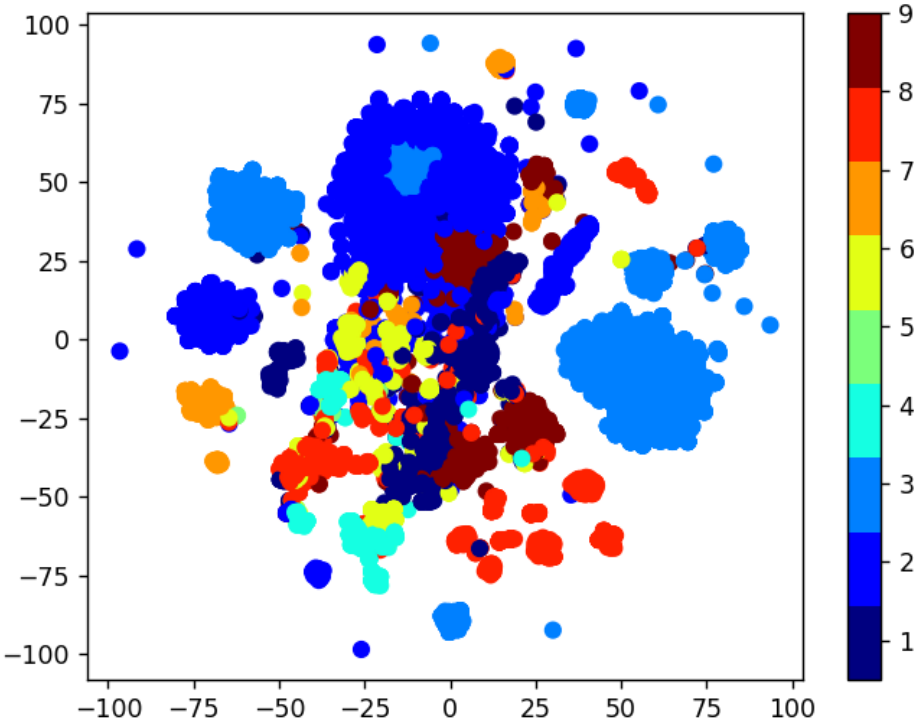
	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.0
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0

5 rows × 260 columns

3.2.4 Multivariate Analysis

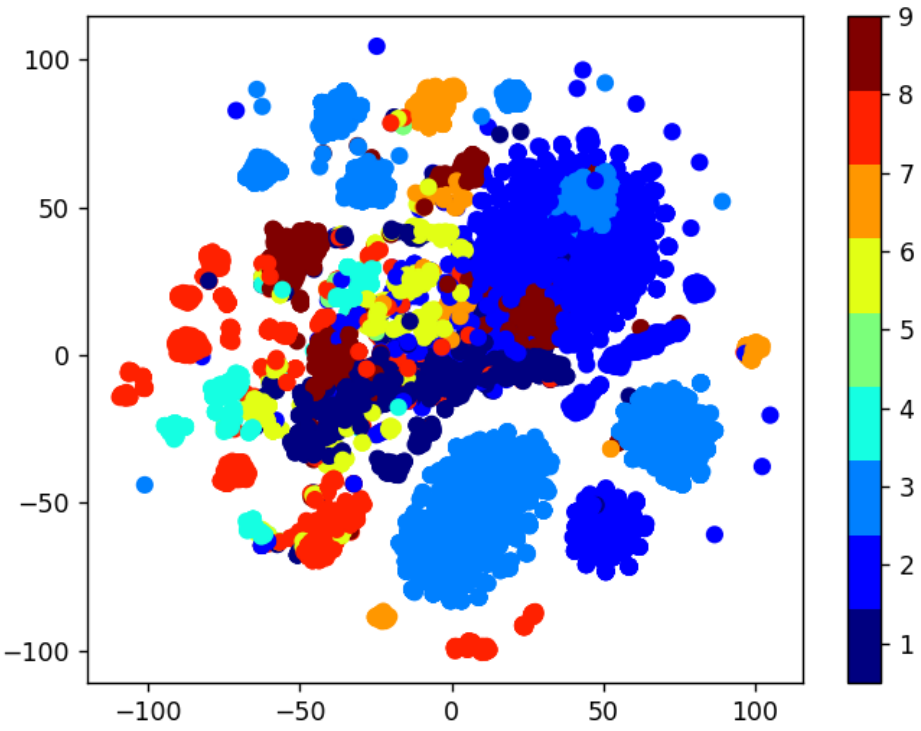
In [0]:

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



In [0]:

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



Train Test split

In [0]:

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

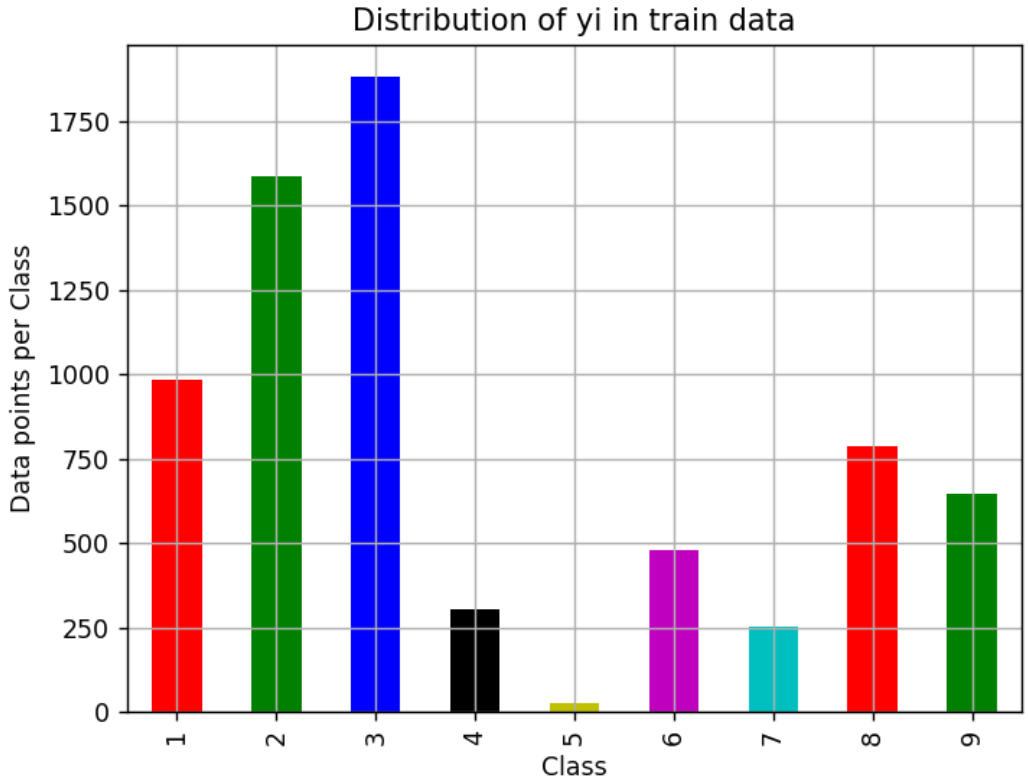
In [0]:

```
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])
```

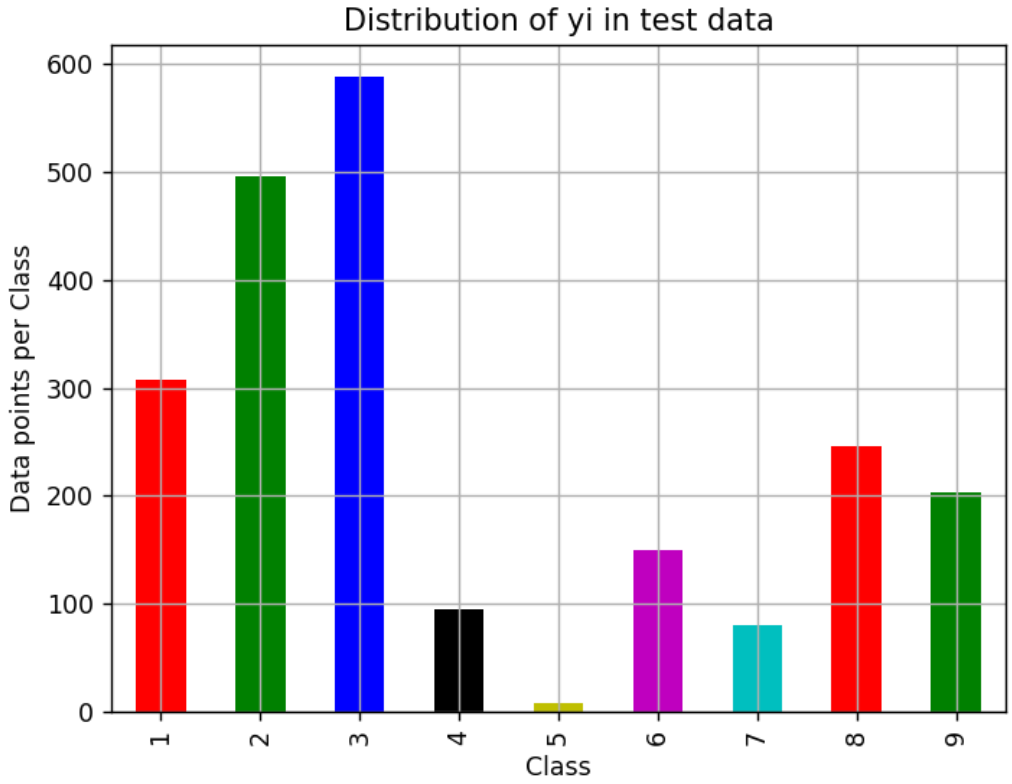
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [0]:

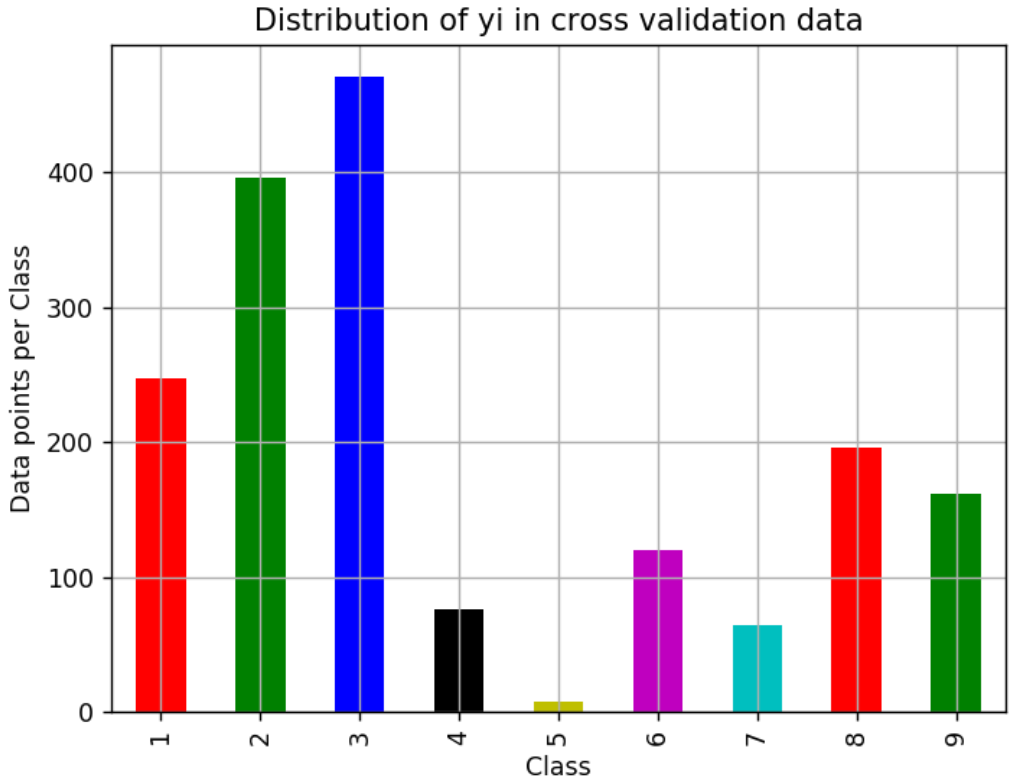
```
# it returns a dict, keys as class labels and values as the number of data points in th
at class
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i
], '(', np.round((train_class_distribution.values[i]/y_train.shape[0]*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],
'(', np.round((test_class_distribution.values[i]/y_test.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],
'(', np.round((cv_class_distribution.values[i]/y_cv.shape[0]*100), 3), '%)')
```




```
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 %)
--------------------------------------------------------------------------------
------
```




```
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 %)
------------------------------------------------------------------------
------
```




```
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 [33]:

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

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

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

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

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

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

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

    # representing B in heatmap format
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
bels)
    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

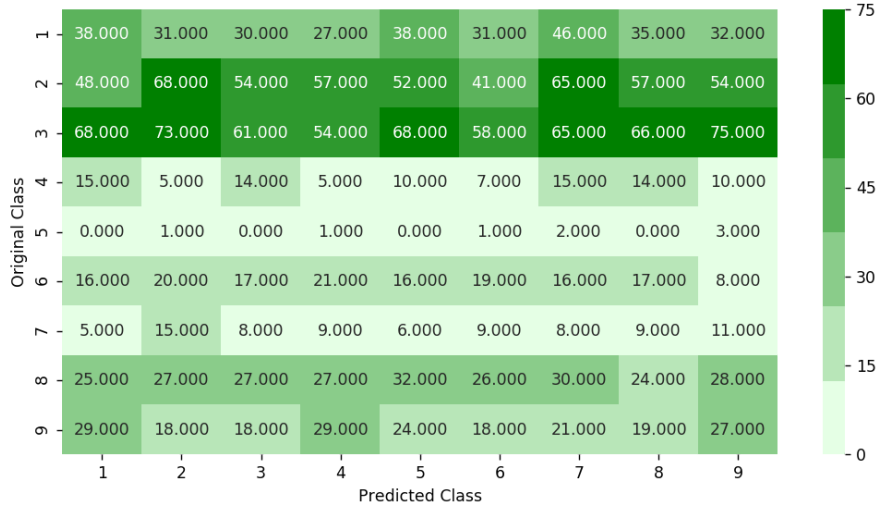
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

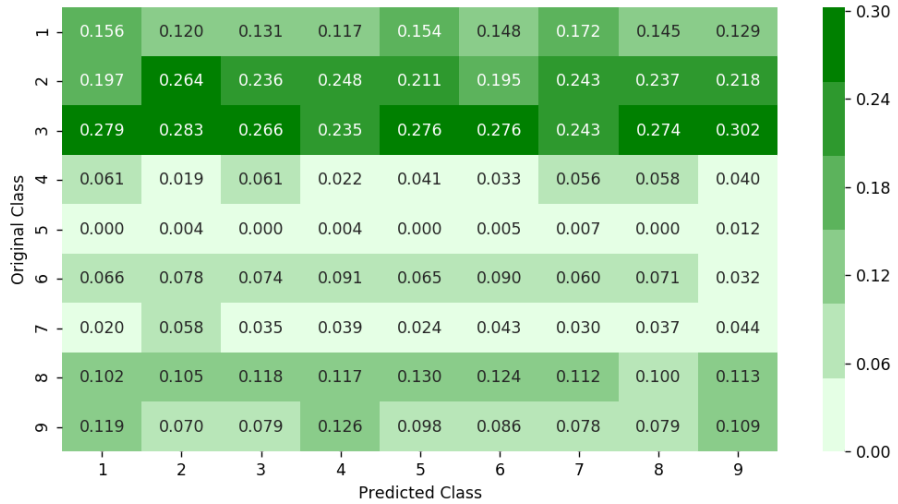
In [0]:

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

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

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


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

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

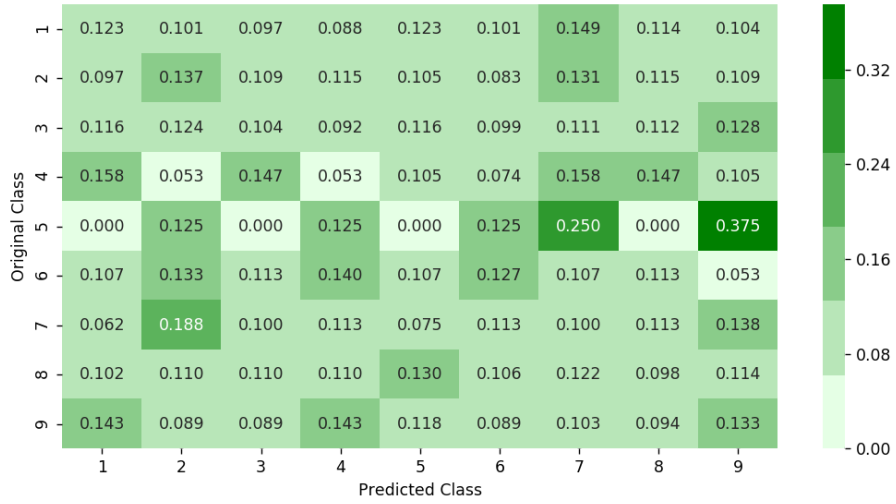
```
Log loss on Cross Validation Data using Random Model 2.45615644965
Log loss on Test Data using Random Model 2.48503905509
Number of misclassified points  88.5004599816
---------------------------------------------------- Confusion matrix ------
------------------------------------------
```




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




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




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

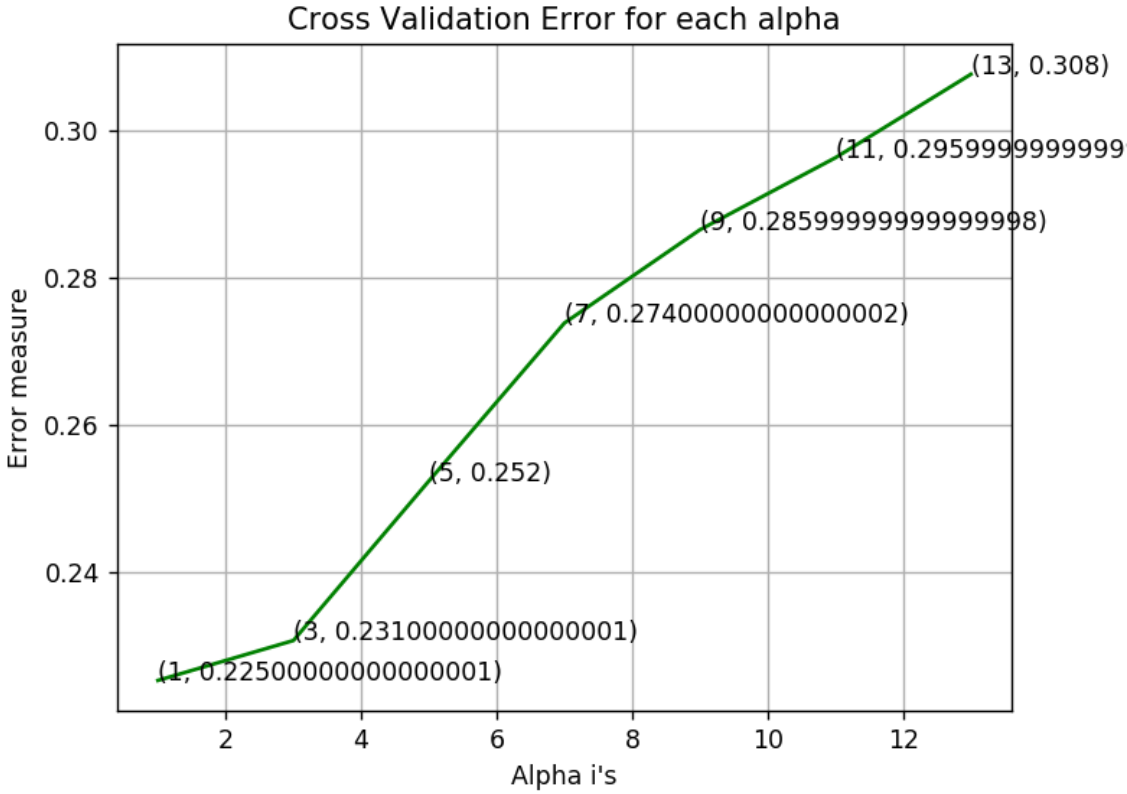
4.1.2. K Nearest Neighbour Classification

In [0]:

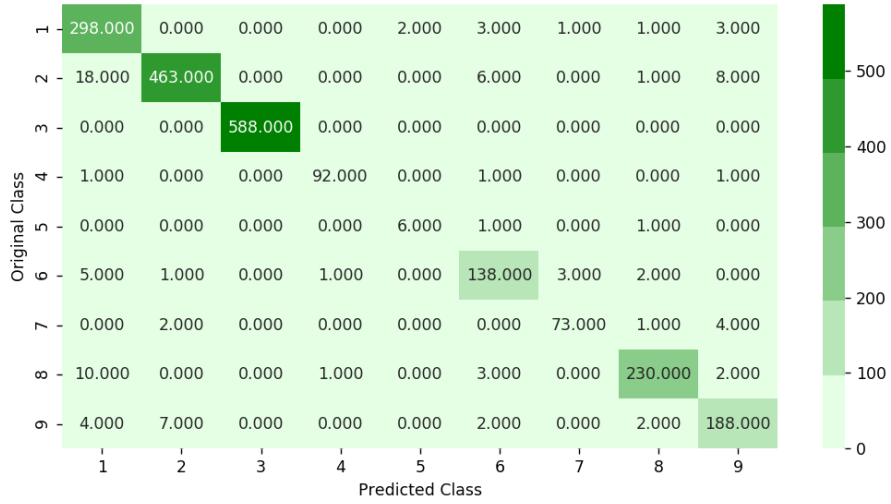
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
nerated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=3
0, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k
-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----------------------------------


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

alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-1
5))

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

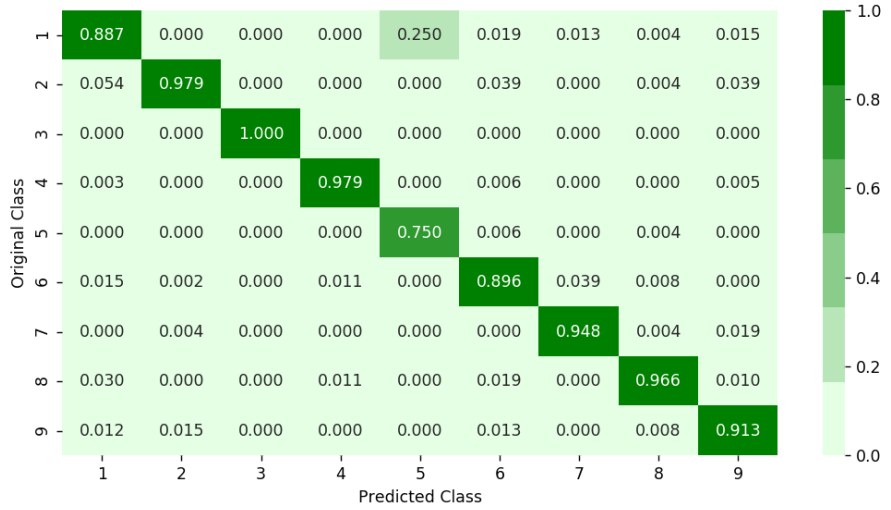
```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154
```



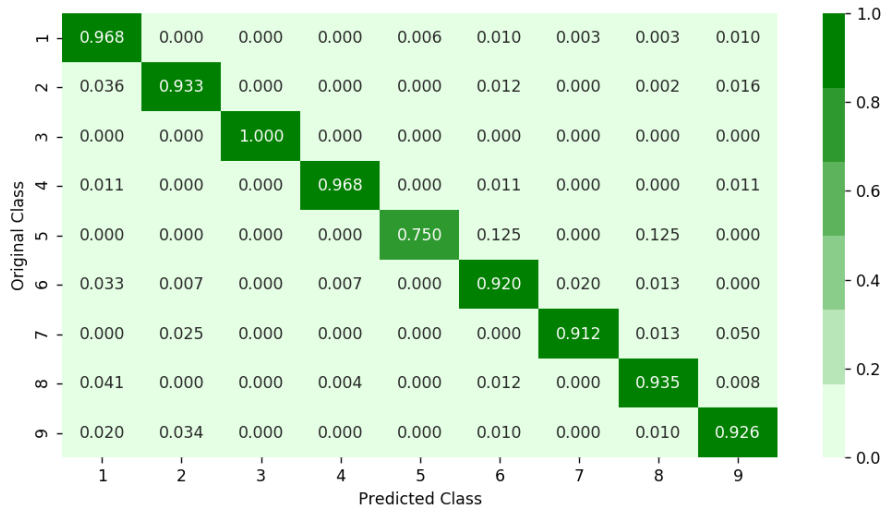

```
For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.22538623
7304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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




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

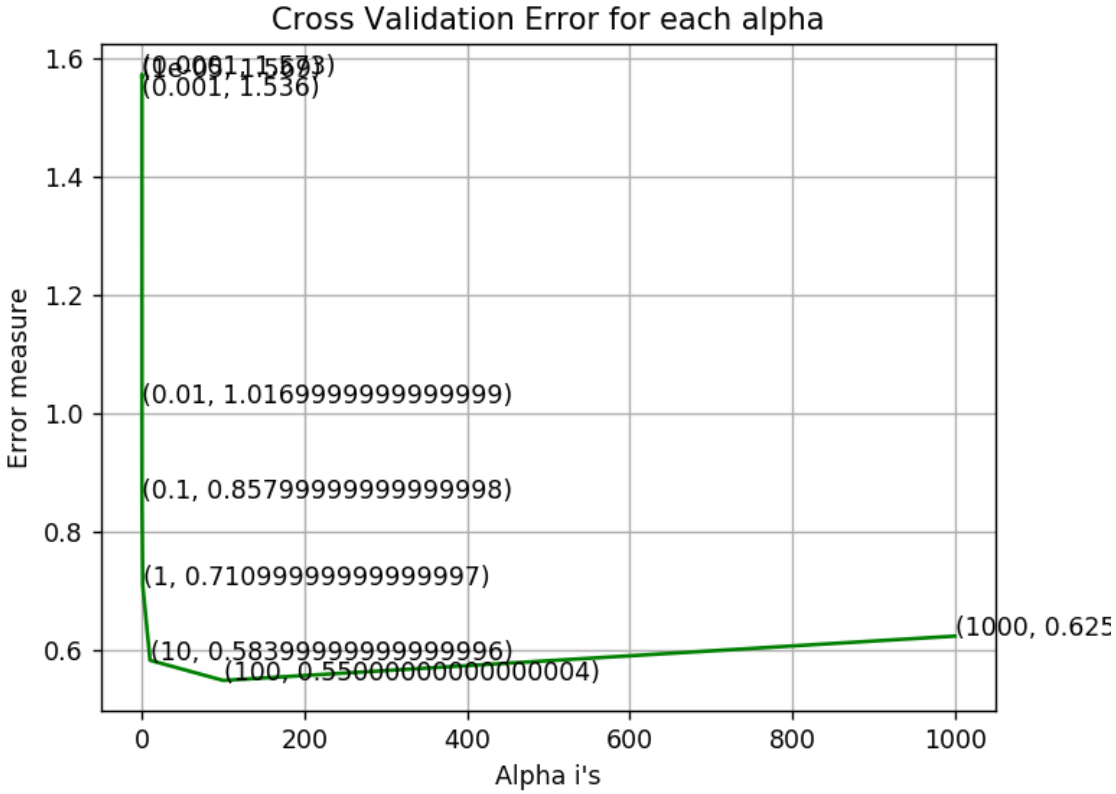
4.1.3. Logistic Regression

In [0]:

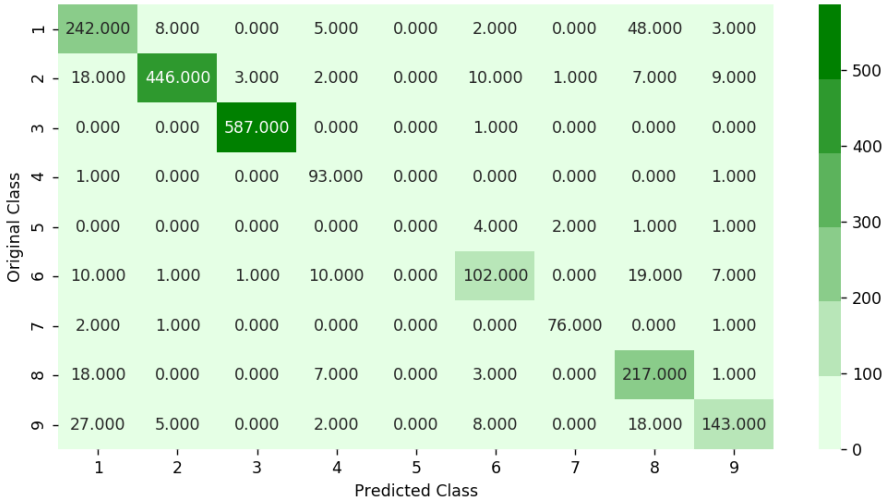
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

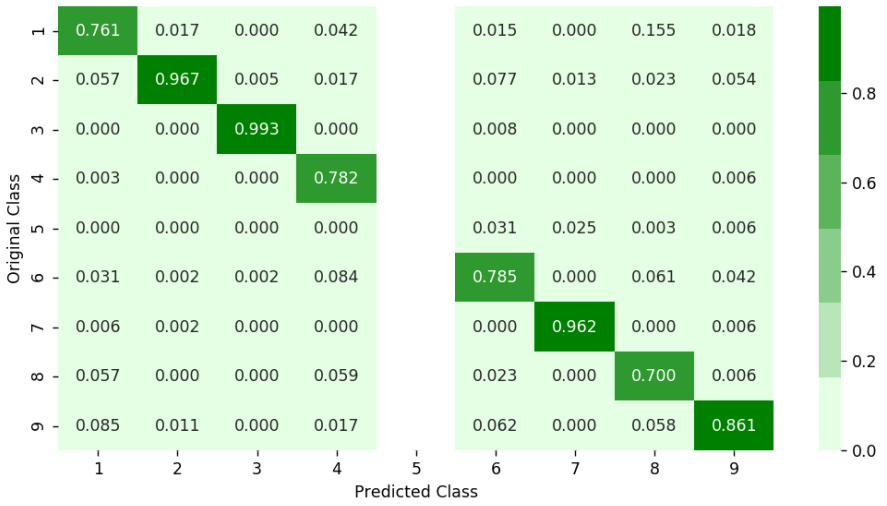
```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =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



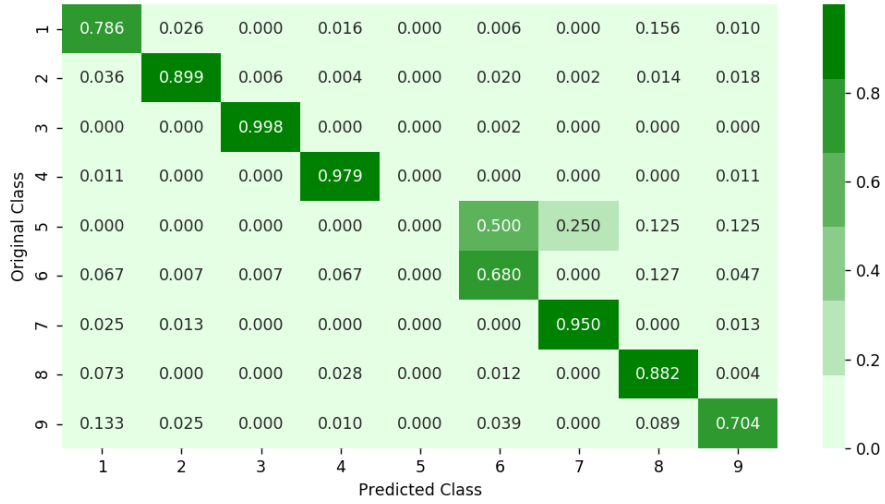
```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```




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




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




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

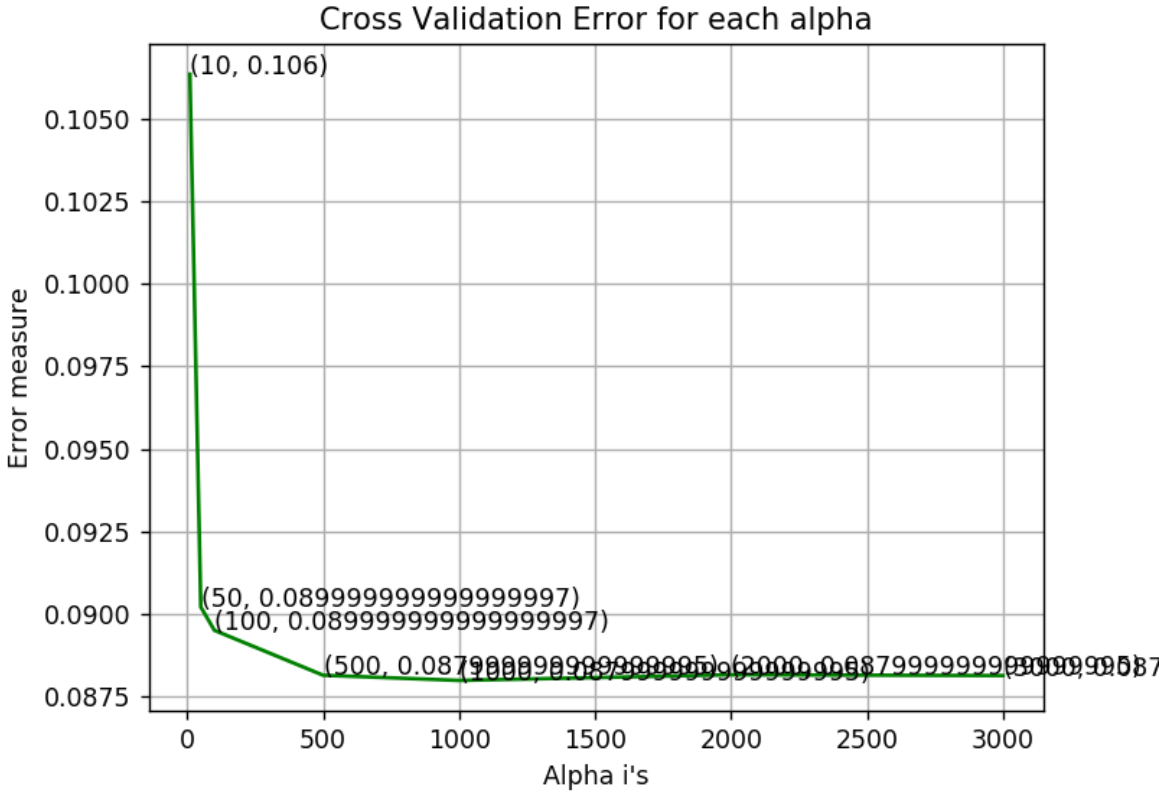
4.1.4. Random Forest Classifier

In [0]:

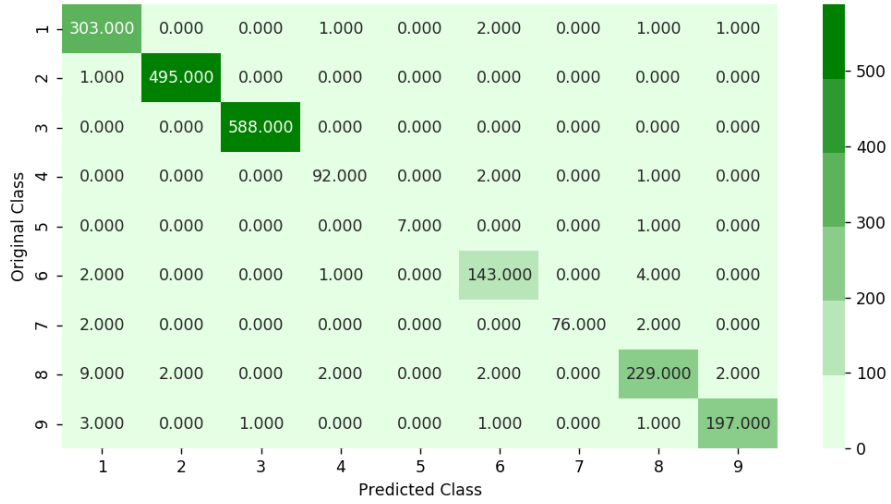
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
e, verbose=0, warm_start=False,
# class_weight=None)

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/r
andom-forest-and-their-construction-2/
# ------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-1
5))

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

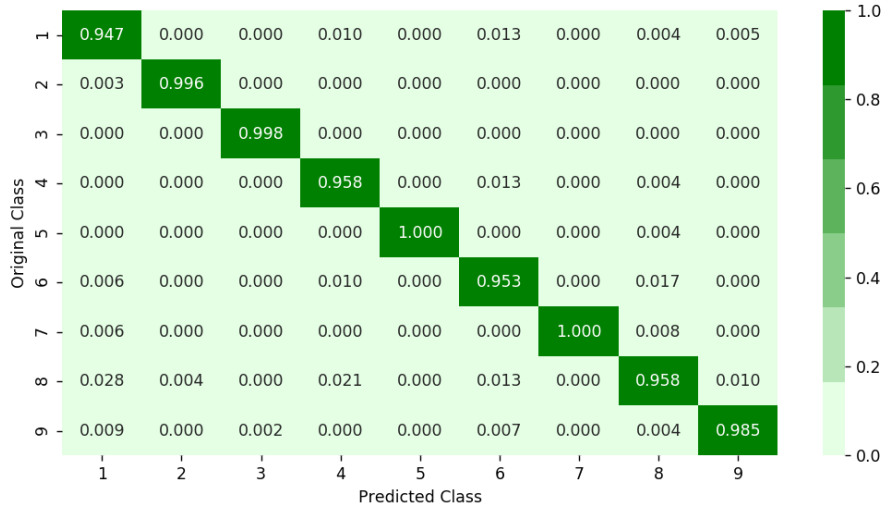
```
log_loss for c =  10 is 0.106357709164
log_loss for c =  50 is 0.0902124124145
log_loss for c =  100 is 0.0895043339776
log_loss for c =  500 is 0.0881420869288
log_loss for c =  1000 is 0.0879849524621
log_loss for c =  2000 is 0.0881566647295
log_loss for c =  3000 is 0.0881318948443
```



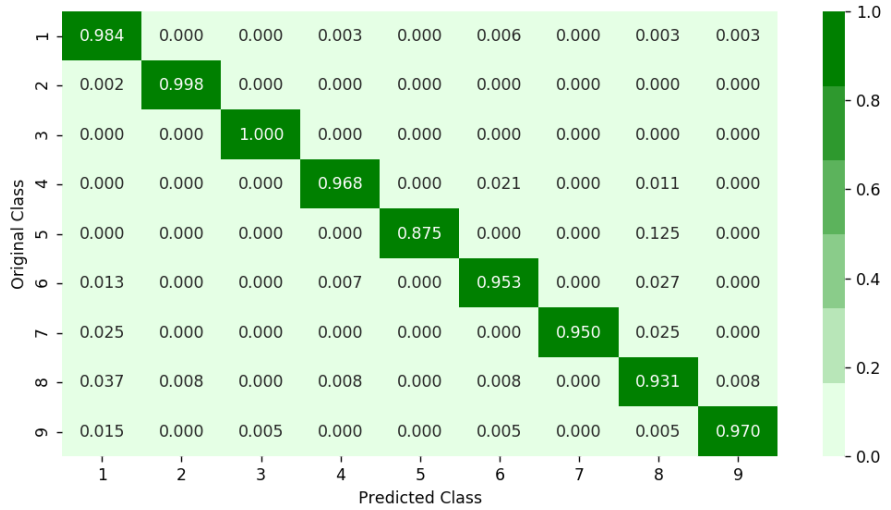

```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.08798
49524621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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



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

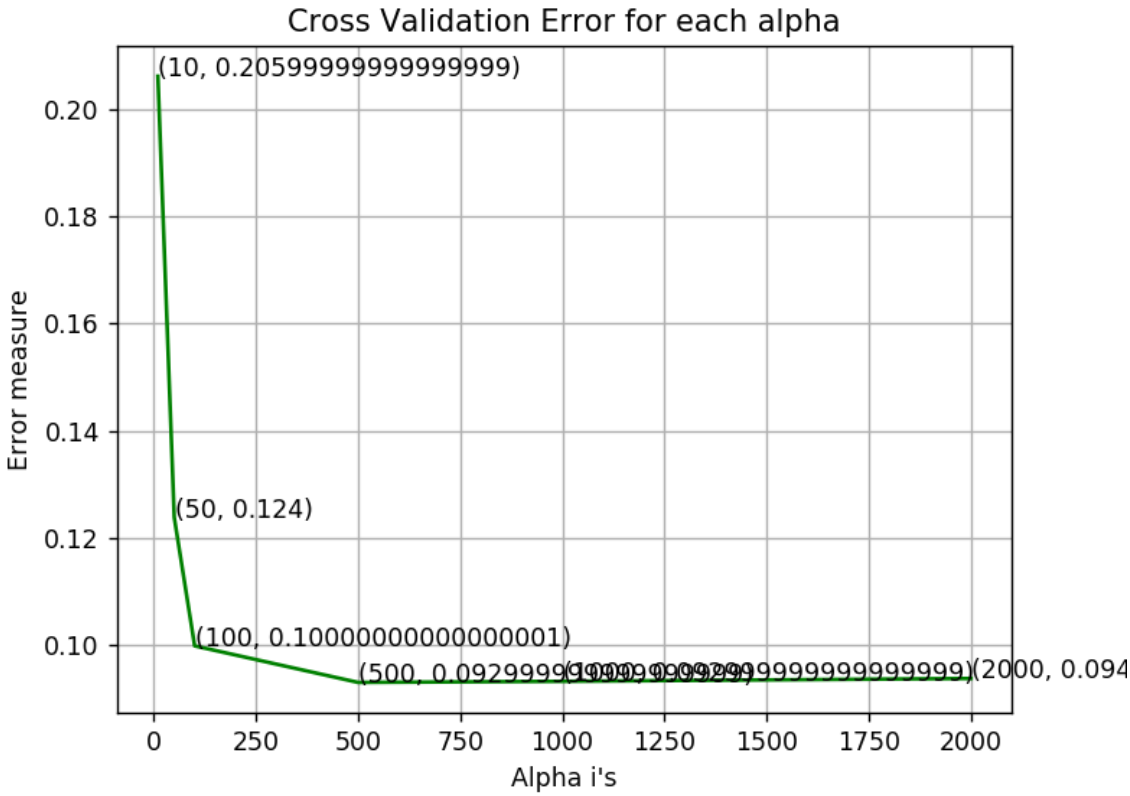
4.1.5. XgBoost Classification

In [0]:

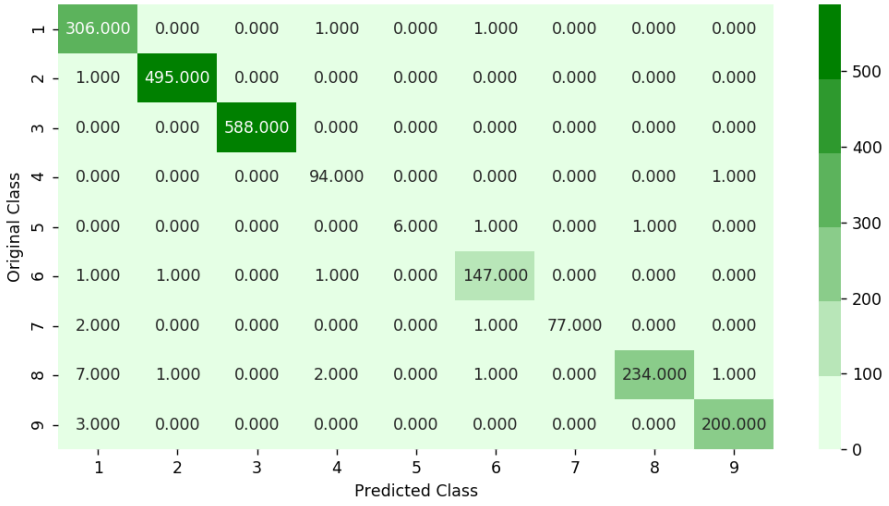
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
sig_clf.fit(X_train, y_train)

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

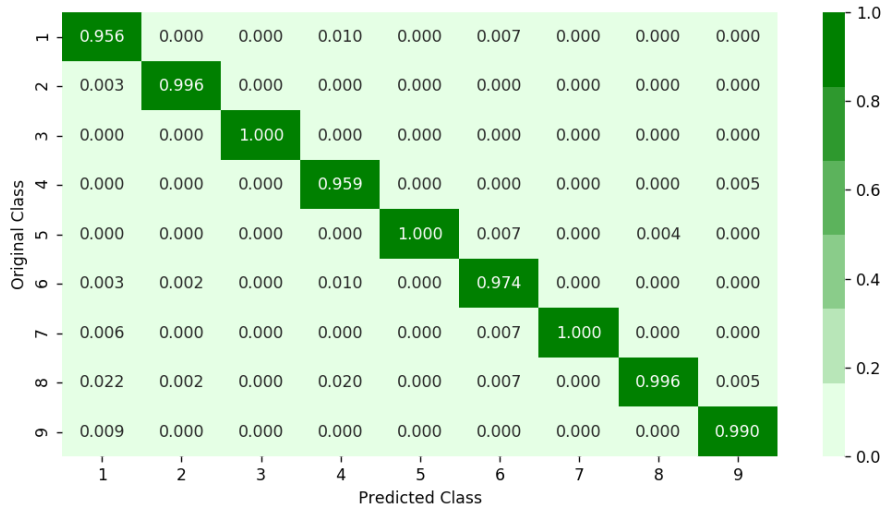
```
log_loss for c =  10 is 0.20615980494
log_loss for c =  50 is 0.123888382365
log_loss for c =  100 is 0.099919437112
log_loss for c =  500 is 0.0931035681289
log_loss for c =  1000 is 0.0933084876012
log_loss for c =  2000 is 0.0938395690309
```



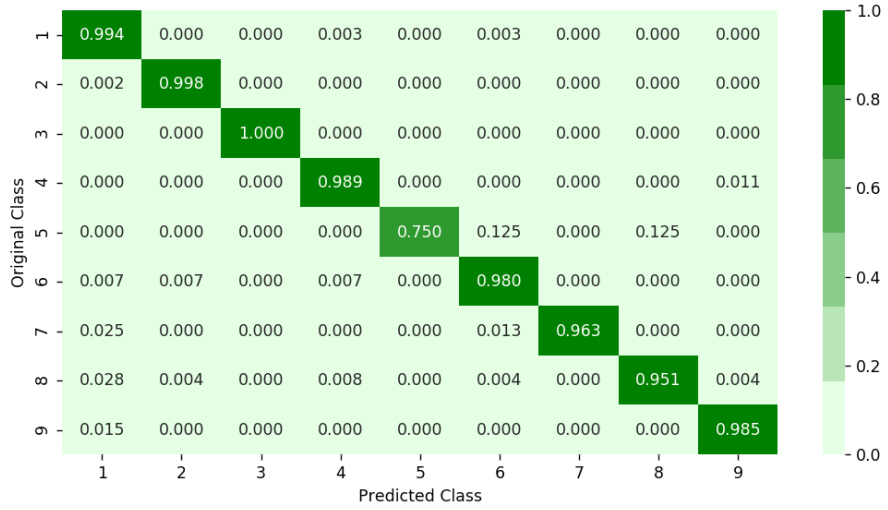

```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.093103
5681289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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




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




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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [0]:

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

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

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

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

Out[0]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, col
sample_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], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5,
1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [0]:

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

```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.0
5, 'colsample_bytree': 0.5}
```

In [0]:

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

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

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

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

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

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

In [ ]:

```
conclusion_table.add_row(["bytes_Bow",log_loss(y_train, predict_y),log_loss(y_test, predict_y)])
```

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In parallel w e 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/b logs.
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 [0]:

```
#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:','.edat
a:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

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

```
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


#same as above
def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edat
a:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','j
z','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
```

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

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edat
a:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','j
z','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('thrid/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fourthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edat
a:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
```

```
'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','j
z','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fifthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edat
a:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','j
z','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
    files = os.listdir('fifth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
```

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


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

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

In [9]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the ext
racted features from .asm files
# 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[9]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.r
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [10]:

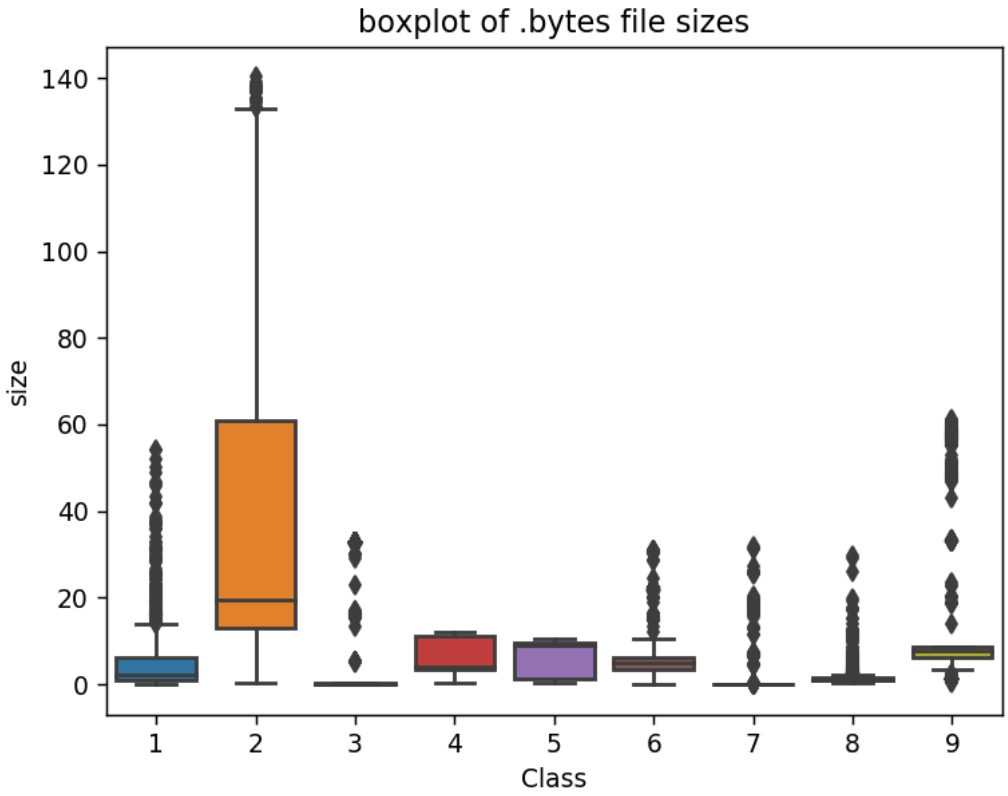
```
#file sizes of byte files


folders = ['first', 'second','third','fourth','fifth']
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for folder in folders:
    files=os.listdir('train/asmFiles_spilt/{}'.format(folder))
    for file in files:
        # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
        # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_
nlink=1, st_uid=0, st_gid=0,
        # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151963852
2)
        # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.h
tm
        statinfo=os.stat('train/asmFiles_spilt/'+folder+'/'+file)
        # split the file name at '.' and take the first part of it i.e the file name
        file=file.split('.')[0]
        if any(file == filename for filename in filenames):
            i=filenames.index(file)
            class_bytes.append(class_y[i])
            # converting into Mb's
            sizebytes.append(statinfo.st_size/(1024.0*1024.0))
            fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

```
                     ID      size  Class
0  01kcPWA9K2BOxQeS5Rju  0.078190      1
1  04EjIdbPV5e1XroFOpiN  4.243397      1
2  05EeG39MTRrI6VY21DPd  0.935457      1
3  05LHG8fR3iPn6agIo9z7  6.637392      6
4  06arUi9q3wHS2C8RZxeB  0.131120      3
```

4.2.1.2 Distribution of .asm file sizes

In [11]:

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



In [12]:

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

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

Out[12]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.r
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	

5 rows × 54 columns

In [13]:

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

Out[13]:

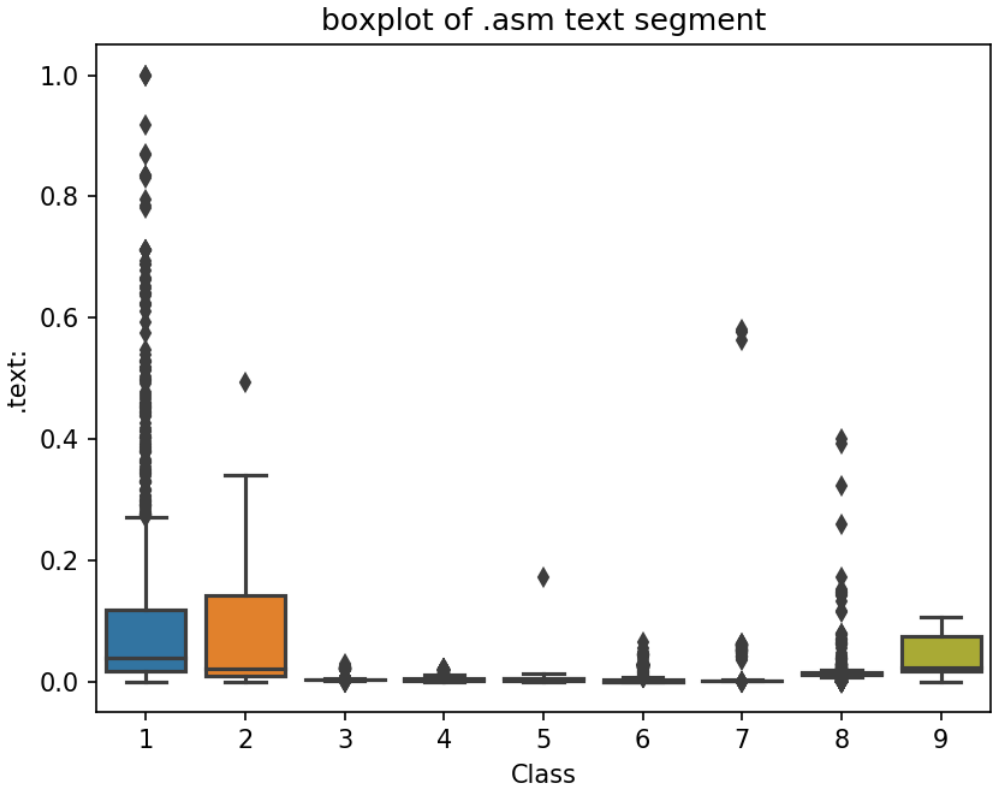
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [37]:

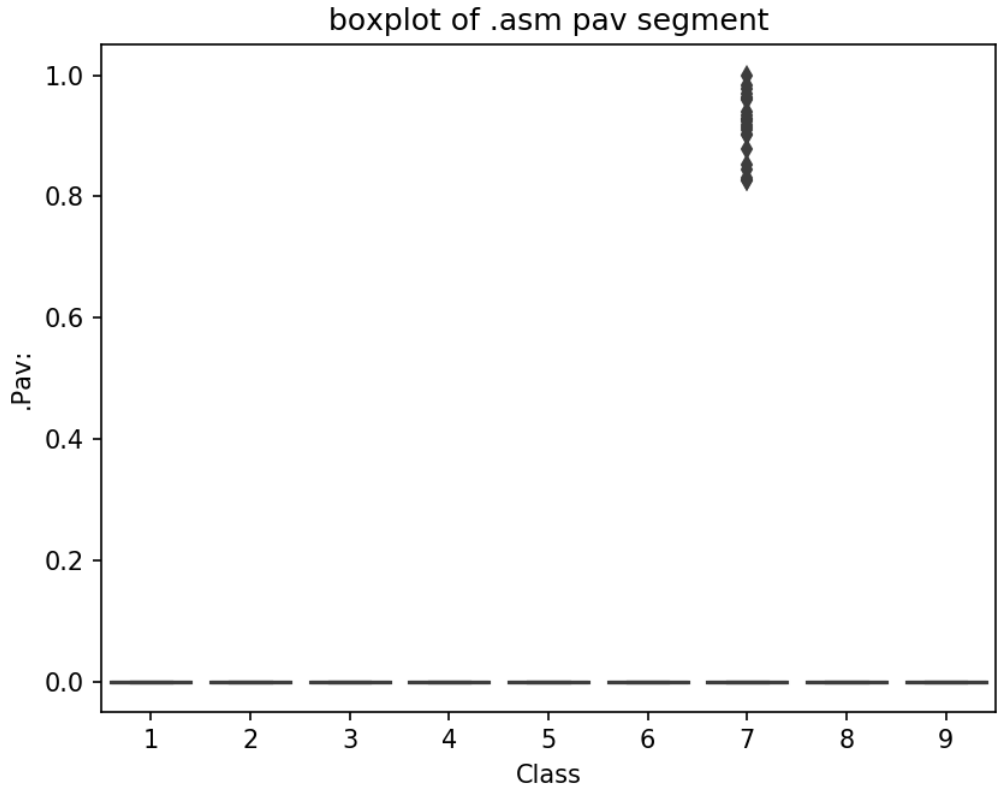
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()Univariate analysis on  asm file features
```



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

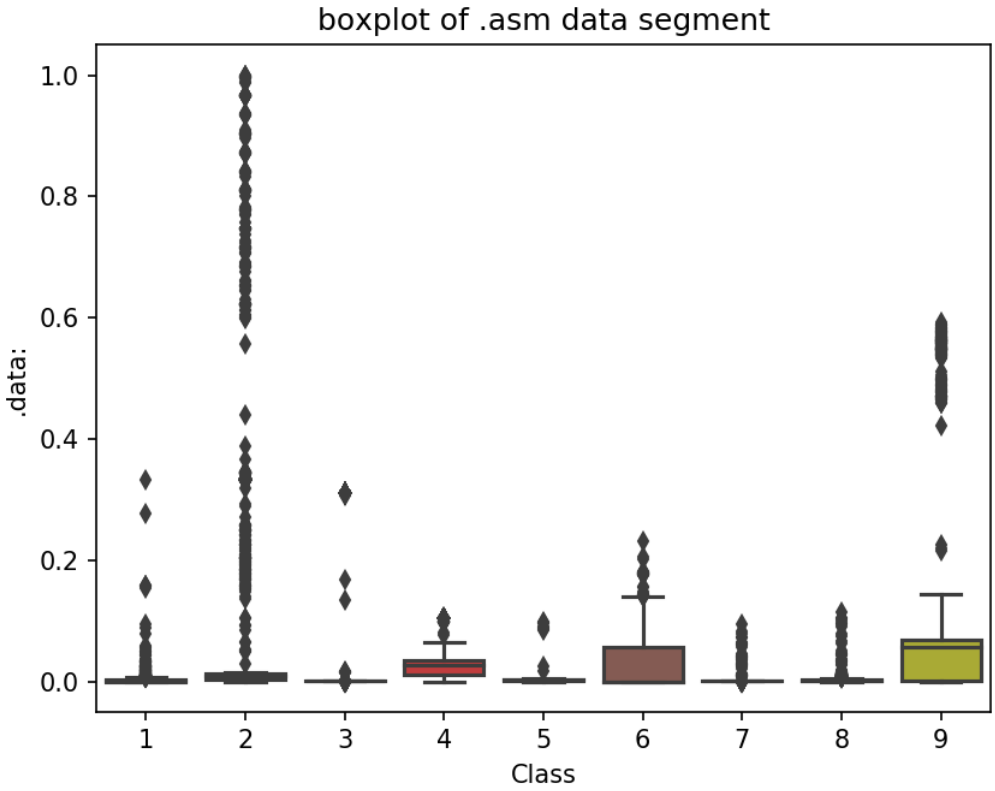
In [38]:

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



In [39]:

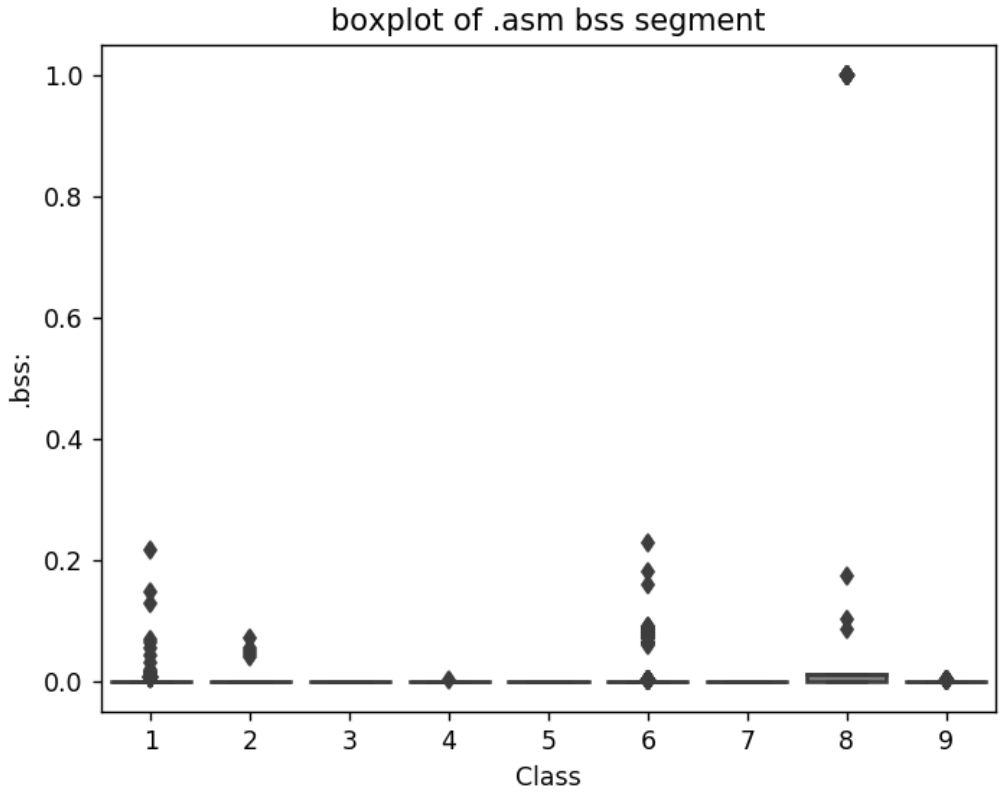
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



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

In [0]:

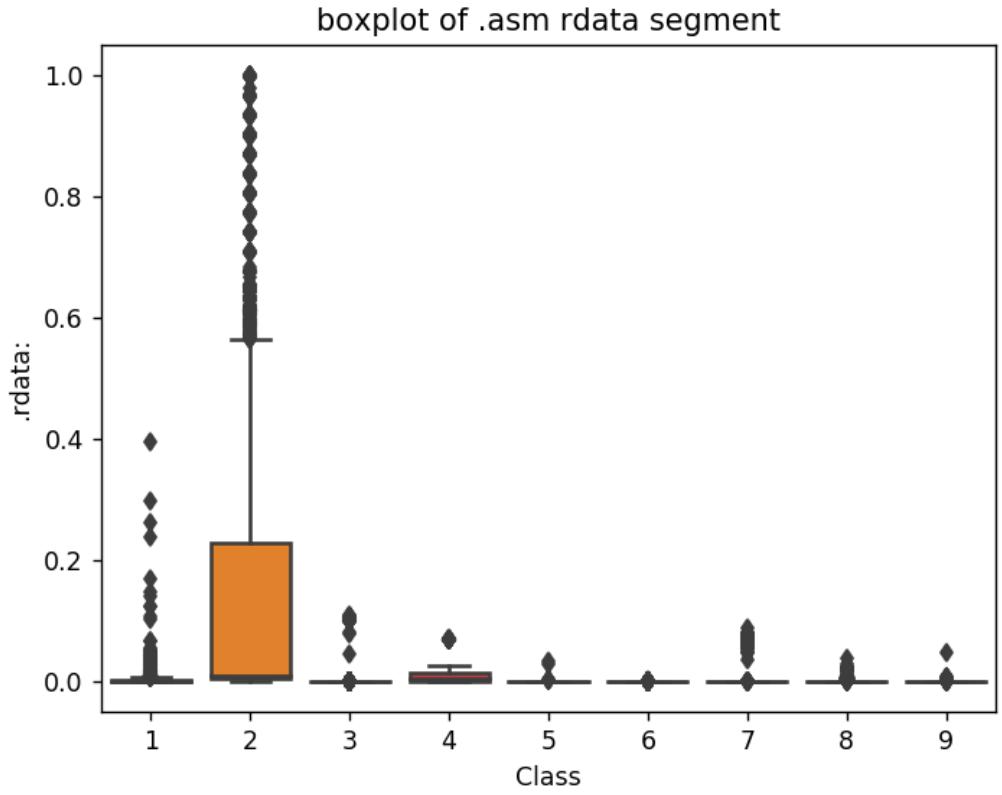
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [0]:

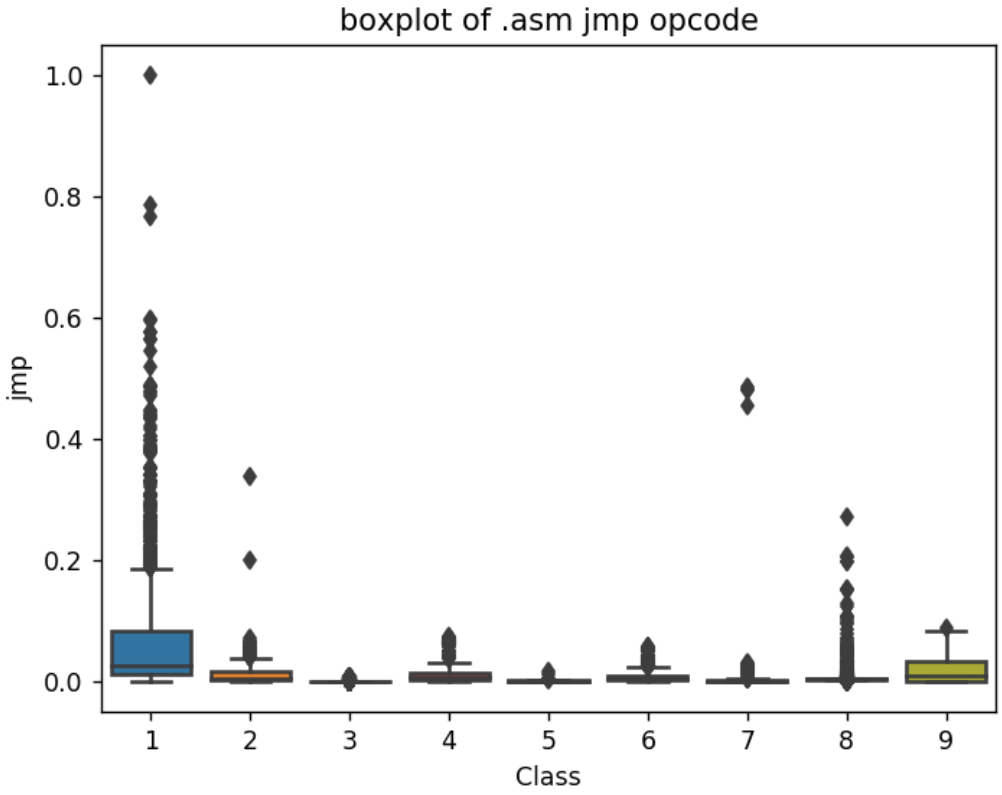
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

In [0]:

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

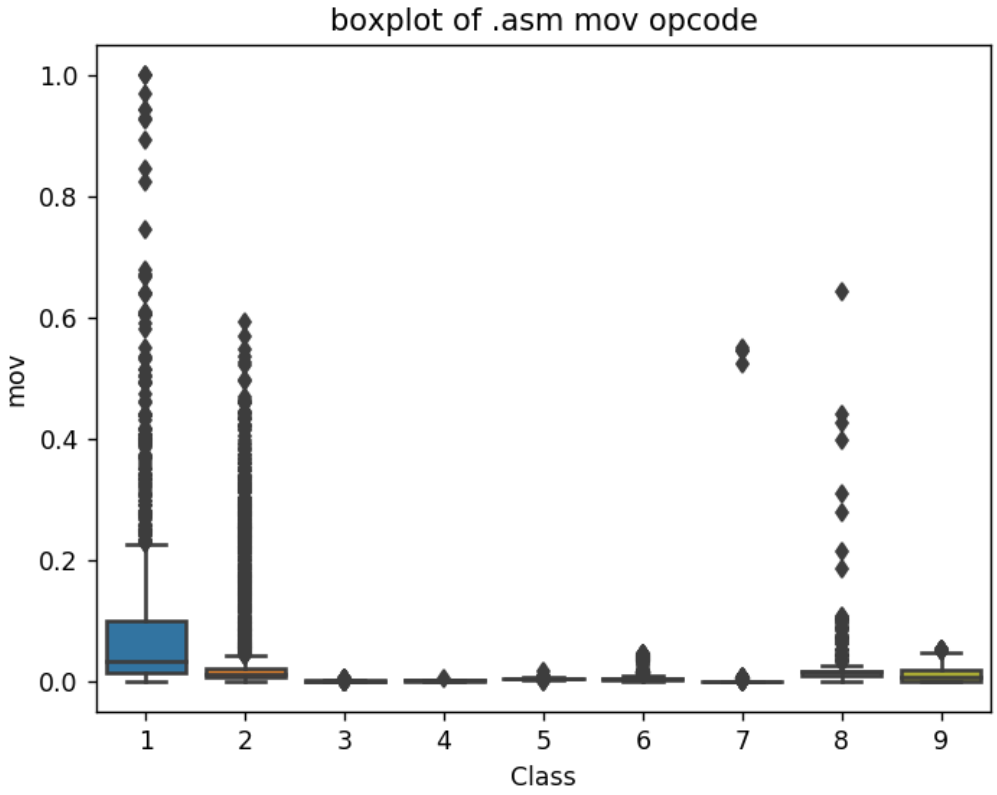


plot between jmp and Class label

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

In [0]:

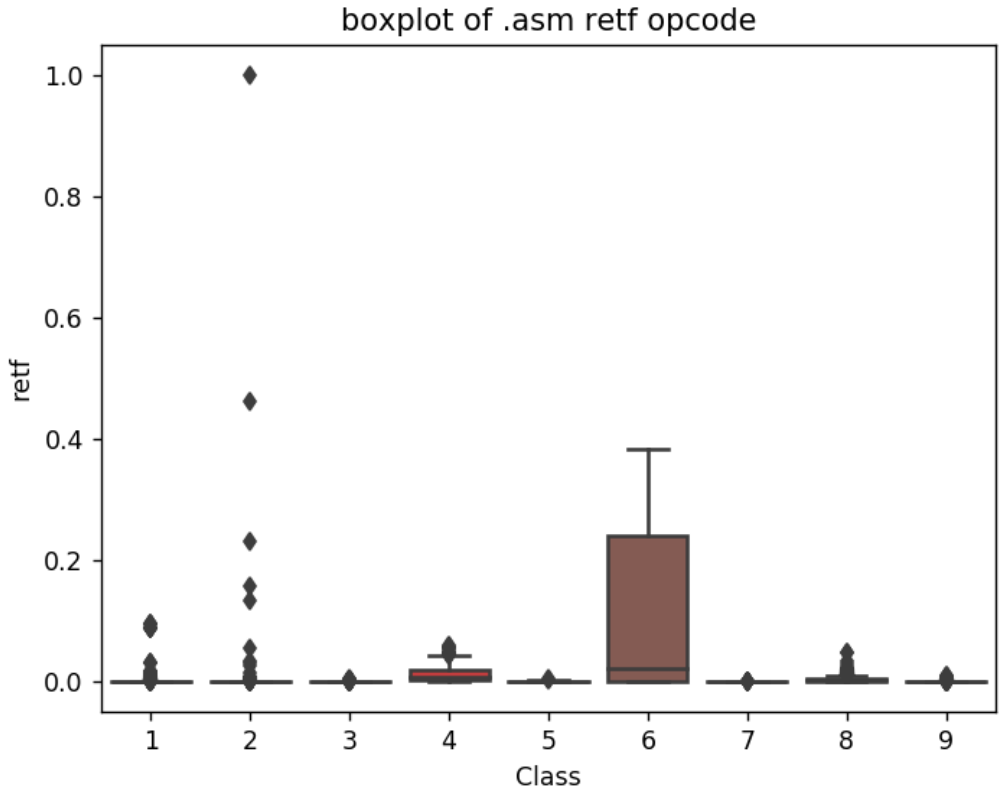
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



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

In [0]:

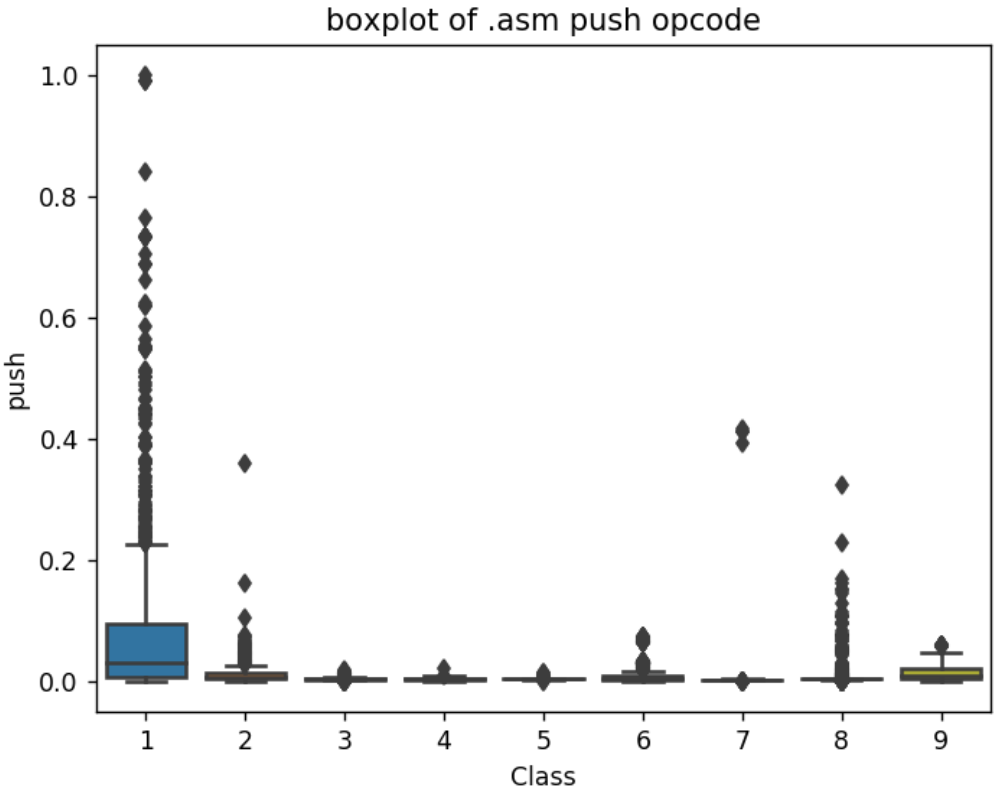
```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```



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

In [0]:

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

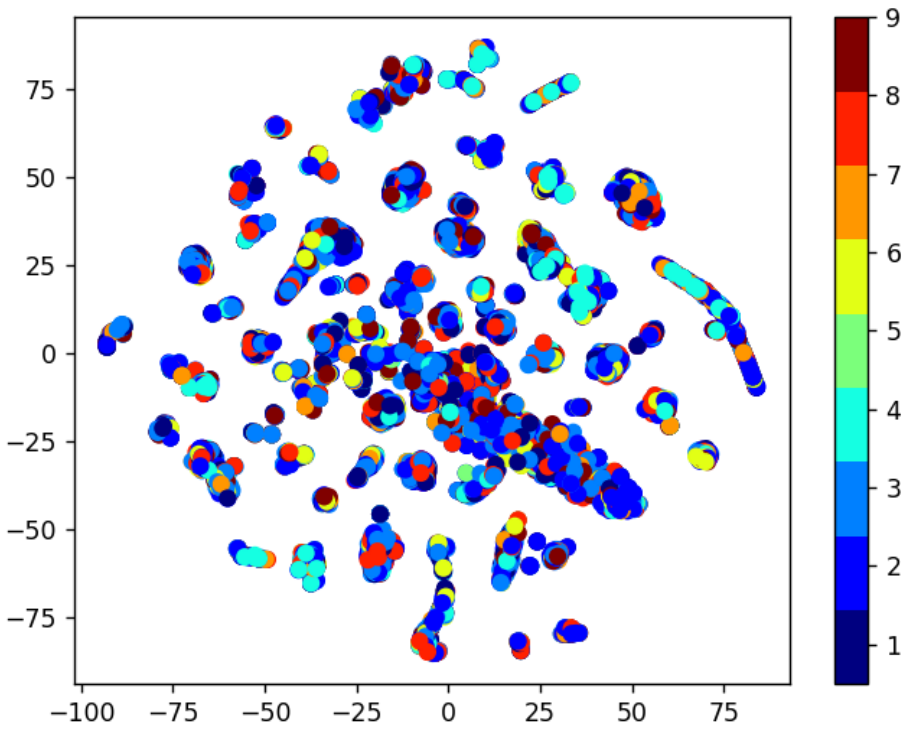


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

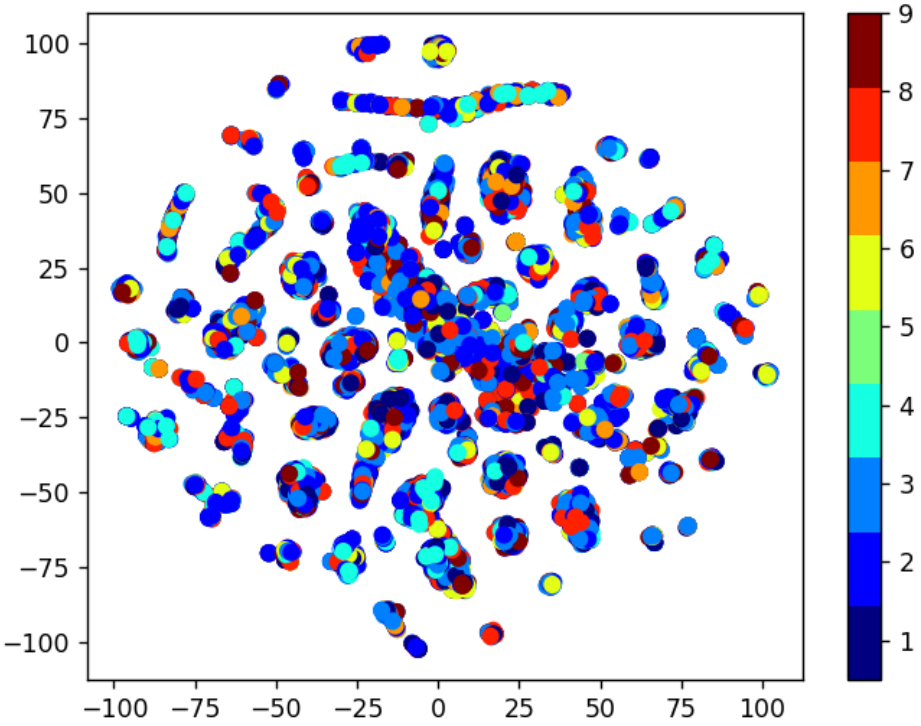
In [0]:

```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed
-stochastic-neighbourhood-embeddingt-sne-part-1/

#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()
```



In [0]:

```
# by univariate analysis on the .asm file features we are getting very negligible infor
mation from
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after r
emoving those features
# the plot looks very messy

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



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 [40]:

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

In [41]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,strati
fy=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_as
m,stratify=y_train_asm,test_size=0.20)
```

In [42]:

```
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

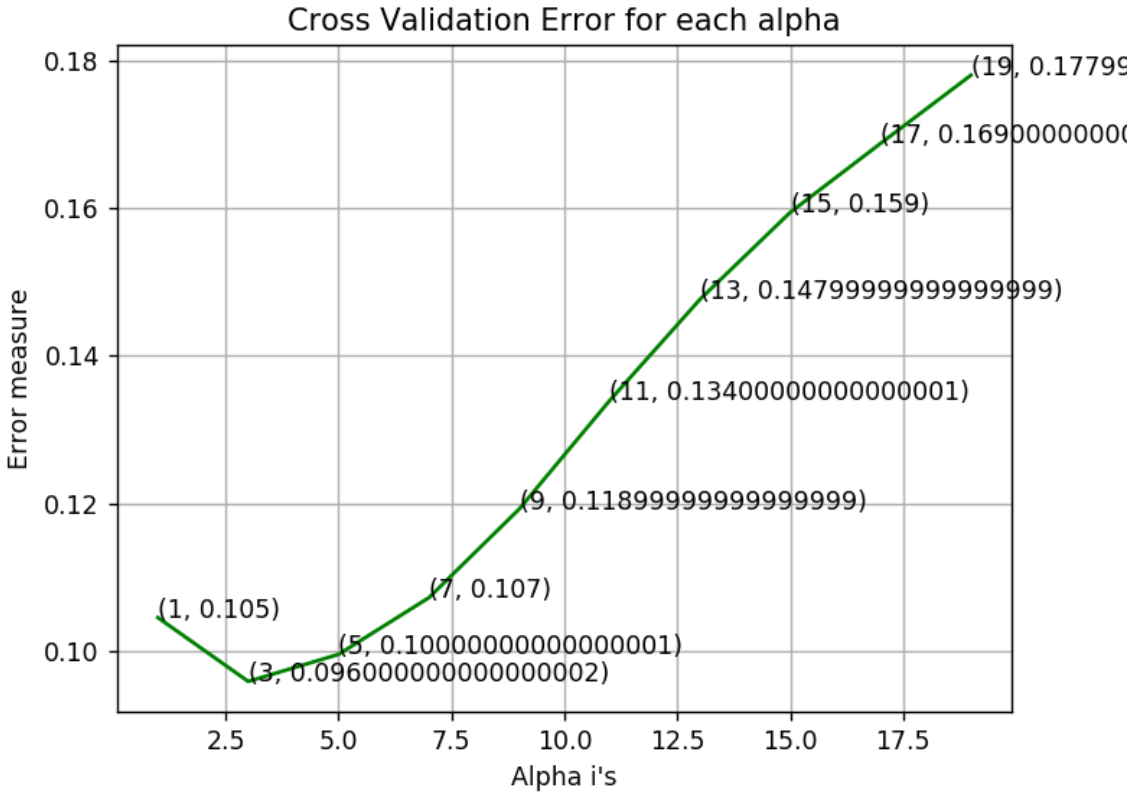
4.4.1 K-Nearest Neigbors

In [0]:

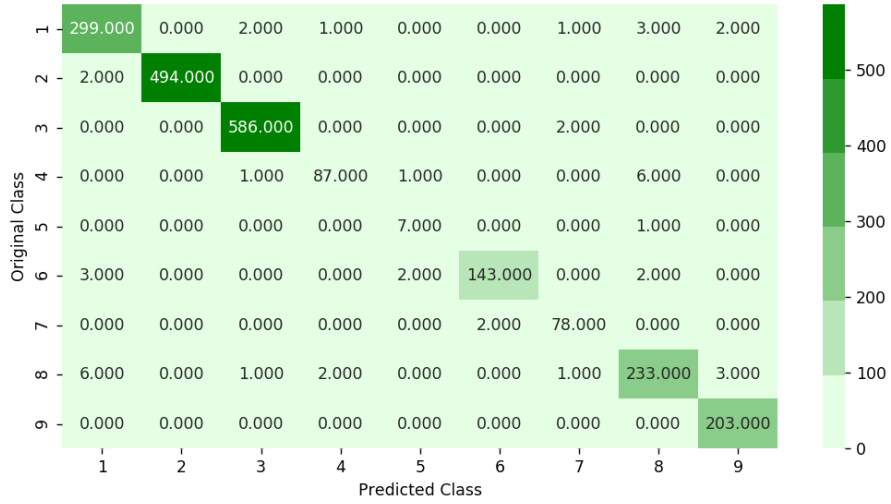
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
nerated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=3
0, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k
-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----------------------------------


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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

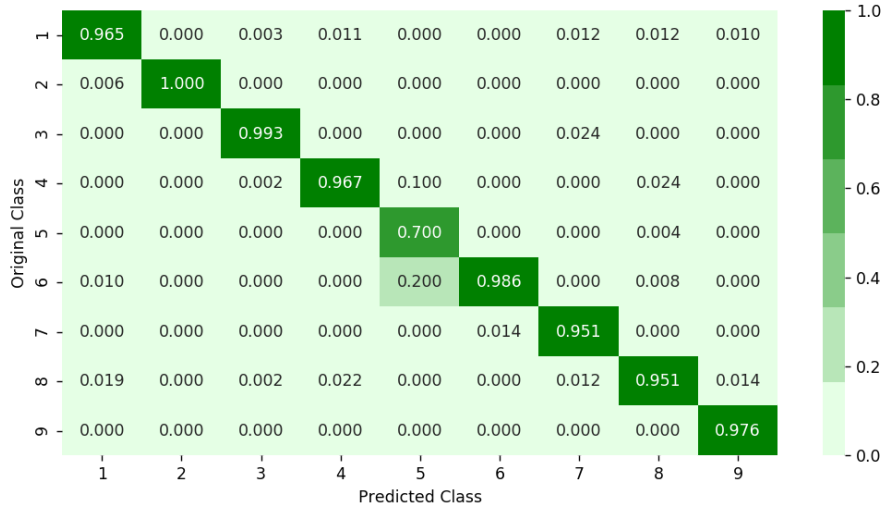
```
log_loss for k =  1 is 0.104531321344
log_loss for k =  3 is 0.0958800580948
log_loss for k =  5 is 0.0995466557335
log_loss for k =  7 is 0.107227274345
log_loss for k =  9 is 0.119239543547
log_loss for k =  11 is 0.133926642781
log_loss for k =  13 is 0.147643793967
log_loss for k =  15 is 0.159439699615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



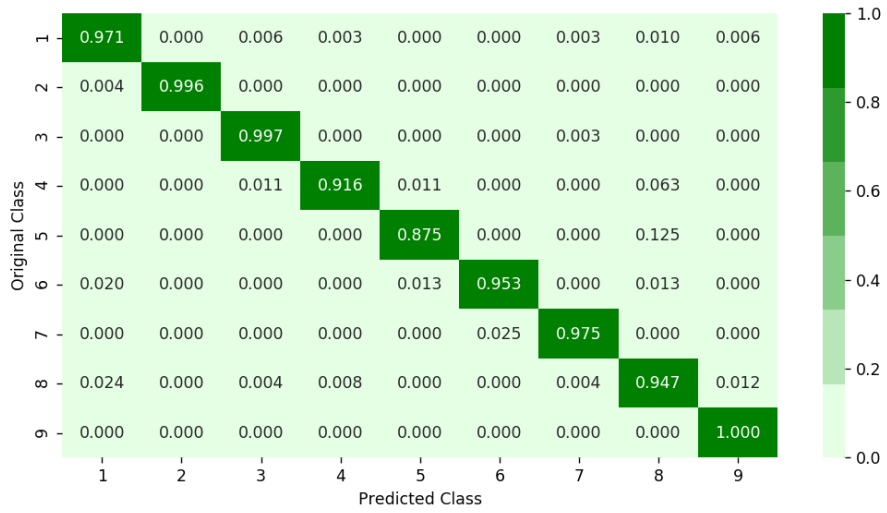

```
log loss for train data 0.0476773462198
log loss for cv data 0.0958800580948
log loss for test data 0.0894810720832
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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




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

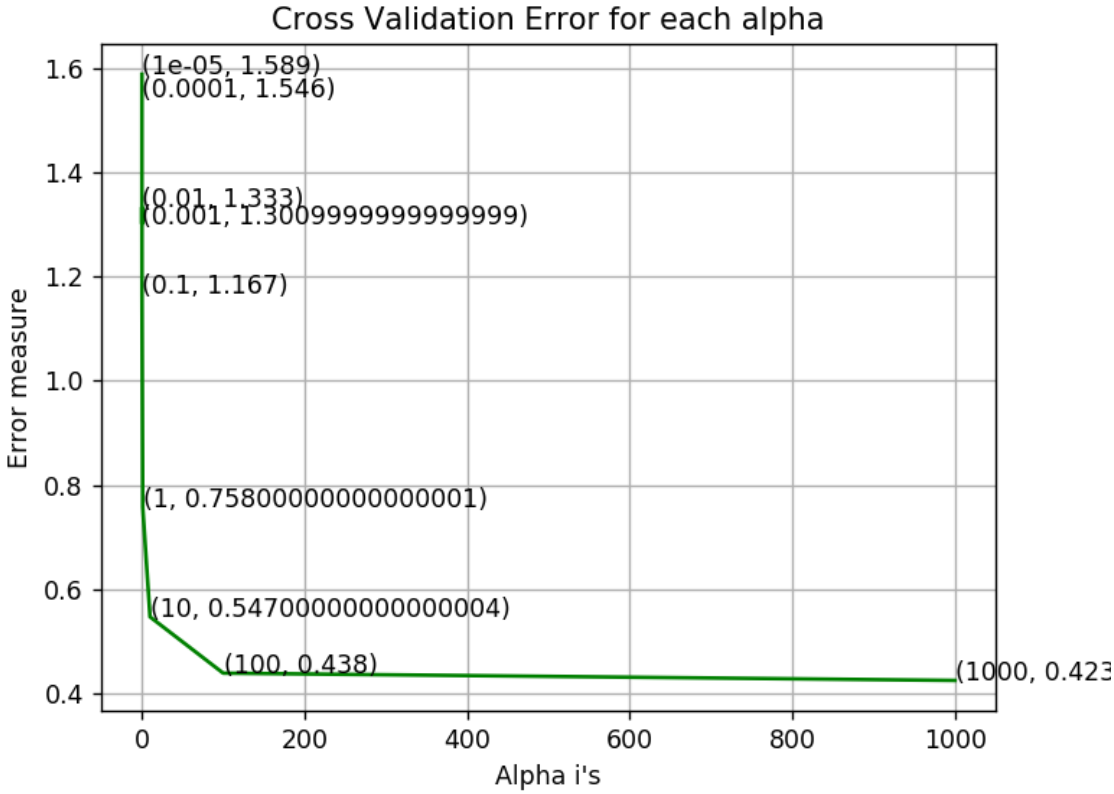
4.4.2 Logistic Regression

In [0]:

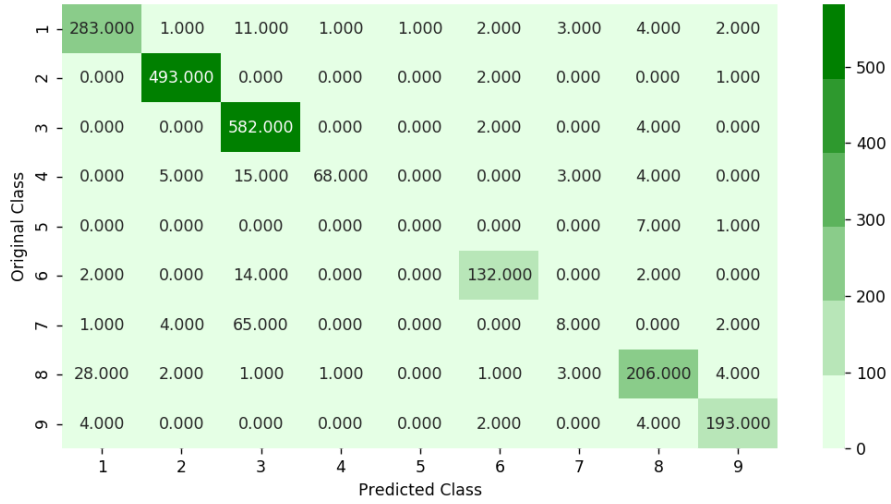
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

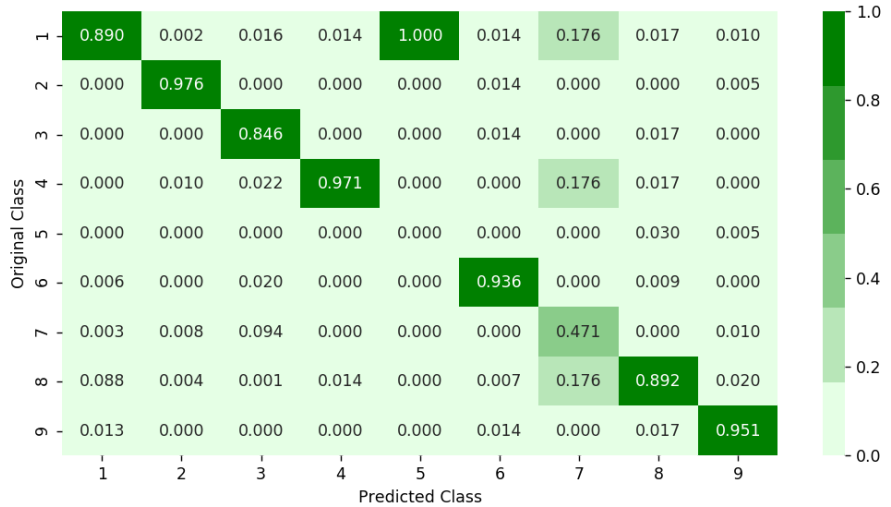
```
log_loss for c =  1e-05 is 1.58867274165
log_loss for c =  0.0001 is 1.54560797884
log_loss for c =  0.001 is 1.30137786807
log_loss for c =  0.01 is 1.33317456931
log_loss for c =  0.1 is 1.16705751378
log_loss for c =  1 is 0.757667807779
log_loss for c =  10 is 0.546533939819
log_loss for c =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



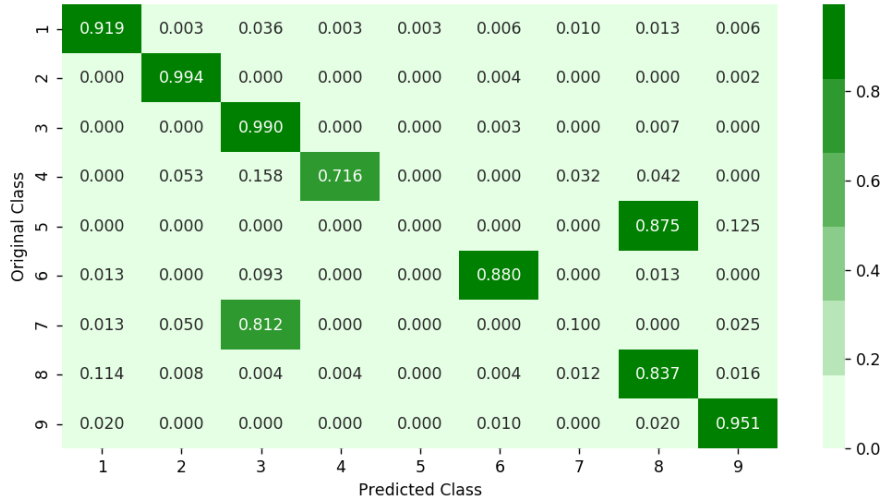

```
log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points  9.61361545538
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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




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

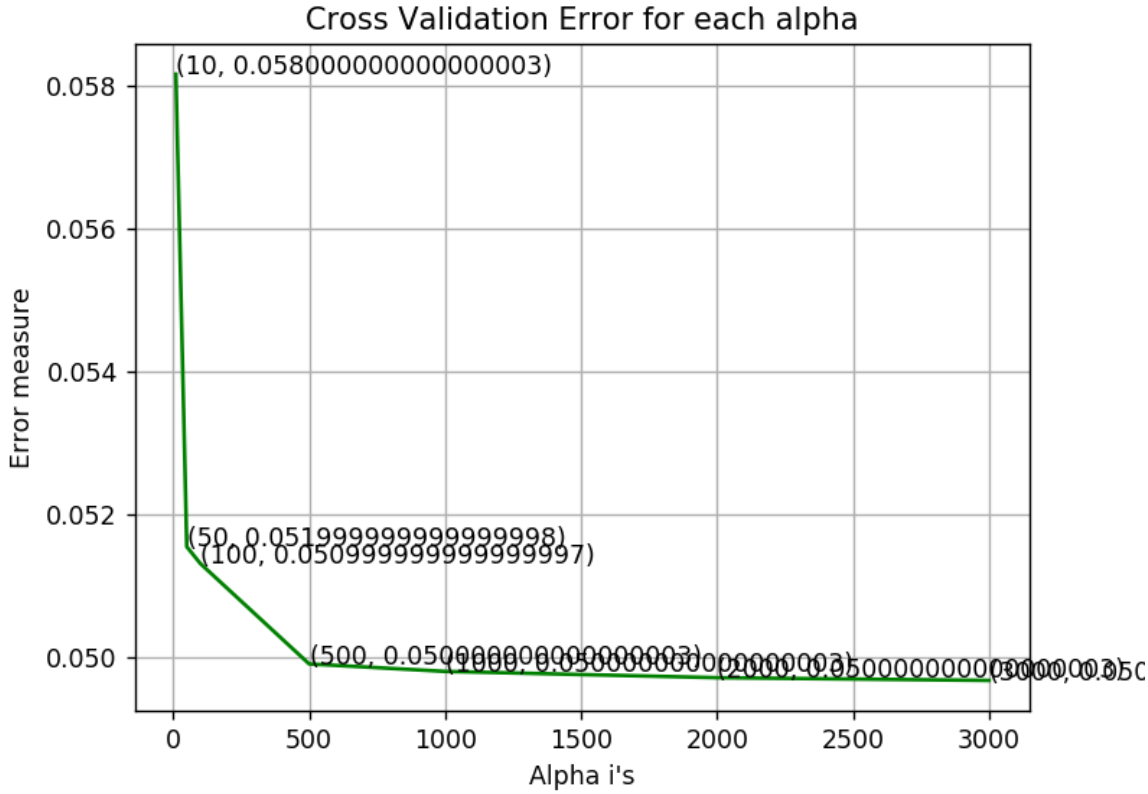
4.4.3 Random Forest Classifier

In [0]:

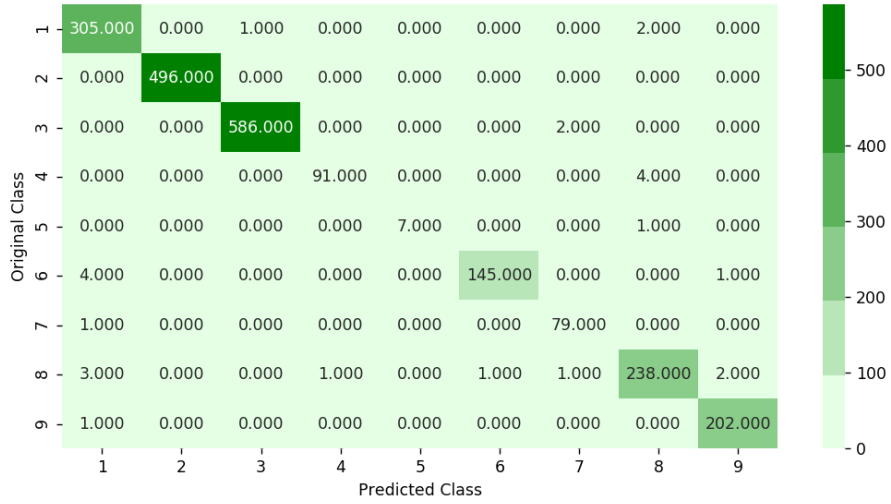
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
e, verbose=0, warm_start=False,
# class_weight=None)

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/r
andom-forest-and-their-construction-2/
# ------------------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, e
ps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes
_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

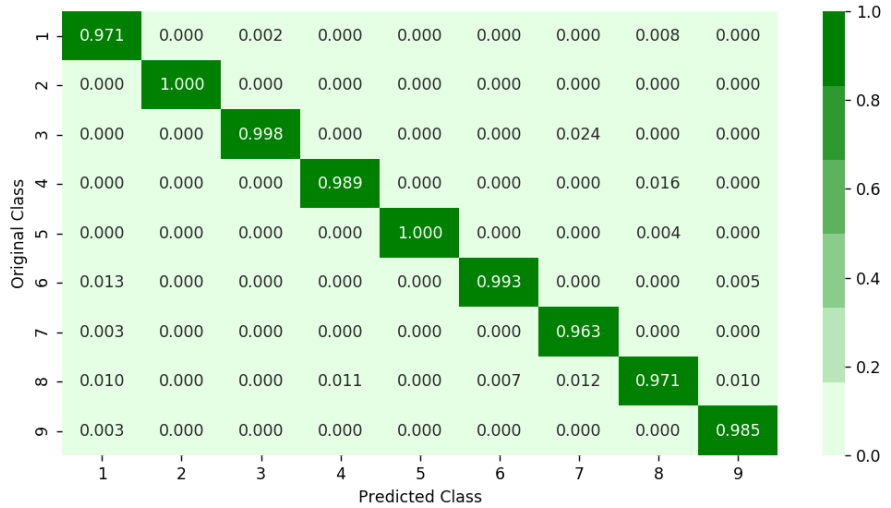
```
log_loss for c =  10 is 0.0581657906023
log_loss for c =  50 is 0.0515443148419
log_loss for c =  100 is 0.0513084973231
log_loss for c =  500 is 0.0499021761479
log_loss for c =  1000 is 0.0497972474298
log_loss for c =  2000 is 0.0497091690815
log_loss for c =  3000 is 0.0496706817633
```



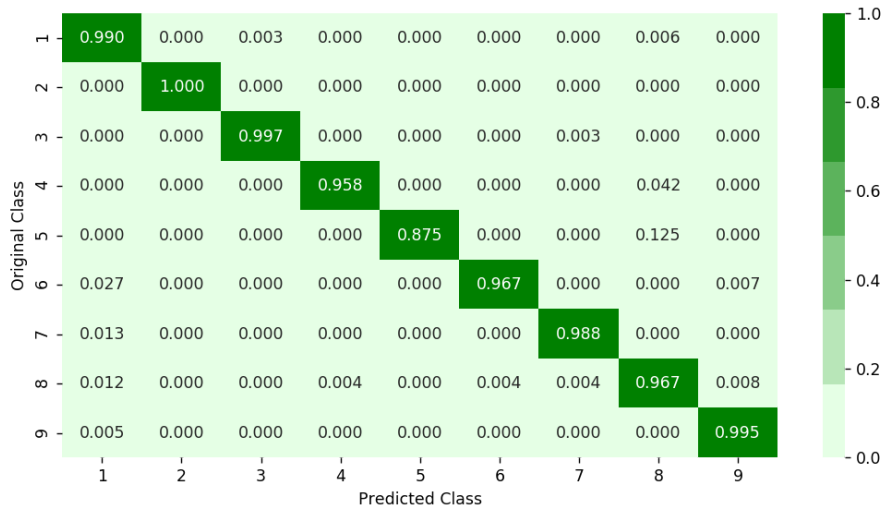

```
log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points  1.14995400184
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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




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

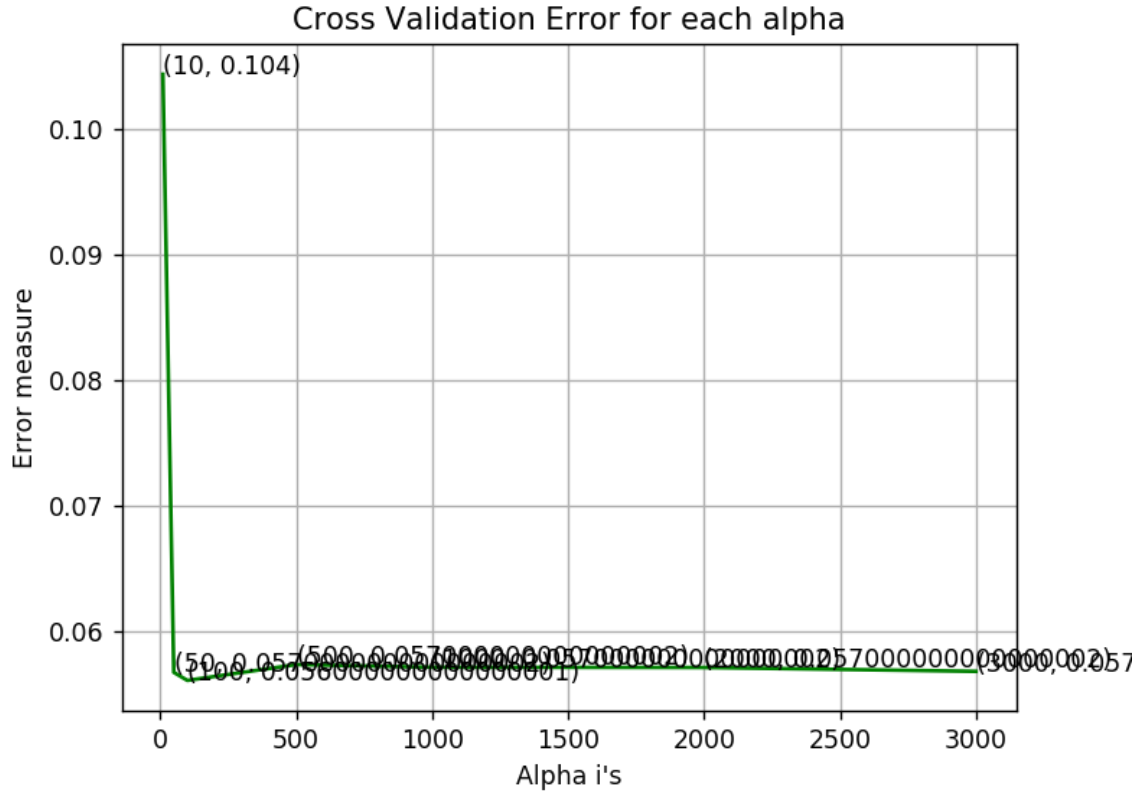
4.4.4 XgBoost Classifier

In [0]:

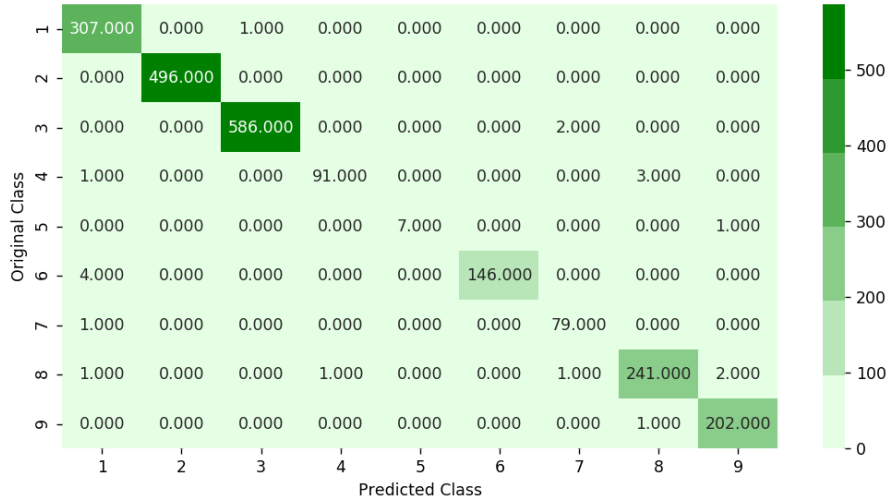
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
predict_y = sig_clf.predict_proba(X_train_asm)

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

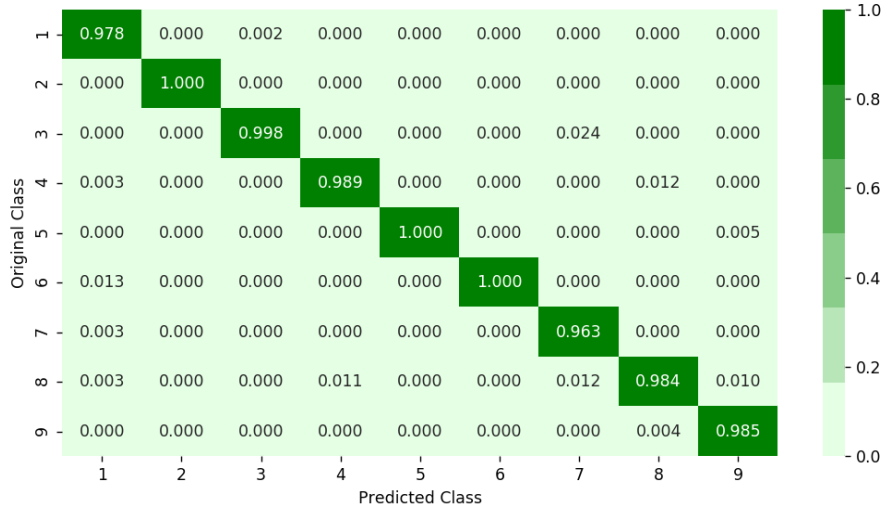
```
log_loss for c =  10 is 0.104344888454
log_loss for c =  50 is 0.0567190635611
log_loss for c =  100 is 0.056075038646
log_loss for c =  500 is 0.057336051683
log_loss for c =  1000 is 0.0571265109903
log_loss for c =  2000 is 0.057103406781
log_loss for c =  3000 is 0.0567993215778
```



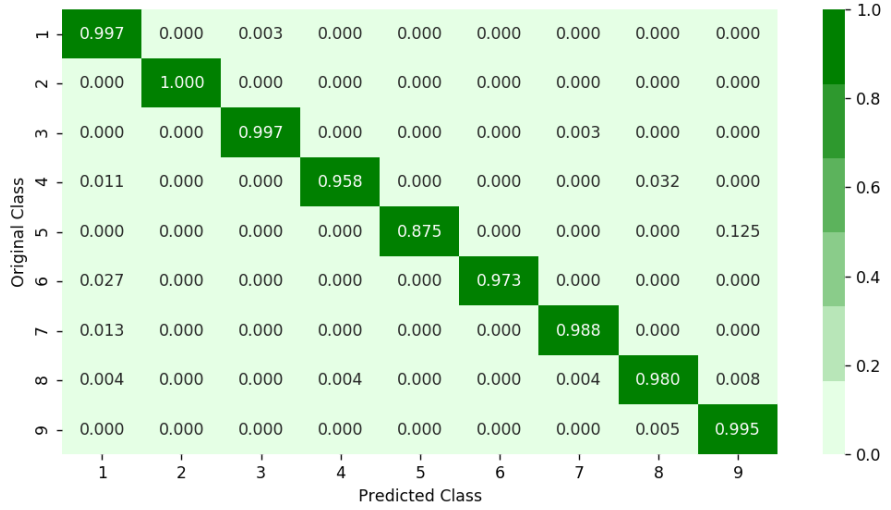

```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.056075
038646
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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



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

4.4.5 Xgboost Classifier with best hyperparameters

In [0]:

```
x_cfl=XGBClassifier()

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

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

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

Out[0]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, col
sample_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], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5,
1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

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

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

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

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

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

In [ ]:

```
conclusion_table.add_row(["asm_codes",log_loss(y_train_asm, predict_y),log_loss(y_test_asm, predict_y)])
```

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

4.5.1. Merging both asm and byte file features

In [11]:

```
result.head()
```

Out[11]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.0
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0

5 rows × 260 columns

In [12]:

```
result_asm.head()
```

Out[12]:

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

5 rows × 54 columns

In [45]:

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

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

In [16]:

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

Out[16]:

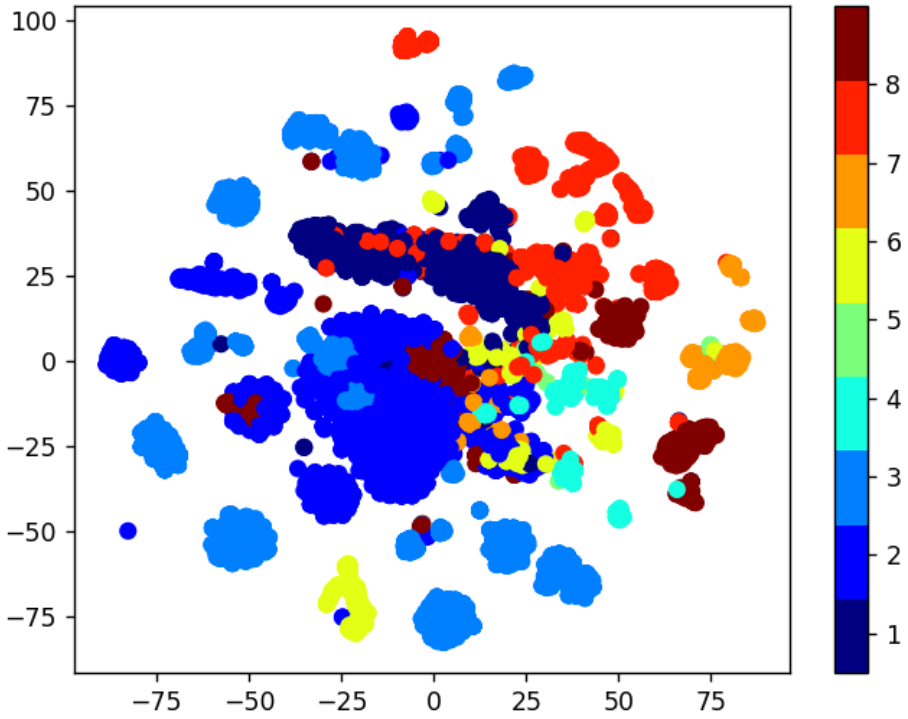
	0	1	2	3	4	5	6	7	8
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104
3	0.009209	0.001708	0.000404	0.000441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

5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

In [0]:

```
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()
```



4.5.3. Train and Test split

In [47]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stra
tify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_trai
n,stratify=y_train,test_size=0.20)
```

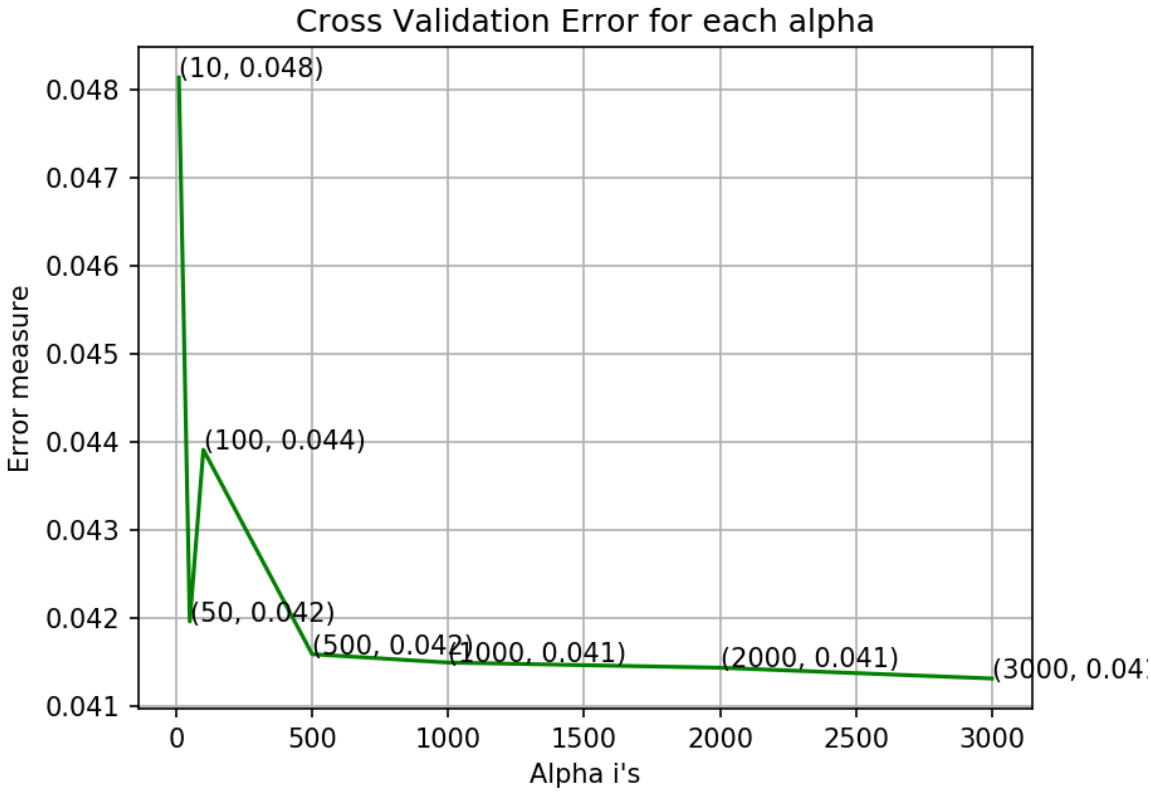
4.5.4. Random Forest Classifier on final features

In [48]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
e, verbose=0, warm_start=False,
# class_weight=None)

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/r
andom-forest-and-their-construction-2/
# ------------------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

predict_y = sig_clf.predict_proba(X_train_merge)
```

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

```
log_loss for c =  10 is 0.048139480270905616
log_loss for c =  50 is 0.041960520620466936
log_loss for c =  100 is 0.04391115781580027
log_loss for c =  500 is 0.04158910863116275
log_loss for c =  1000 is 0.04149395820070371
log_loss for c =  2000 is 0.041435191169850796
log_loss for c =  3000 is 0.041312931204471756
```




```
For values of best alpha =  3000 The train log loss is: 0.0148531857227308
38
For values of best alpha =  3000 The cross validation log loss is: 0.04131
2931204471756
For values of best alpha =  3000 The test log loss is: 0.04652190496317460
5
```

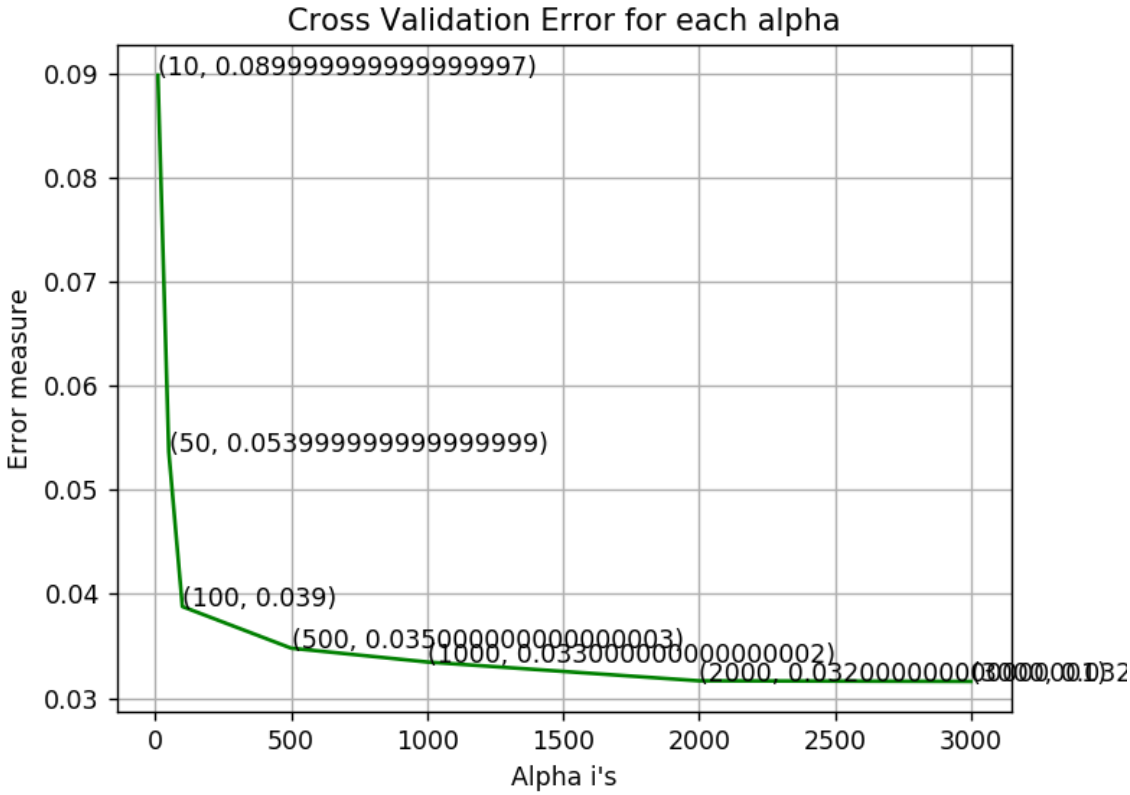
4.5.5. XgBoost Classifier on final features

In [0]:

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
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
```




```
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: 0.03159
72694477
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 [0]:

```
x_cfl=XGBClassifier()

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

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

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

Out[0]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, col
sample_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], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5,
1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

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

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

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

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

In [44]:

```
conclusion_table.add_row(["bytesbow_asmcodes_merged",log_loss(y_train_merge, predict_y),log_loss(y_test_merge, predict_y)])
```

5. Assignments

1. **Add bi-grams and n-gram features on byte files and improve the log-loss**
2. **Using the 'dchad' github account (https://github.com/dchad/malware-detection), decrease the logloss to <=0.01**
3. **Watch the video (https://www.youtube.com/watch?v=VLQTRILGz5Y) that was in reference section and implement the image features to improve the logloss**

https://pdfs.semanticscholar.org/8db2/69106ea6e1f59e4
(https://pdfs.semanticscholar.org/8db2/69106ea6e1f59e

http://www.cs.sjsu.edu/faculty/stamp/papers/topics/topi
s11416-012-0175-y.pdf
(http://www.cs.sjsu.edu/faculty/stamp/papers/topics/top
s11416-012-0175-y.pdf)

https://www.quora.com/What-is-the-difference-between-opcode-operand-and-instruction
(https://www.quora.com/What-is-the-difference-between-opcode-operand-and-instruction)

graph similarity features
https://sites.cs.ucsb.edu/~xyan/papers/tods06_similarit
(https://sites.cs.ucsb.edu/~xyan/papers/tods06_similari

Construction Call_graphs for each asm file using networkx

In [ ]:

```
def create_call_graph_from(source,filename):

    graph = nx.DiGraph()
    node = '.program_entry_point' # if first we find is an opcaode instead of function
 starting then its start from main()
    call_opcodes = ['call','int']
    call_blocks = ['sub_']

    with codecs.open(source+filename,encoding='cp1252',errors ='replace') as lines:
        for row in lines:
            row = row.rstrip('\r\n')  # get rid of newlines they are annoying.
            if ';' in row:
                row = row.split(';')[0] # get rid of comments they are annoying.
                #print(row)

            # get rid of all these things they are annoying.
            row = row.replace('short','').replace('ds:',' ')
            row = row.replace('dword','').replace('near','')
            row = row.replace('ptr','').replace(':',' ').replace(',',' ') #.replace
('??',' ')
            row = row.replace('@','').replace('?','')
            parts = row.split() # tokenize code line

            if (len(parts) < 4): # this is just a comment line
                continue

            if (parts[3] == 'endp'): # ignore subroutine end labels
                continue


            #check for subroutines and block labels
            #block and subroutine labels are always after the .text HHHHHHHH relative a
ddress
            for block in call_blocks:
                token = parts[2]
                idx = token.find(block)
                if ((idx == 0) or (parts[3] == 'proc')):# found a sub_ created
                    # we do not take exteranl routines bcuz they do work but dont call
 local routines
                    # add new node to the graph, we are now in a new subroutine
                    node = token
                    graph.add_node(node)
                    #print("Node: " + node)
                    break # to skip other call_blocks if found


            # now check for edge opcode
            for opcode in call_opcodes: # check the line for a new edge
                if opcode in parts:#found a edge to local or external fundtion
                    # Extract desination address/function name/interrupt number as the
 directed edge.
                    idx = parts.index(opcode)# gives index of call or int
                    if ((idx + 1) < len(parts)): # in a few ASM files there is no opera
nd, disassembly error?
                        next_node = parts[idx + 1]# gives the rountine name either loca
l or external
                    else:
                        next_node = "none"#no address/function name/interrupt number af
ter call or int means disassembly error
```

```
                    graph.add_edge(node, next_node)
                    #print("Edge: " + node + " " + next_node)

                    break#to skip other opcodes if found

    return graph
```

In [13]:

```
import random
from tqdm import tqdm_notebook as tqdm
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='train/asmFiles_spilt/first'
folder_2 ='train/asmFiles_spilt/second'
folder_3 ='train/asmFiles_spilt/third'
folder_4 ='train/asmFiles_spilt/fourth'
folder_5 ='train/asmFiles_spilt/fifth'
folder_6 = 'train/asmFiles_spilt/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/asmFiles/'
files = os.listdir('train/asmFiles/')

data=list(range(0,6165))
random.shuffle(data)
for i in tqdm(range(0,6165)):
    if i % 5==0:
#         print(i)
        shutil.move(source+files[data[i]],folder_1)
    elif i%5==1:
        shutil.move(source+files[data[i]],folder_2)
    elif i%5 ==2:
        shutil.move(source+files[data[i]],folder_3)
    elif i%5 ==3:
        shutil.move(source+files[data[i]],folder_4)
    elif i%5==4:
        shutil.move(source+files[data[i]],folder_5)
```

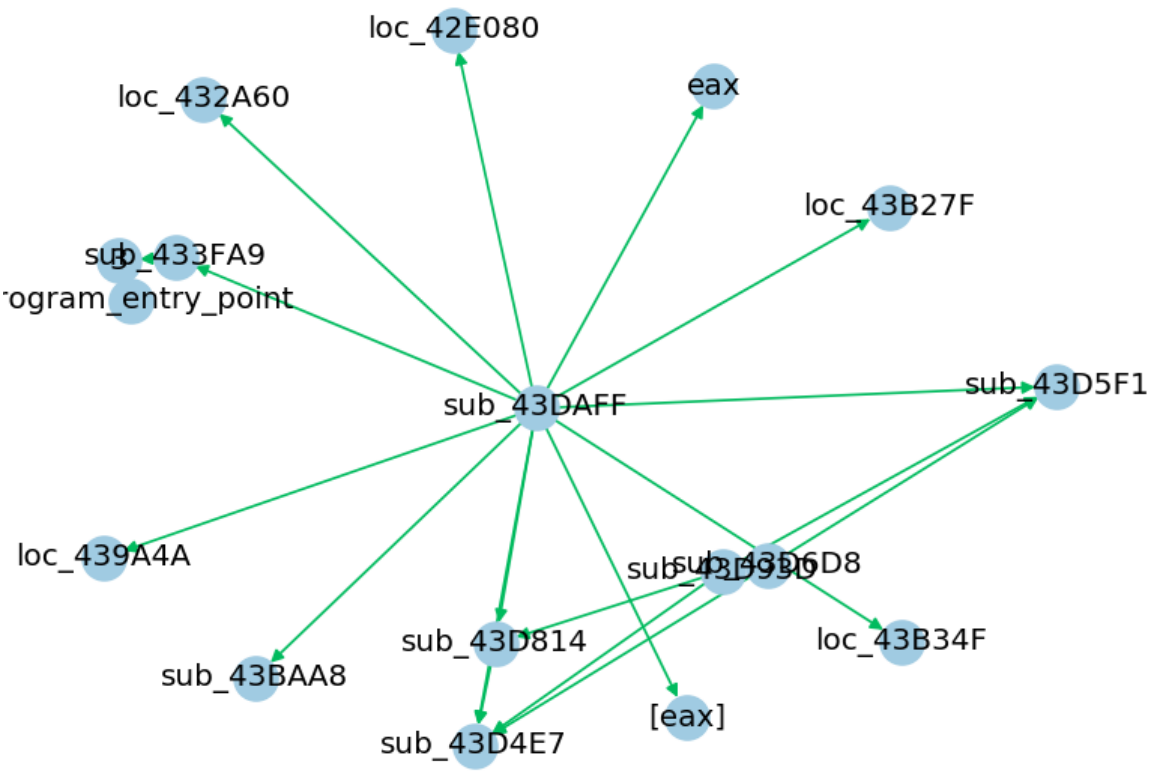
In [ ]:

```
def firstprocess():
    source='train/asmFiles_spilt/first/'
    files = os.listdir(source)
    for filename in tqdm(files):
        nx.write_gpickle(create_call_graph_from(source,filename), "train/asmGraphs/{}.g
pickle".format(filename.split('.')[0]))

def secondprocess():
    source='train/asmFiles_spilt/second/'
    files = os.listdir(source)
    for filename in tqdm(files):
        nx.write_gpickle(create_call_graph_from(source,filename), "train/asmGraphs/{}.g
pickle".format(filename.split('.')[0]))

def thirdprocess():
    source='train/asmFiles_spilt/third/'
    files = os.listdir(source)
    for filename in tqdm(files):
        nx.write_gpickle(create_call_graph_from(source,filename), "train/asmGraphs/{}.g
pickle".format(filename.split('.')[0]))

def fourthprocess():
    source='train/asmFiles_spilt/fourth/'
    files = os.listdir(source)
    for filename in tqdm(files):
        nx.write_gpickle(create_call_graph_from(source,filename), "train/asmGraphs/{}.g
pickle".format(filename.split('.')[0]))

def fifthprocess():
    source='train/asmFiles_spilt/fifth/'
    files = os.listdir(source)
    for filename in tqdm(files):
        nx.write_gpickle(create_call_graph_from(source,filename), "train/asmGraphs/{}.g
pickle".format(filename.split('.')[0]))

def main():
    if not os.path.isdir('train/asmGraphs'):
        os.makedirs('train/asmGraphs')
    #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()
```

```
    return None

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

In [381]:

```
source='train/asmGraphs/jNUAeECmnsxGJlQrTubv.gpickle'
G = nx.read_gpickle(source)
pos=nx.spring_layout(G)
nx.draw(G,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm.Blues,
with_labels=True)
plt.show()
print(nx.info(G))
```




```
Name:
Type: DiGraph
Number of nodes: 17
Number of edges: 23
Average in degree:   1.3529
Average out degree:   1.3529
```

construction of in graph related features

In [3]:

```
all_features = []
source='train/asmGraphs/'
files = os.listdir(source)
index = 0
for filename in tqdm(files):
    G = nx.read_gpickle(source+filename)

    # adding features
    features = []
    features.append(filename.split('.')[0])

    # no. od nodes
    features.append(G.number_of_nodes())

    # no. of features
    features.append(G.number_of_edges())

    # avg. in_degree
    try:
        features.append(round(sum([count[1] for count in G.out_degree()])/len(G.out_deg
ree()),4))
    except:
        features.append(0)

    # avg. out_degree
    try:
        features.append(round(sum([count[1] for count in G.in_degree()])/len(G.in_degre
e()),4))
    except:
        features.append(0)

    # if contains an entery point or not
    features.append((lambda: 0, lambda: 1)[G.__contains__('.program_entry_point')]())

    # max. out_degree
    features.append(max([count[1] for count in G.out_degree()],default=0))

    # max. in_degree
    features.append(max([count[1] for count in G.in_degree()],default=0))

    # no. of weakly and strongly connected features
    features.append(len(list(nx.weakly_connected_components(G))))
    features.append(len(list(nx.weakly_connected_components(G))))

    # kartz centrality for all nodes
    try:
        katz = nx.katz.katz_centrality(G,alpha=0.005,beta=1)

        # min,max,mean kartz cen. in a graph
        features.append(katz[min(katz, key=katz.get)])
        features.append(katz[max(katz, key=katz.get)])
        features.append(float(sum(katz.values())) / len(katz))

    except:
        features.append(0)
        features.append(0)
        features.append(0)

    # Hits score for all nodes
```

```
    try:
        hits = nx.hits(G, max_iter=1000, tol=1e-05, nstart=None, normalized=True)

        # max,mean  Hubs score in a graph
        features.append(hits[0][max(hits[0], key=hits[0].get)])
        features.append(float(sum(hits[0].values())) / len(hits[0]))

        # max,mean  authorities score in a graph
        features.append(hits[1][max(hits[1], key=hits[1].get)])
        features.append(float(sum(hits[1].values())) / len(hits[1]))

    except:
        features.append(0)
        features.append(0)
        features.append(0)
        features.append(0)

    # Page ranking for all nodes
    try:
        pr = nx.pagerank(G, alpha=0.85)

        # max,mean  Hits score in a graph
        features.append(pr[min(pr, key=pr.get)])
        features.append(pr[max(pr, key=pr.get)])
        features.append(float(sum(pr.values())) / len(pr))

    except:
        features.append(0)
        features.append(0)
        features.append(0)


    all_features.append(features)
```

In [23]:

```
# feature space
feature_names = ['ID', 'nodes', 'edges', 'avg_in_degree', 'avg_out_degree','has_program
_entry_point',
'max_out_degree','max_in_degree','no_weakly_connected','no_strongly_connected','min_kat
z','max_katz',
                 'mean_katz','max_hub','mean_hub','max_authorities','mean_authorities',
'min_rank','max_rank',
                 'mean_rank']
result_asm_graph = pd.DataFrame(all_features,columns=feature_names)
```

In [133]:

```
result_asm_graph = pd.merge(result_asm_graph,Y,on='ID', how='left')
result_asm_graph.head()
```

Out[133]:

	ID	has_program_entry_point	nodes	edges	avg_in_degree	a
0	01azqd4InC7m9JpocGv5	0	0.010336	0.047084	0.769626	
1	01IsoiSMh5gxyDYTI4CB	0	0.008469	0.007154	0.142723	
2	01jsnpXSAlgw6aPeDxrU	0	0.001600	0.002472	0.260980	
3	01kcPWA9K2BOxQeS5Rju	0	0.002734	0.001461	0.090271	
4	01SuzwMJEIXsK7A8dQbl	0	0.006402	0.004795	0.126533	

5 rows × 22 columns

In [171]:

```
# to csv
result_asm_graph.to_csv('asm_graph_features.csv',index=False)
```

In [14]:

```
# read from csv
result_asm_graph = pd.read_csv('asm_graph_features.csv')
result_asm_graph.head()
```

Out[14]:

	ID	has_program_entry_point	nodes	edges	avg_in_degree	a
0	01azqd4InC7m9JpocGv5	0	0.010336	0.047084	0.769626	
1	01IsoiSMh5gxyDYTI4CB	0	0.008469	0.007154	0.142723	
2	01jsnpXSAlgw6aPeDxrU	0	0.001600	0.002472	0.260980	
3	01kcPWA9K2BOxQeS5Rju	0	0.002734	0.001461	0.090271	
4	01SuzwMJEIXsK7A8dQbl	0	0.006402	0.004795	0.126533	

5 rows × 21 columns

In [15]:

```
# Normalize
Id = result_asm_graph['ID']
has_program_entry_point = result_asm_graph['has_program_entry_point']
result_asm_graph = result_asm_graph.drop(['ID','has_program_entry_point'], axis=1)
result_asm_graph = normalize(result_asm_graph)
result_asm_graph = pd.concat([Id,has_program_entry_point,result_asm_graph], axis=1)
result_asm_graph.head()
```

Out[15]:

	ID	has_program_entry_point	nodes	edges	avg_in_degree	a
0	01azqd4InC7m9JpocGv5	0	0.010336	0.047084	0.769626	
1	01IsoiSMh5gxyDYTl4CB	0	0.008469	0.007154	0.142723	
2	01jsnpXSAlgw6aPeDxrU	0	0.001600	0.002472	0.260980	
3	01kcPWA9K2BOxQeS5Rju	0	0.002734	0.001461	0.090271	
4	01SuzwMJEIXsK7A8dQbl	0	0.006402	0.004795	0.126533	

5 rows × 21 columns

In [16]:

```
result_asm_graph.isnull().all()
```

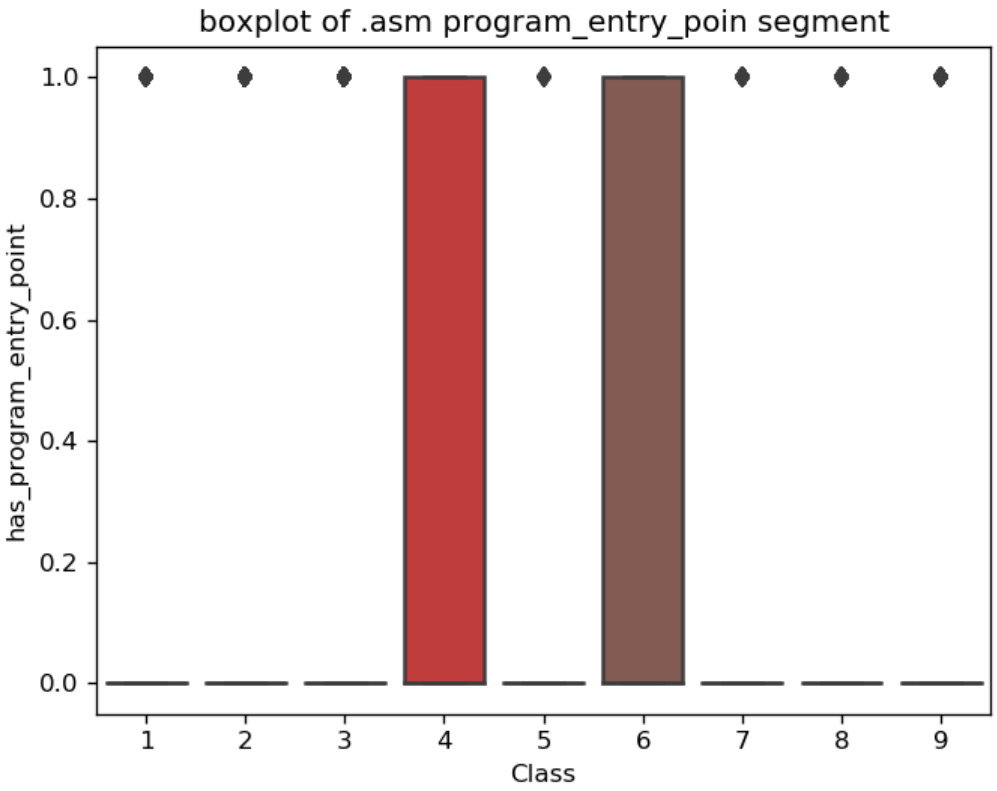
Out[16]:

```
ID                        False
has_program_entry_point   False
nodes                     False
edges                     False
avg_in_degree             False
avg_out_degree            False
max_out_degree            False
max_in_degree             False
no_weakly_connected       False
no_strongly_connected     False
min_katz                  False
max_katz                  False
mean_katz                 False
max_hub                   False
mean_hub                  False
max_authorities           False
mean_authorities          False
min_rank                  False
max_rank                  False
mean_rank                 False
Class                     False
dtype: bool
```

Univariate analysis on asm Graph features

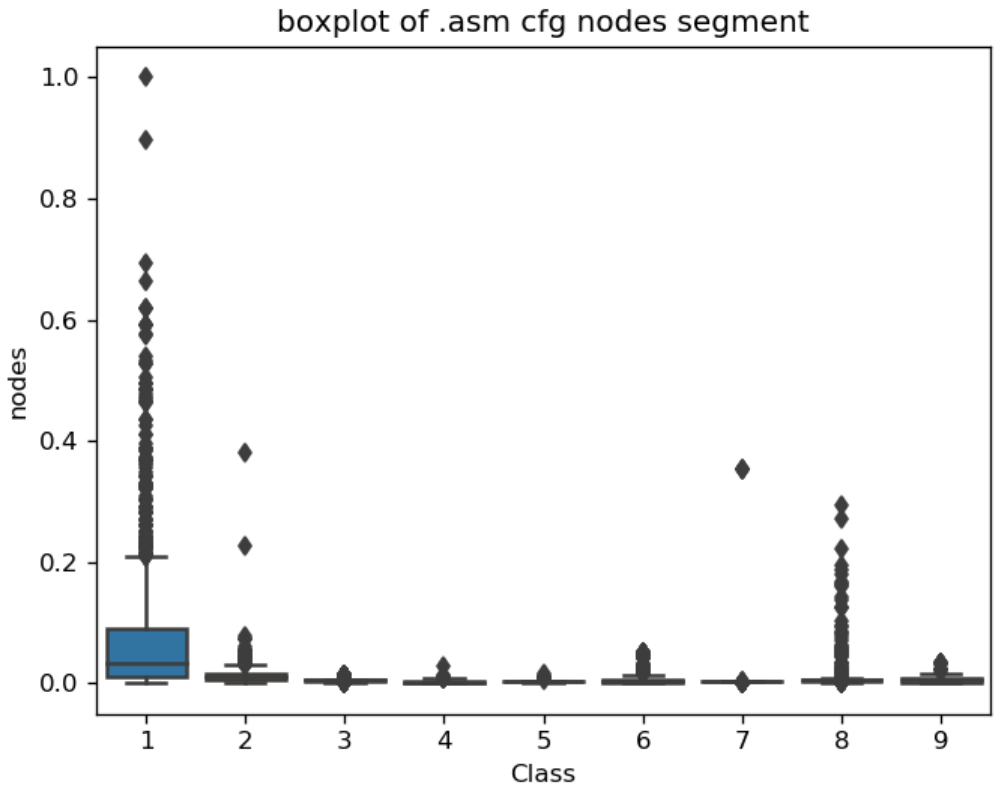
In [28]:

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



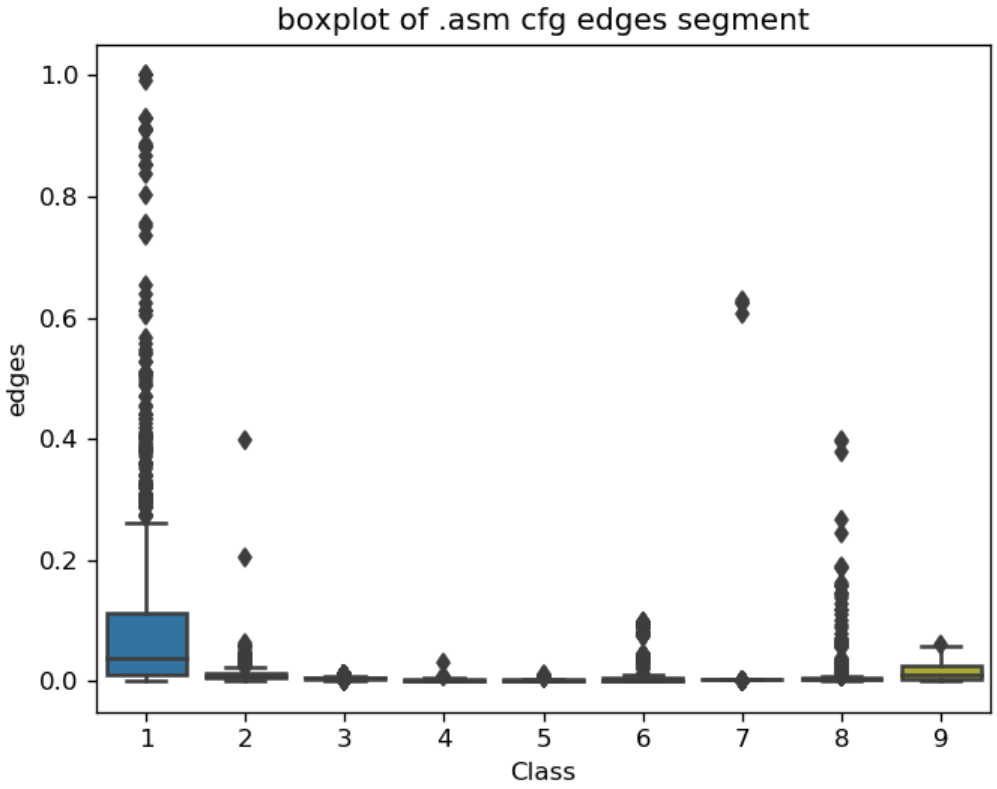
In [29]:

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



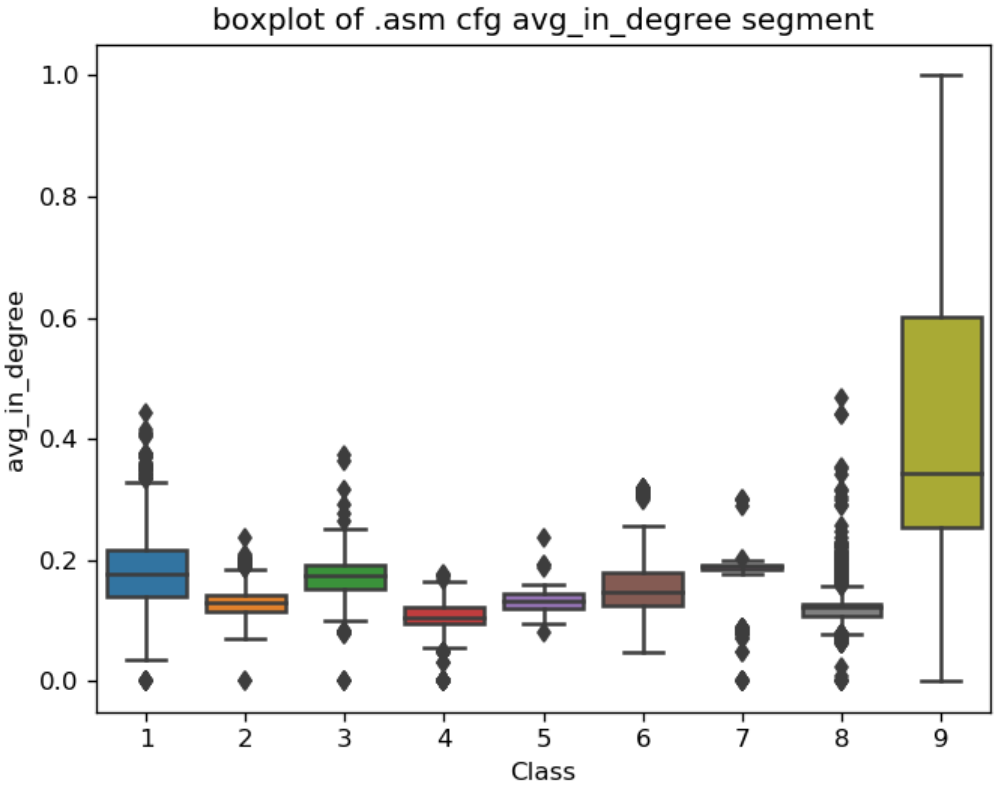
In [30]:

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



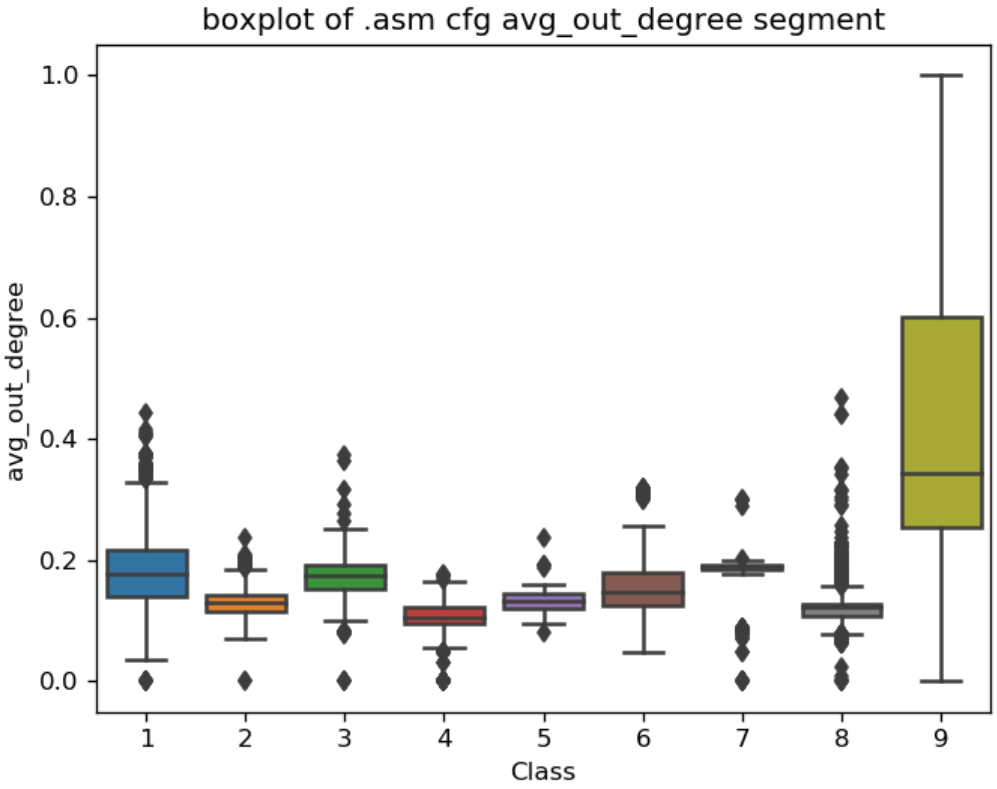
In [31]:

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



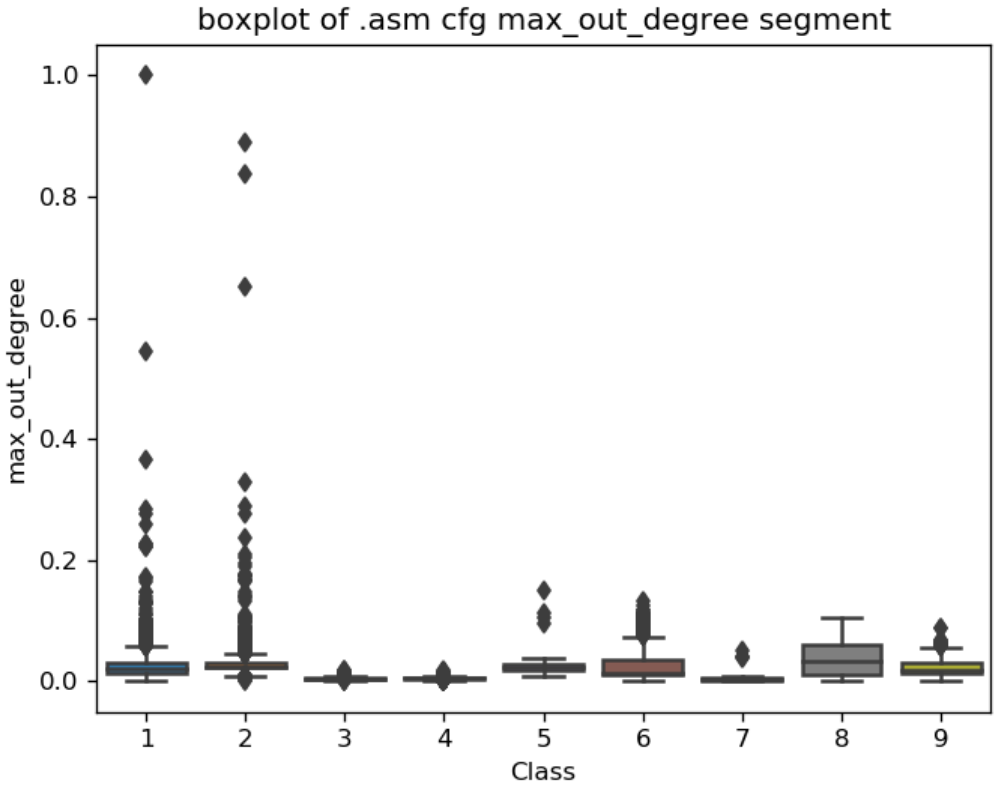
In [32]:

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



In [33]:

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



In [34]:

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

Machine leraning model on features of asm graph

In [32]:

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

Out[32]:

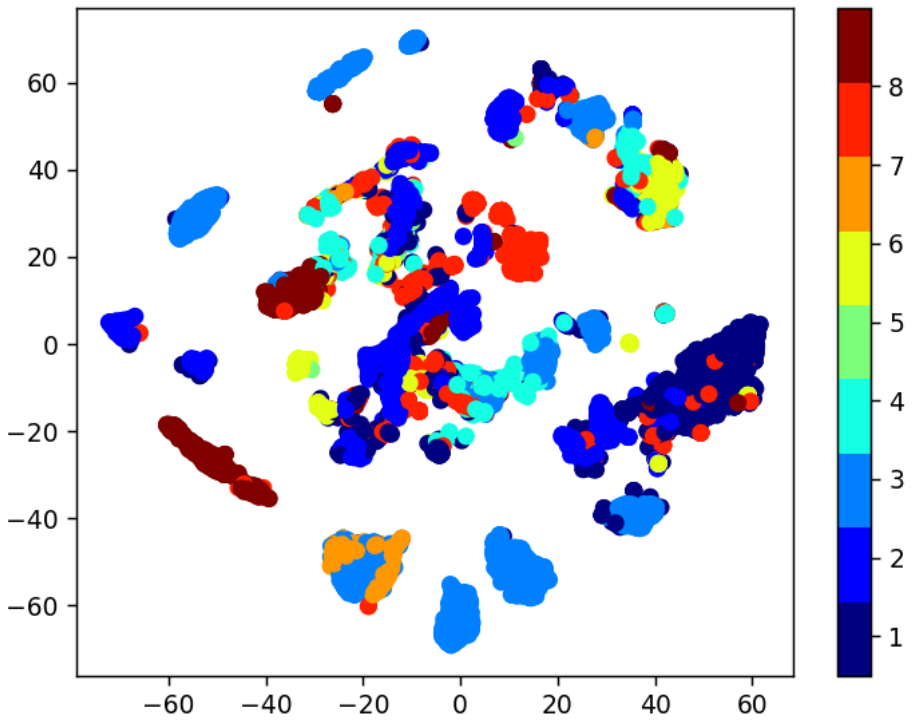
	has_program_entry_point	nodes	edges	avg_in_degree	avg_out_degree	max_out_de
0	0	0.010336	0.047084	0.769626	0.769626	0.04
1	0	0.008469	0.007154	0.142723	0.142723	0.04
2	0	0.001600	0.002472	0.260980	0.260980	0.01
3	0	0.002734	0.001461	0.090271	0.090271	0.00
4	0	0.006402	0.004795	0.126533	0.126533	0.05

In [19]:

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

In [20]:

```
xtsne=TSNE(perplexity=100)
results=xtsne.fit_transform(result_x)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```



In [33]:

```
X_train_graph, X_test_graph, y_train_graph, y_test_graph = train_test_split(result_x,re
sult_y ,stratify=result_y,test_size=0.20)
X_train_graph, X_cv_graph, y_train_graph, y_cv_graph = train_test_split(X_train_graph,
y_train_graph,stratify=y_train_graph,test_size=0.20)
```

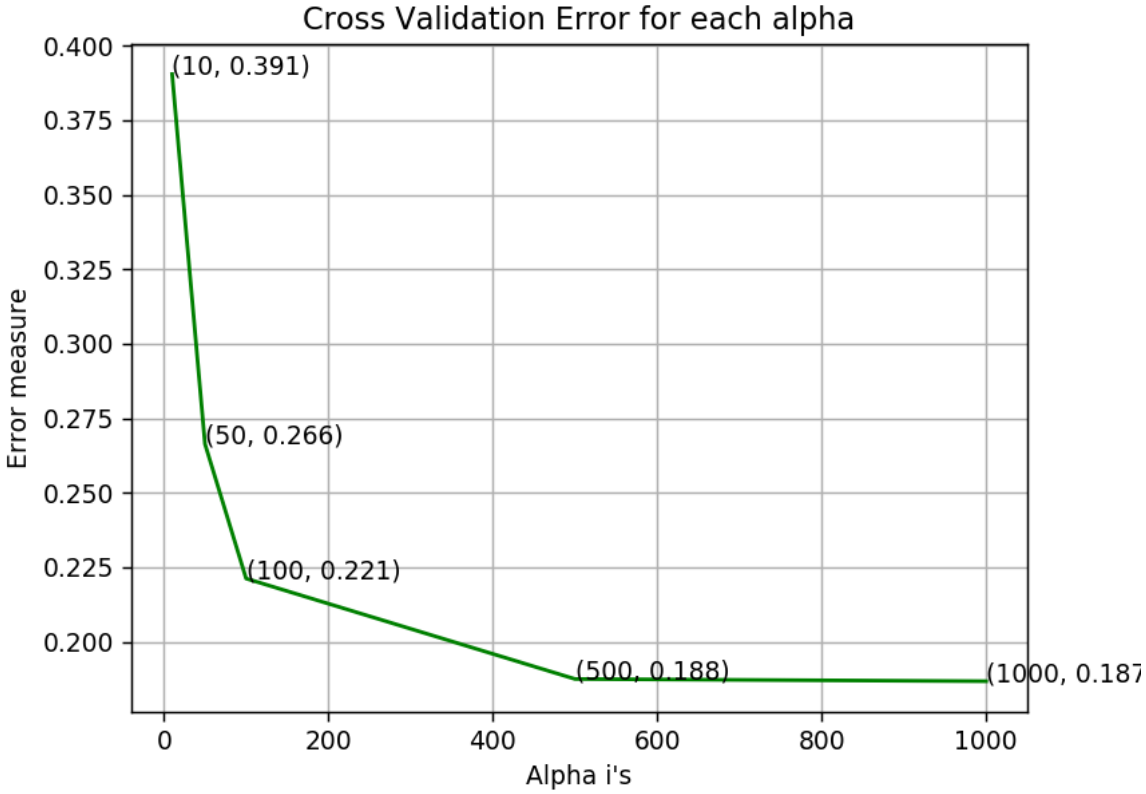
4.4.4 XgBoost Classifier

In [42]:

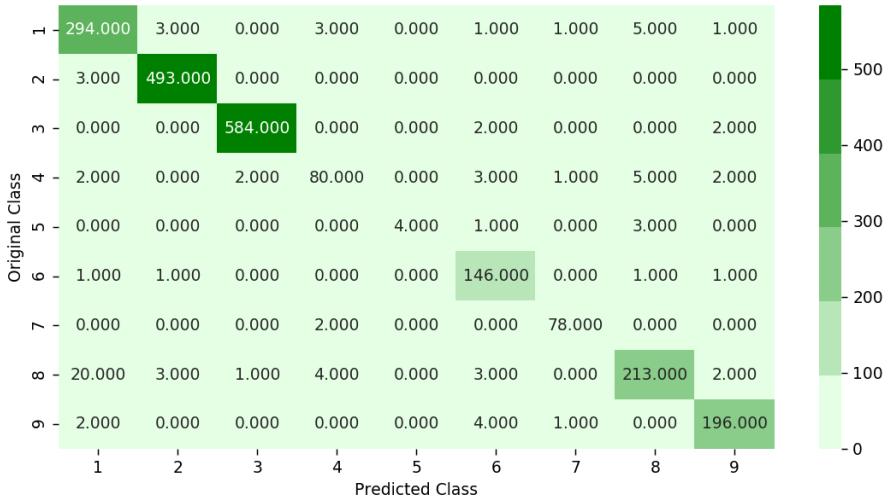
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

alpha=[10,50,100,500,1000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i, nthread=-1)
    x_cfl.fit(X_train_graph,y_train_graph)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_graph, y_train_graph)
    predict_y = sig_clf.predict_proba(X_cv_graph)
    cv_log_error_array.append(log_loss(y_cv_graph, predict_y, labels=x_cfl.classes_, eps=1e-15))
    print ('log_loss for c = ',i,'is',log_loss(y_cv_graph, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
sig_clf.fit(X_train_graph, y_train_graph)

predict_y = sig_clf.predict_proba(X_train_graph)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
oss(y_train_graph, predict_y))
predict_y = sig_clf.predict_proba(X_cv_graph)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv_graph, predict_y))
predict_y = sig_clf.predict_proba(X_test_graph)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test_graph, predict_y))
plot_confusion_matrix(y_test_graph,sig_clf.predict(X_test_graph))
```

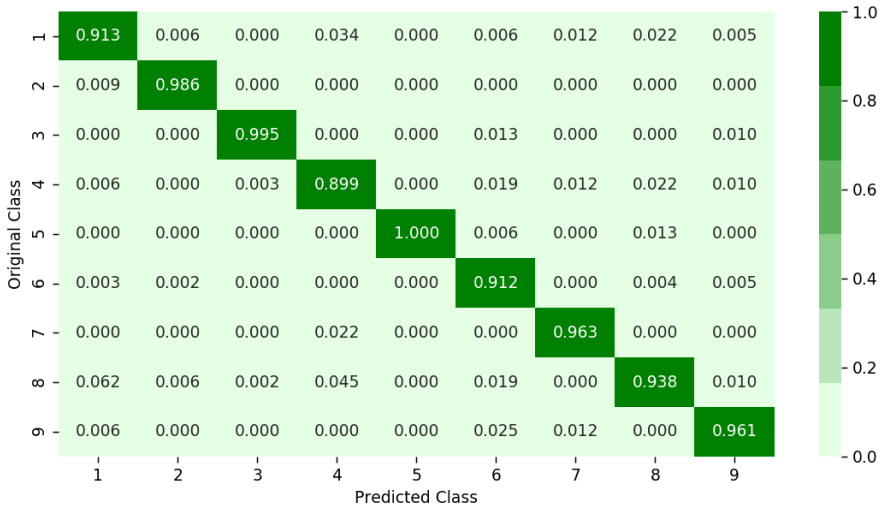
```
log_loss for c =  10 is 0.39050993240289195
log_loss for c =  50 is 0.2663667909342192
log_loss for c =  100 is 0.22121991978648897
log_loss for c =  500 is 0.18751395131374837
log_loss for c =  1000 is 0.1867589444943258
```



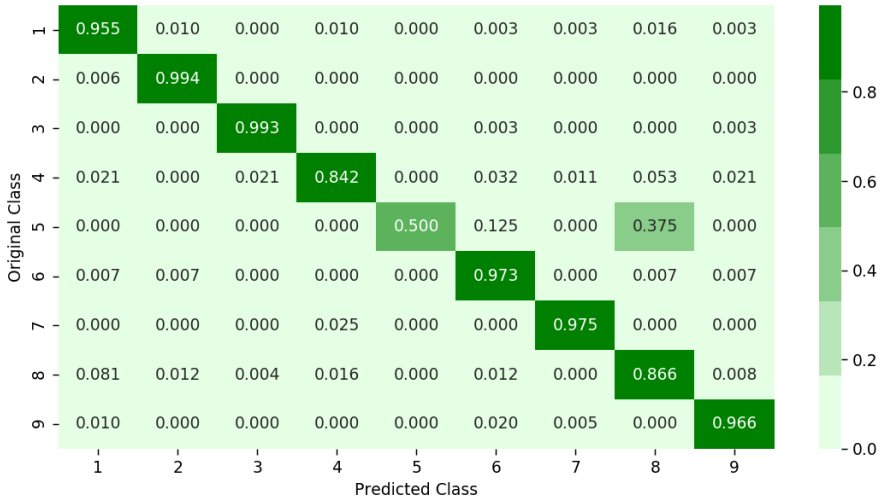

```
For values of best alpha =  1000 The train log loss is: 0.0628009396254028
For values of best alpha =  1000 The cross validation log loss is: 0.18675
89444943258
For values of best alpha =  1000 The test log loss is: 0.17284964723632895
Number of misclassified points  3.9558417663293466
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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



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

In [43]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2,0.25,0.3],
     'n_estimators':[100,300,500,800,1000],
     'max_depth':[3,4,5,6],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,n_it
er=30)
random_cfl.fit(X_train_graph, y_train_graph)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   16.8s
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed:   40.2s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:   56.2s
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done  61 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  77 out of  90 | elapsed:  3.1min remaining:
31.6s
[Parallel(n_jobs=-1)]: Done  87 out of  90 | elapsed:  3.4min remaining:
6.9s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:  3.4min finished
```

Out[43]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtre
e',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_st
ep=0,
                                           max_depth=3, min_child_weight=
1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsamp
le=1,
                                           verbosity=1),
                   iid='warn', n_iter=30, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.
5, 1],
                                        'learning_rate': [0.01, 0.03, 0.0
5, 0.1,
                                                          0.15, 0.2, 0.25,
                                                          0.3],
                                        'max_depth': [3, 4, 5, 6],
                                        'n_estimators': [100, 300, 500, 80
0,
                                                         1000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [44]:

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

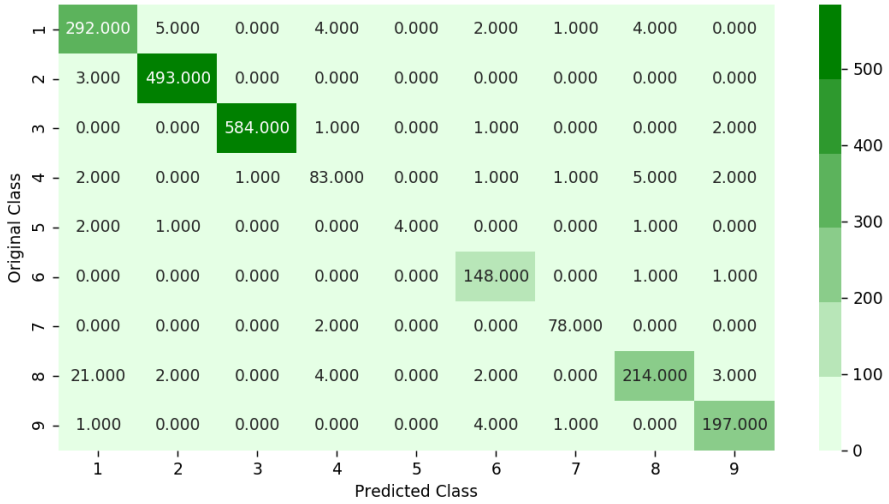
```
{'subsample': 1, 'n_estimators': 100, 'max_depth': 6, 'learning_rate': 0.1
5, 'colsample_bytree': 1}
```

4.4.5 Xgboost Classifier with best hyperparameters

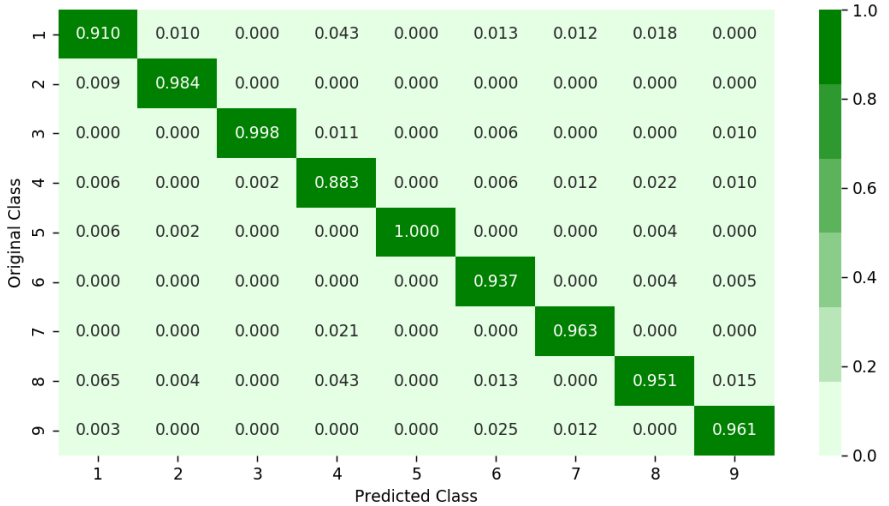
In [46]:

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

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

x_cfl=XGBClassifier(n_estimators=100,max_depth=6,learning_rate=0.15,colsample_bytree=1, subsample=1,nthread=-1)
x_cfl.fit(X_train_graph,y_train_graph,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_graph, y_train_graph)

predict_y = sig_clf.predict_proba(X_train_graph)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_graph, predict_y))
predict_y = sig_clf.predict_proba(X_cv_graph)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_graph, predict_y))
predict_y = sig_clf.predict_proba(X_test_graph)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_graph, predict_y))
plot_confusion_matrix(y_test_graph,sig_clf.predict(X_test_graph))
```

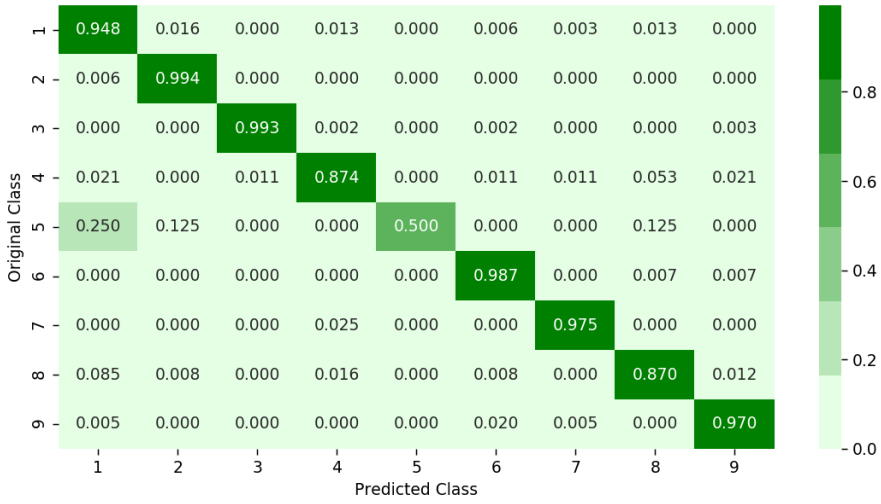
```
For values of best alpha =  1000 The train log loss is: 0.0617291387134080
44
For values of best alpha =  1000 The cross validation log loss is: 0.18324
88156717289
For values of best alpha =  1000 The test log loss is: 0.1618956619696228
Number of misclassified points  3.7258509659613614
---------------------------------------------------- Confusion matrix ------
------------------------------------------
```




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




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



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

In [ ]:

```
conclusion_table.add_row(["asm_Graph",log_loss(y_train_graph, predict_y),log_loss(y_tes
t_graph, predict_y)])
```

Observations---->

classes 4 and 6 can be well sepearted by feature has_program_entry_point

avg_out_degree turns out to be a good feature

Contructing external subroutines features

Cannot take local subroutines into account as same subroutines in different programs have different symbolic names but not in case of external subrotines symbolic names they are same in different programs

So, we take only external subrotines into account and apply Tfidf

We can also use some features to get similarity between local subroutines and put them in common sets of local routines then we can take lcoal subroutines also into account

we can get sub_routines similarity by three methods

1. **Mathcing local subroutines based on common external subroutines calls**
2. **Matching subroutines based on there x86instruction as 15 bit-color code and vector of opcode(used when no external subroutines)**
3. **3**

Here we donot local subroutines niether used any similarity macthing

This paper shows we can achieve this
http://www.cs.sjsu.edu/faculty/stamp/papers/topics/topic5/10.1007-s11416-012-0175-y.pdf (http://www.cs.sjsu.edu/faculty/stamp/papers/topics/topic5/10.1007-s11416-012-0175-y.pdf)

In [117]:

```
import re
subrotine_asm = []
subrotine_asm_y = []
source='train/asmGraphs/'
files = os.listdir(source)
index = 0
for filename in tqdm(files):
    G = nx.read_gpickle(source+filename)
    subrotine_asm_y.append(filename.split('.')[0])
    row = []
    if len(list(G.nodes)) != 0:
        for node in list(G.nodes):

                # Get those routines which are external
                if not node.startswith('sub_'):

                # remove integer named routines
                    if not isinstance(node, int):

                        # remove special char from routines
                        node = re.sub(r'[^A-Za-z0-9#+.\-]+','',node)
                        row.append(str(node))

        subrotine_asm.append(str(" ".join(row).lower()))
    else:
        subrotine_asm.append('None')

    index+=1
```

In [119]:

```
result_asm_subroutines = pd.DataFrame(list(zip(subrotine_asm_y,subrotine_asm)), columns
=['ID','subroutines'])
result_asm_subroutines = pd.merge(result_asm_subroutines,Y,on='ID', how='left')
result_asm_subroutines.head()
```

Out[119]:

	ID	subroutines	Class
0	01azqd4InC7m9JpocGv5	atexit findwindowa isvalidlocale virtualalloc ...	9
1	01IsoiSMh5gxyDYTl4CB	3yaxpaxz eax 2yapaxiz 0exceptionstdqaeabqbdz c...	2
2	01jsnpXSAlgw6aPeDxrU	virtualalloc loadlibrarya 545279 winmain16 3	9
3	01kcPWA9K2BOxQeS5Rju	0basicstringguchartraitsgstdvallocatorg2stdqae...	1
4	01SuzwMJEIXsK7A8dQbl	ecx exitprocess ebp+var10 processtrace createt...	8

In [120]:

```
# to csv
result_asm_subroutines.to_csv('asm_subroutines_features.csv',index=False)
```

In [17]:

```
# read csv
result_asm_subroutines = pd.read_csv('asm_subroutines_features.csv')
result_asm_subroutines.head()
```

Out[17]:

	ID	subroutines	Class
0	01azqd4InC7m9JpocGv5	atexit findwindowa isvalidlocale virtualalloc ...	9
1	01IsoiSMh5gxyDYTl4CB	3yaxpaxz eax 2yapaxiz 0exceptionstdqaeabqbdz c...	2
2	01jsnpXSAlgw6aPeDxrU	virtualalloc loadlibrarya 545279 winmain16 3	9
3	01kcPWA9K2BOxQeS5Rju	0basicstringguchartraitsgstdvallocatorg2stdqae...	1
4	01SuzwMJEIXsK7A8dQbl	ecx exitprocess ebp+var10 processtrace createt...	8

Bow on routines

In [48]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer =  CountVectorizer(binary='false')
bow_subrotine = vectorizer.fit_transform(result_asm_subroutines['subroutines'])
print("Number of features:",len(vectorizer.get_feature_names()))
```

```
Number of features: 220110
```

In [49]:

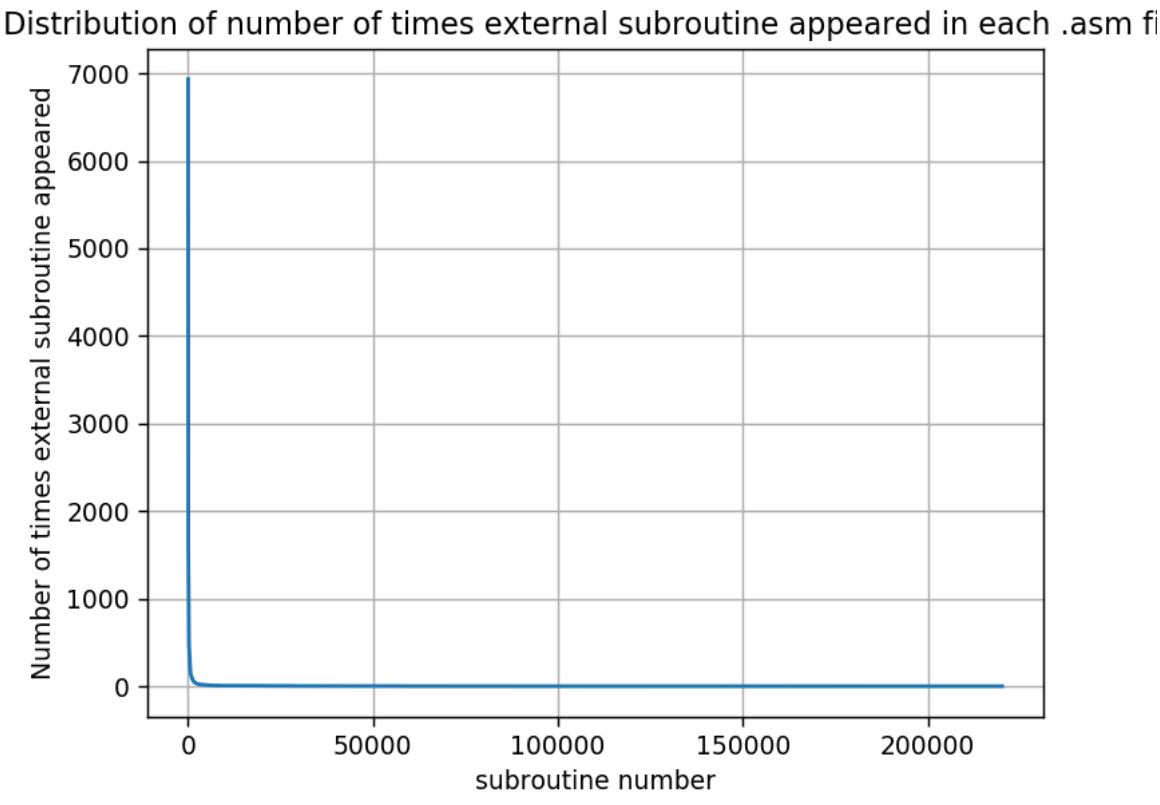
```
frequency_subroutines = pd.DataFrame(bow_subrotine.sum(axis=0).T,columns=['frequncy'])
frequency_subroutines =frequency_subroutines.join(pd.DataFrame(vectorizer.get_feature_n
ames(), columns=['feature']))
frequency_subroutines = frequency_subroutines.sort_values(['frequncy'], ascending=False
)
subroutines_counts = frequency_subroutines['frequncy'].values
frequency_subroutines.head()
```

Out[49]:

	frequncy	feature
57460	6934	eax
58754	5826	edi
61056	5172	esi
108994	4644	loadlibrarya
57519	4242	ebp

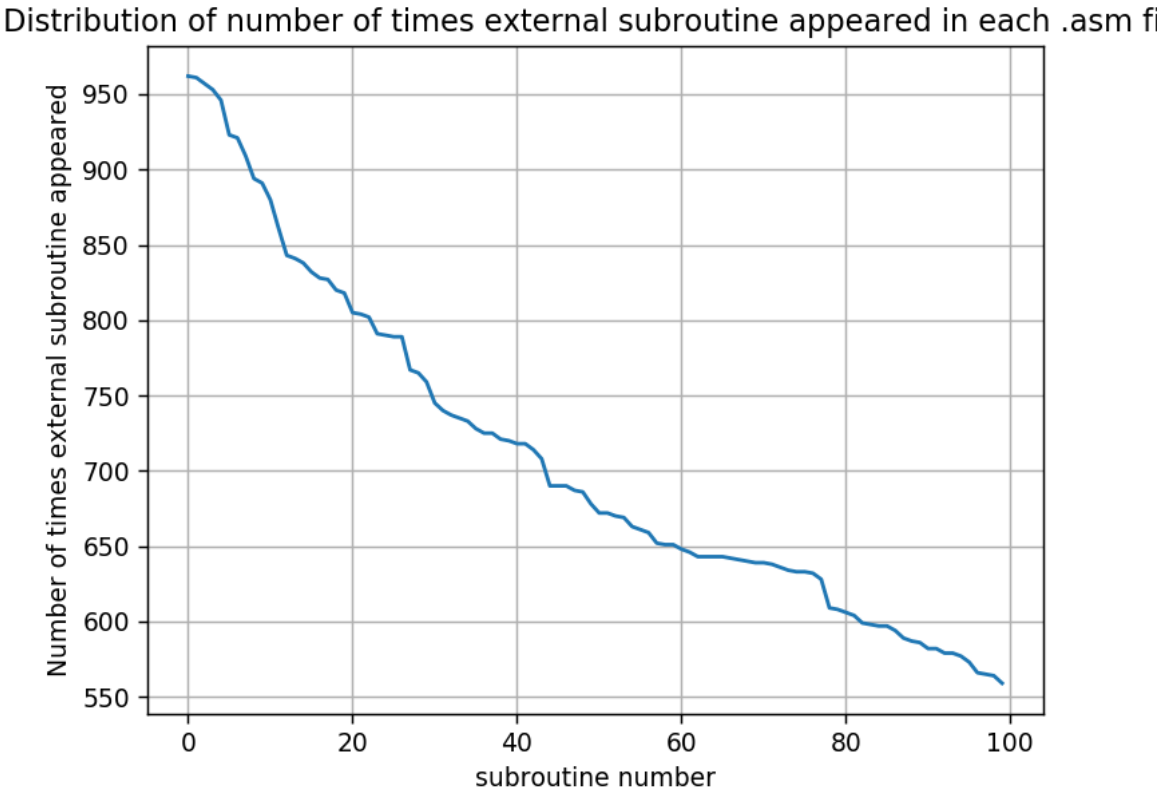
In [50]:

```
plt.plot(subroutines_counts)
plt.title("Distribution of number of times external subroutine appeared in each .asm fi
le")
plt.grid()
plt.xlabel("subroutine number")
plt.ylabel("Number of times external subroutine appeared")
plt.show()
```



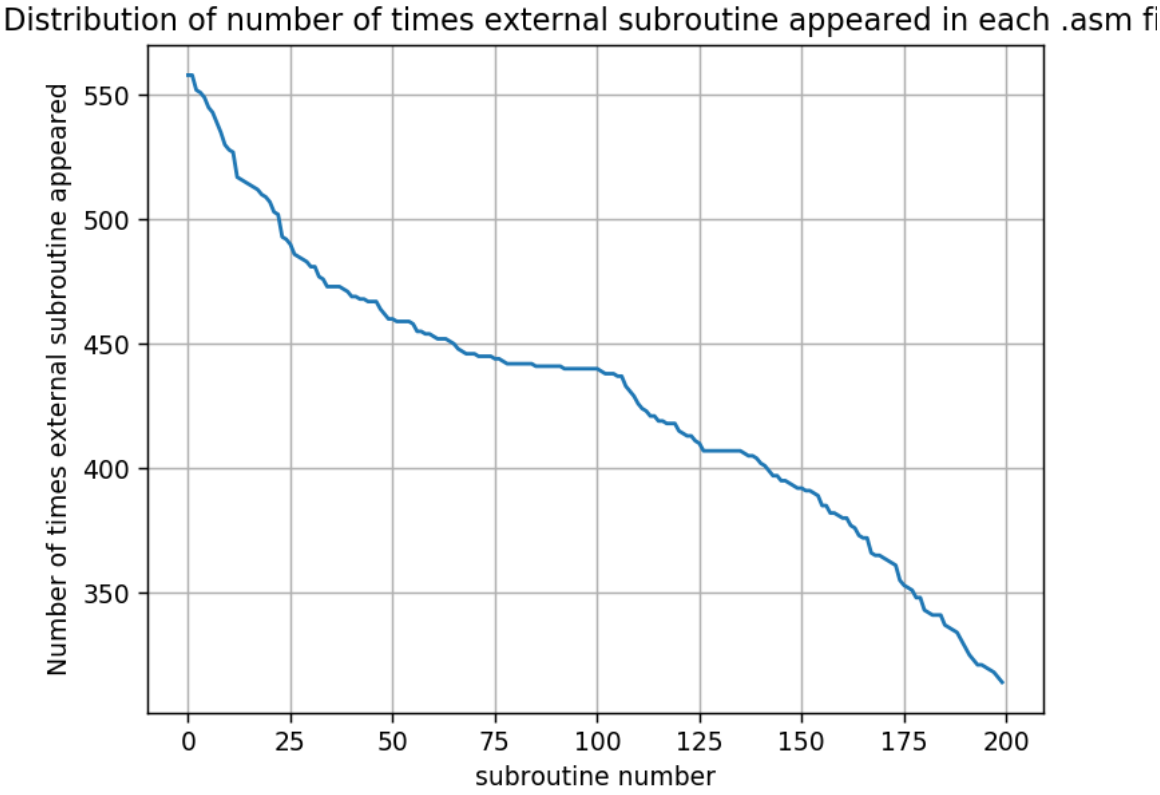
In [51]:

```
plt.plot(subroutines_counts[100:200])
plt.title("Distribution of number of times external subroutine appeared in each .asm fi
le")
plt.grid()
plt.xlabel("subroutine number")
plt.ylabel("Number of times external subroutine appeared")
plt.show()
```



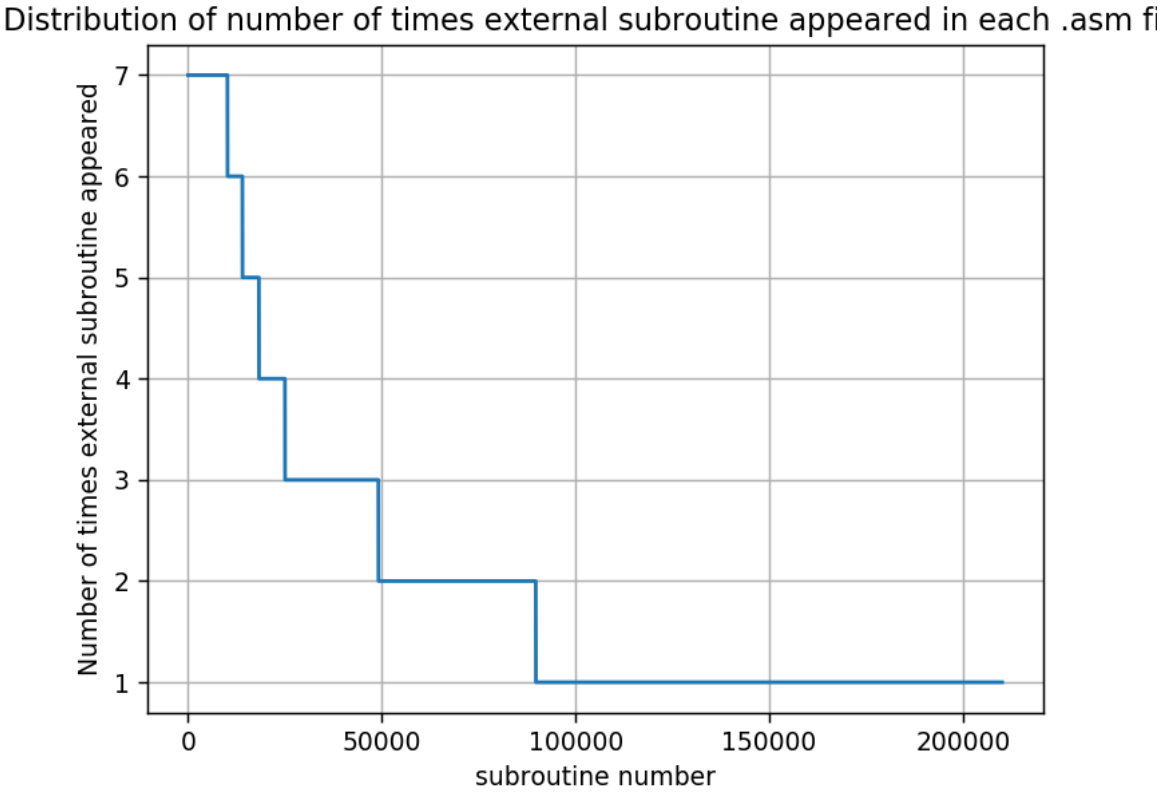
In [52]:

```
plt.plot(subroutines_counts[200:400])
plt.title("Distribution of number of times external subroutine appeared in each .asm fi
le")
plt.grid()
plt.xlabel("subroutine number")
plt.ylabel("Number of times external subroutine appeared")
plt.show()
```



In [53]:

```
plt.plot(subroutines_counts[10000:])
plt.title("Distribution of number of times external subroutine appeared in each .asm fi
le")
plt.grid()
plt.xlabel("subroutine number")
plt.ylabel("Number of times external subroutine appeared")
plt.show()
```



In [54]:

```
subroutines_gt_1 = frequency_subroutines[frequency_subroutines.frequncy>100]
print ('{} subroutines are used more than 100 times'.format(len(subroutines_gt_1)))
subroutines_gt_5 = frequency_subroutines[frequency_subroutines.frequncy>500]
print ('{} subroutines are used more than 500 times'.format(len(subroutines_gt_5)))
subroutines_gt_1k = frequency_subroutines[frequency_subroutines.frequncy>1000]
print ('{} subroutines are used more than 1000 times'.format(len(subroutines_gt_1k)))
subroutines_gt_5k = frequency_subroutines[frequency_subroutines.frequncy>5000]
print ('{} subroutines are used more than 5000 times'.format(len(subroutines_gt_5k)))
print("External sub routine 'eax' appears 6934 times")
```

```
998 subroutines are used more than 100 times
223 subroutines are used more than 500 times
91 subroutines are used more than 1000 times
3 subroutines are used more than 5000 times
External sub routine 'eax' appears 6934 times
```

Tfidf with lowest 2000 idf_ values

In [55]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer =  TfidfVectorizer(min_df=0.0015,binary='false')
tfidf_subrotine = vectorizer.fit_transform(result_asm_subroutines['subroutines'])
print("Number of features:",len(vectorizer.get_feature_names()))
```

```
Number of features: 4466
```

In [56]:

```
idf_features = pd.DataFrame(list(zip(vectorizer.get_feature_names(),vectorizer.idf_)),
columns=['features','idf'])
```

In [57]:

```
features = vectorizer.get_feature_names()
idf_features_n = idf_features.sort_values(['idf'], ascending=True)[:500]
idf_features_n.head()
```

Out[57]:

	features	idf
1063	eax	1.449334
1070	edi	1.623412
1134	esi	1.742462
2753	loadlibrarya	1.850123
1065	ebp	1.940644

In [58]:

```
# Only Tfidf with least 2000 idf_ values
index = [features.index(feature) for feature in idf_features_n['features']]
tfidf_subrotine = tfidf_subrotine[:,index]
```

machine learning model on features of external subroutines from asm file

In [59]:

```
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(tfidf_subrotine,res
ult_asm_subroutines['Class'] ,stratify=result_asm_subroutines['Class'],test_size=0.20)
X_train_sub, X_cv_sub, y_train_sub, y_cv_sub = train_test_split(X_train_sub, y_train_su
b,stratify=y_train_sub,test_size=0.20)
```

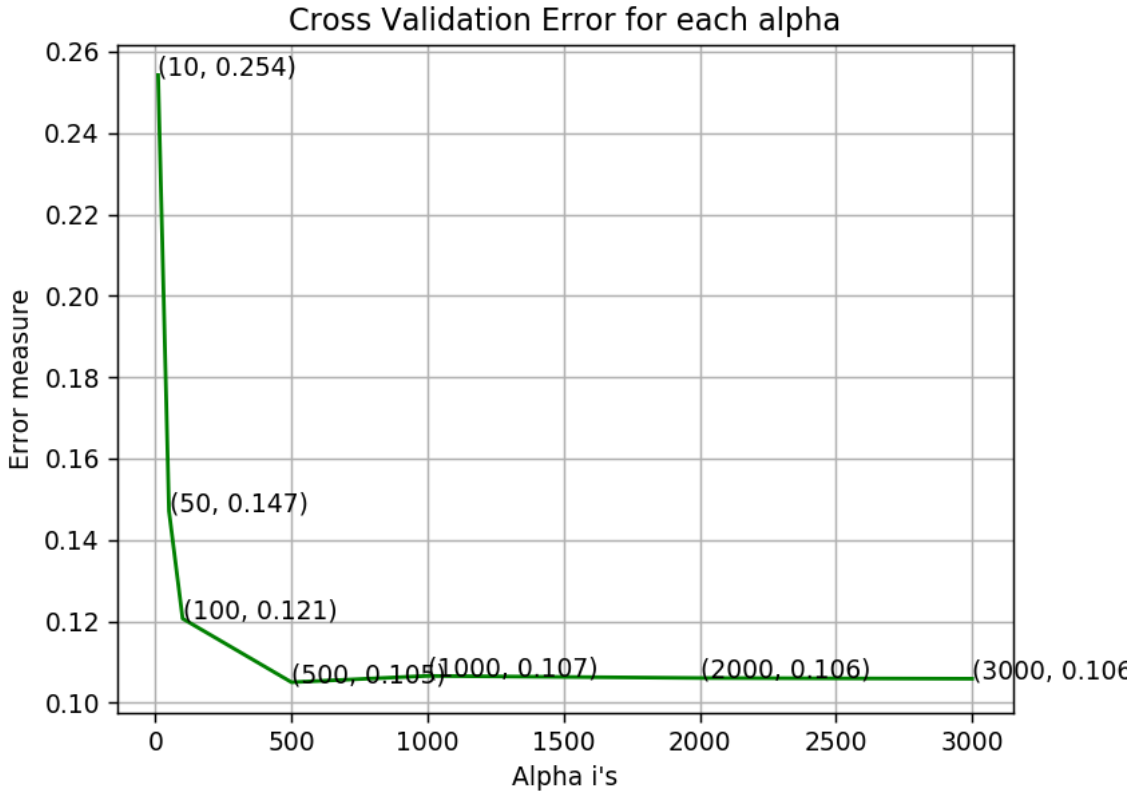
4.4.4 XgBoost Classifier

In [64]:

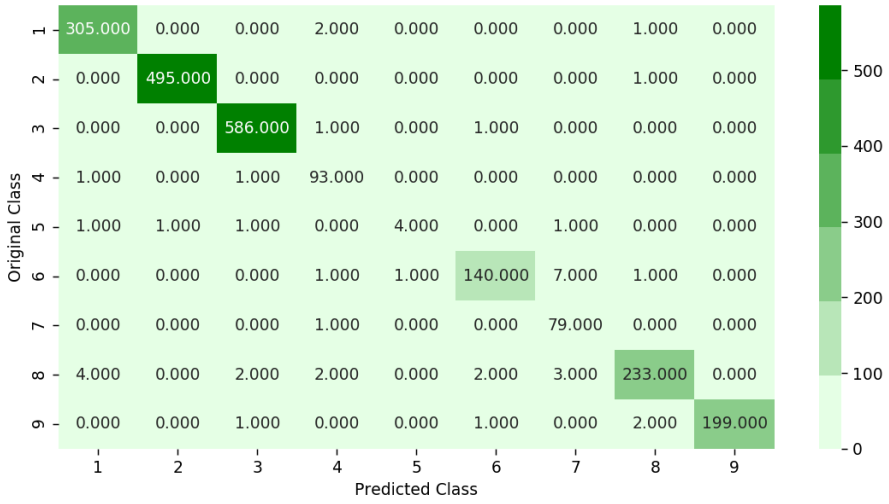
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
predict_y = sig_clf.predict_proba(X_train_sub)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
oss(y_train_sub, predict_y))
predict_y = sig_clf.predict_proba(X_cv_sub)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv_sub, predict_y))
predict_y = sig_clf.predict_proba(X_test_sub)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test_sub, predict_y))
plot_confusion_matrix(y_test_sub,sig_clf.predict(X_test_sub))
```

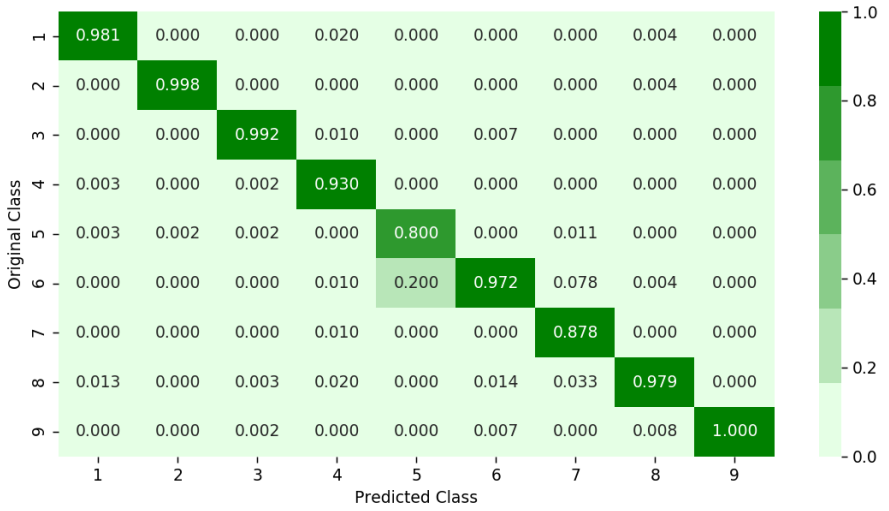
```
log_loss for c =  10 is 0.254229532600997
log_loss for c =  50 is 0.14711410773739744
log_loss for c =  100 is 0.12070219837698394
log_loss for c =  500 is 0.10506646010614465
log_loss for c =  1000 is 0.10662204449875147
log_loss for c =  2000 is 0.1061030712057786
log_loss for c =  3000 is 0.1059230056663791
```



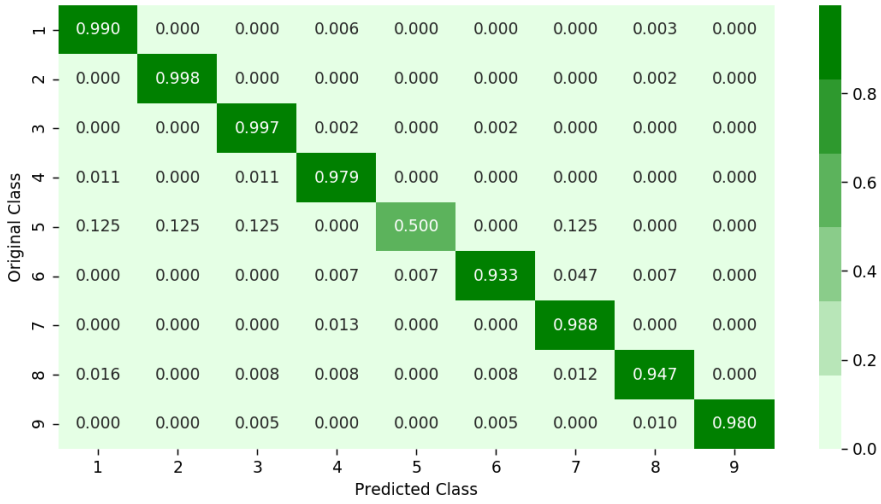

```
For values of best alpha =  500 The train log loss is: 0.05544327108635855
For values of best alpha =  500 The cross validation log loss is: 0.105066
46010614465
For values of best alpha =  500 The test log loss is: 0.09211435053144845
Number of misclassified points  1.8399264029438822
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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



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

In [65]:

```
x_cfl=XGBClassifier()

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

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   18.5s
[Parallel(n_jobs=-1)]: Done  11 out of  30 | elapsed:  2.4min remaining:
4.2min
[Parallel(n_jobs=-1)]: Done  15 out of  30 | elapsed:  2.8min remaining:
2.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  3.8min remaining:
2.2min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  4.8min remaining:
1.5min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  5.2min remaining:
34.6s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  5.3min finished
```

Out[65]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtre
e',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_st
ep=0,
                                           max_depth=3, min_child_weight=
1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsamp
le=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.
5, 1],
                                        'learning_rate': [0.01, 0.03, 0.0
5, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 10
00,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [66]:

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

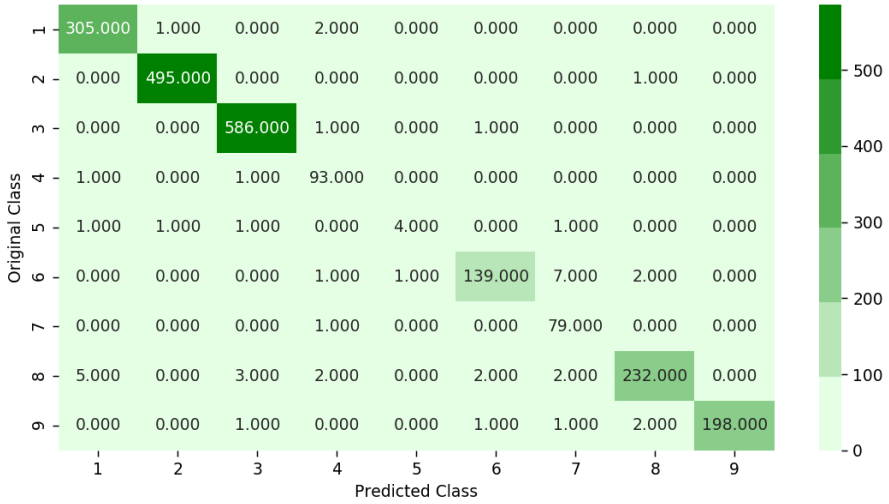
```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 10, 'learning_rate': 0.
1, 'colsample_bytree': 0.3}
```

4.4.5 Xgboost Classifier with best hyperparameters

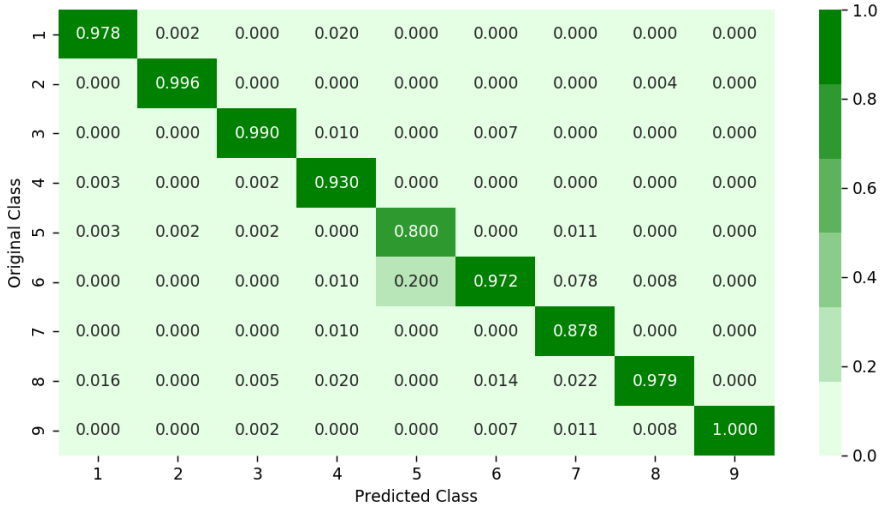
In [67]:

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

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

x_cfl=XGBClassifier(n_estimators=200,max_depth=10,learning_rate=0.1,colsample_bytree=0.3,subsample=1,nthread=-1)
x_cfl.fit(X_train_sub,y_train_sub,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_sub, y_train_sub)

predict_y = sig_clf.predict_proba(X_train_sub)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_sub, predict_y))
predict_y = sig_clf.predict_proba(X_cv_sub)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_sub, predict_y))
predict_y = sig_clf.predict_proba(X_test_sub)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_sub, predict_y))
plot_confusion_matrix(y_test_sub,sig_clf.predict(X_test_sub))
```

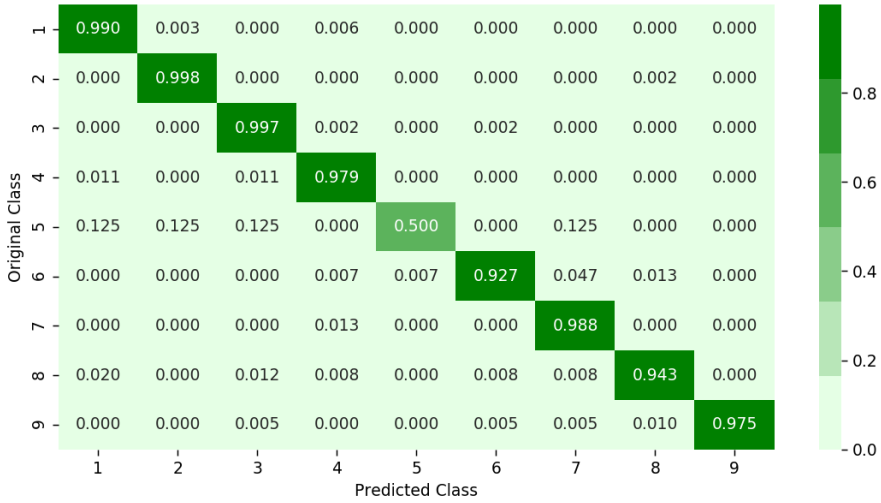
```
For values of best alpha =  500 The train log loss is: 0.05531057965840496
For values of best alpha =  500 The cross validation log loss is: 0.095001
87728185276
For values of best alpha =  500 The test log loss is: 0.09410721669170027
Number of misclassified points  1.9779208831646733
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```




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




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



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

In [81]:

```
conclusion_table.add_row(["asm_subroutines",log_loss(y_train_sub, predict_y),log_loss(y
_test_sub, predict_y)])
```

Constructing similarity b/w local subroutine

Core-Idea --> two programs with similar local subroutine appears to be of same classes

In [184]:

```
# those subroutines as key which have 1 or more external subroutines relation only out
 degree and value are the external sub.
subroutines = {}
source='train/asmGraphs/'
files = os.listdir(source)
index = 0
for filename in tqdm(files):
#     print(filename)
    G = nx.read_gpickle(source+filename)
    for node in list(G.nodes):
        if node.startswith('sub_'):
            list_out_degree =  G.out_degree(node)
            external_subroutines = [ node  for node in list(G.successors(node)) if not
node.startswith('sub_')]
            if len(external_subroutines)>=4:
                subroutines[node] = set(external_subroutines)
#             break
#     break
    index+=1
```

We use this formulae to evalute similarity between two local subroutines

len(common_ex_routines_in_both_local_subroutines)/max(len(local_subroutines), len(local_subroutines)) > some%

examples -->

1. **l1 = [a,b,c], l2 = [a,d,e], sim(l1,l2) = 33.33%**
2. **l1 = [a,b,c], l2 = [a,b,e], sim(l1,l2) = 66.66%**
3. **l1 = [a,b,c], l2 = [a,b], sim(l1,l2) = 66.66% size of list dosen't matter**
4. **l1 = [a,b,c,d,e], l2 = [a,b,c], sim(l1,l2) = 66.66%**

In [ ]:

```
# if not work make sim of sub b/w  programs not in itself also
```

In [ ]:

```
# those subroutines as key which have 1 or more external subroutines relation only out
 degree and value are the external sub.
import time
# containing all common sub sets with keys as index which is key in subroutines_cluster
_index values as set of common sub
subroutines_common_sub_sets = {}

# containing all extertanl sub sets with keys as index which is key in subroutines_comm
on_sub_sets
#values as set of cluster ex_sub
subroutines_cluster_index = {}

cluster_count = 0

source='train/asmGraphs/'

files = os.listdir(source)

index = 0

# These are hyparameter
# min_ex_sub : min number of external subroutines required to appear to take account fo
r its similarity
# as example ['some_ex_sub'],['someother_ex_sub'] can have 100% similarity

# min_ex_sub : min percentage of similarity by formulae below to make two local subrout
ines similar
#len(common_ex_routines_in_both_local_subroutines)/max( len(local_subroutines), len(loc
al_subroutines) ) > some%
min_ex_sub = 1
min_sim = 0.95

for filename in tqdm(files):

    G = nx.read_gpickle(source+filename)

    for node in list(G.nodes):

        if node.startswith('sub_'):

            list_out_degree =  G.out_degree(node)

            # getting local routines with external subroutine call
            external_subroutines = set([ node  for node in list(G.successors(node)) if
not node.startswith('sub_')])

            # if there are more n externl subroutine only then will available to check
 for similarity
            if len(external_subroutines)>=min_ex_sub:

                # comparing similarity with all subroutines clusters

                # checking for first run i.e it has no clusters
                if len(subroutines_cluster_index) !=0 and len(subroutines_common_sub_se
ts) !=0:
                    # check which cluster does this subroutine similar to (highest siml
arity)

                    # Get sim matrix b/w all cluster till this current loop
```

```
                    sim_matrix_cluster = np.zeros((1,len(subroutines_cluster_index)))

                    for idx_key,ex_sub_cluster in subroutines_cluster_index.items():
                        sim = (len(external_subroutines.intersection(ex_sub_cluster))/
                                max(len(external_subroutines),len(ex_sub_cluster)))
                        sim_matrix_cluster[:,idx_key] = sim

                    # found a similar cluster of external_subroutines adding
                    if np.amax(sim_matrix_cluster) >= min_sim:

                        # get index of cluster
                        max_sim_index = np.where(sim_matrix_cluster == np.amax(sim_matr
ix_cluster))[1][0]

                         #get common subs of that index from subroutines_common_sub_set
s add to common node
                        subroutines_common_sub_sets.get(max_sim_index).add(node)

                    else:
                        # creating new cluster
                        subroutines_cluster_index[cluster_count] = external_subroutines
                        subroutines_common_sub_sets[cluster_count] = set([node])
                        cluster_count+=1

                else:
                    # inserting for first time
                    subroutines_cluster_index[cluster_count] = external_subroutines
                    subroutines_common_sub_sets[cluster_count] = set([node])
                    cluster_count+=1

            #break
    #break
    # flush those which have only 1 sub_ in cluster or set (this will misout some infor
mation but makes code run faster)
#     if index%100 == 0:
#         subroutines_common_sub_sets = {}
#         for key, value in subroutines_common_sub_sets.items():
#             if len(value)==1:
#                 subroutines_common_sub_sets.pop(key,None)
#                 subroutines_cluster_index.pop(key,None)
    index+=1
```

In [338]:

```
# get those cluster who have > 1 length of routines
cluster_common_sub_routines = []
for key, value in subroutines_common_sub_sets.items():
    if len(value)>=2:
        cluster_common_sub_routines.append(list(value))
```

In [345]:

```
# construct an array with these common subroutines
subrotine_asm = np.zeros((10868,len(cluster_common_sub_routines)))
subrotine_asm_y = []

source='train/asmGraphs/'
files = os.listdir(source)
index = 0
for filename in tqdm(files):
    G = nx.read_gpickle(source+filename)
    subrotine_asm_y.append(Y[Y['ID'] == filename.split('.')[0]]['Class'].values[0])
    row = []
    for idx,common_sub in enumerate(cluster_common_sub_routines):
        for subroutines in common_sub:
            if G.has_node(subroutines):
                subrotine_asm[index,idx]+=1
    index+=1
```

In [371]:

```
print("Number of files with no common sub_routines: {}".format(np.count_nonzero(subroti
ne_asm.sum(axis=1))))
```

```
Number of files with no common sub_routines: 9858
```

Using similarity based on local_subroutines as a features is not possible because very less inforamtion available

we can fine tune this by changing two paramters min_ex_sub and min_sim

we can also reduce features using features(sub_routines) which appears less frequent but not very much less in files

Machine learning model on both asm_graph and asm_subroutines features

merging both asm_graph and asm_subroutines features

In [83]:

```
result_asm_subroutines.head()
```

Out[83]:

	ID	subroutines	Class
0	01azqd4InC7m9JpocGv5	atexit findwindowa isvalidlocale virtualalloc ...	9
1	01IsoiSMh5gxyDYTl4CB	3yaxpaxz eax 2yapaxiz 0exceptionstdqaeabqbdz c...	2
2	01jsnpXSAlgw6aPeDxrU	virtualalloc loadlibrarya 545279 winmain16 3	9
3	01kcPWA9K2BOxQeS5Rju	0basicstringguchartraitsgstdvallocatorg2stdqae...	1
4	01SuzwMJEIXsK7A8dQbl	ecx exitprocess ebp+var10 processtrace createt...	8

In [84]:

```
result_asm_graph.head()
```

Out[84]:

	ID	has_program_entry_point	nodes	edges	avg_in_degree	a
0	01azqd4InC7m9JpocGv5	0	0.010336	0.047084	0.769626	
1	01IsoiSMh5gxyDYTl4CB	0	0.008469	0.007154	0.142723	
2	01jsnpXSAlgw6aPeDxrU	0	0.001600	0.002472	0.260980	
3	01kcPWA9K2BOxQeS5Rju	0	0.002734	0.001461	0.090271	
4	01SuzwMJEIXsK7A8dQbl	0	0.006402	0.004795	0.126533	

5 rows × 21 columns

In [85]:

```
print(result_asm_subroutines.shape)
print(result_asm_graph.shape)
```

```
(10868, 3)
(10868, 21)
```

In [86]:

```
result_x = pd.merge(result_asm_graph,result_asm_subroutines.drop(['Class'], axis=1),on=
'ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['Class','ID'], axis=1)
result_x.head()
```

Out[86]:

	has_program_entry_point	nodes	edges	avg_in_degree	avg_out_degree	max_out_de
0	0	0.010336	0.047084	0.769626	0.769626	0.04
1	0	0.008469	0.007154	0.142723	0.142723	0.04
2	0	0.001600	0.002472	0.260980	0.260980	0.01
3	0	0.002734	0.001461	0.090271	0.090271	0.00
4	0	0.006402	0.004795	0.126533	0.126533	0.05

In [87]:

```
X_train_merge2, X_test_merge2, y_train_merge2, y_test_merge2 = train_test_split(result_
x,result_y ,stratify=result_y,test_size=0.20)
X_train_merge2, X_cv_merge2, y_train_merge2, y_cv_merge2 = train_test_split(X_train_mer
ge2, y_train_merge2,stratify=y_train_merge2,test_size=0.20)
```

Tfidf with least 2000 idf_ values

In [88]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer =  TfidfVectorizer(min_df=0.0015,binary='false')
tfidf_subrotine_tr = vectorizer.fit_transform(X_train_merge2['subroutines'])
print("Number of features:",len(vectorizer.get_feature_names()))
```

```
Number of features: 4576
```

In [89]:

```
tfidf_subrotine_cv = vectorizer.transform(X_cv_merge2['subroutines'])
tfidf_subrotine_te = vectorizer.transform(X_test_merge2['subroutines'])
```

find least idf_values

In [90]:

```
idf_features = pd.DataFrame(list(zip(vectorizer.get_feature_names(),vectorizer.idf_)),
columns=['features','idf'])
```

In [91]:

```
features = vectorizer.get_feature_names()
idf_features_n = idf_features.sort_values(['idf'], ascending=True)[:1000]
idf_features_n.head()
```

Out[91]:

	features	idf
1119	eax	1.457546
1126	edi	1.634520
1194	esi	1.748717
2814	loadlibrarya	1.856106
1121	ebp	1.942656

filter train, test , cv with onlt selected idf values features

In [92]:

```
# Only Tfidf with least 2000 idf_ values
index = [features.index(feature) for feature in idf_features_n['features']]
tfidf_subrotine_tr = tfidf_subrotine_tr[:,index]
tfidf_subrotine_cv = tfidf_subrotine_cv[:,index]
tfidf_subrotine_te = tfidf_subrotine_te[:,index]
```

combine features

In [93]:

```
import scipy.sparse as sparse
x_merge2_tr = sparse.hstack([sparse.csr_matrix(X_train_merge2.drop(['subroutines'], axi
s=1)),tfidf_subrotine_tr])
x_merge2_cv = sparse.hstack([sparse.csr_matrix(X_cv_merge2.drop(['subroutines'], axis=1
)),tfidf_subrotine_cv])
x_merge2_te = sparse.hstack([sparse.csr_matrix(X_test_merge2.drop(['subroutines'], axis
=1)),tfidf_subrotine_te])

print("Train size",x_merge2_tr.shape)
print("Train size",x_merge2_cv.shape)
print("Train size",x_merge2_te.shape)
```

```
Train size (6955, 1019)
Train size (1739, 1019)
Train size (2174, 1019)
```

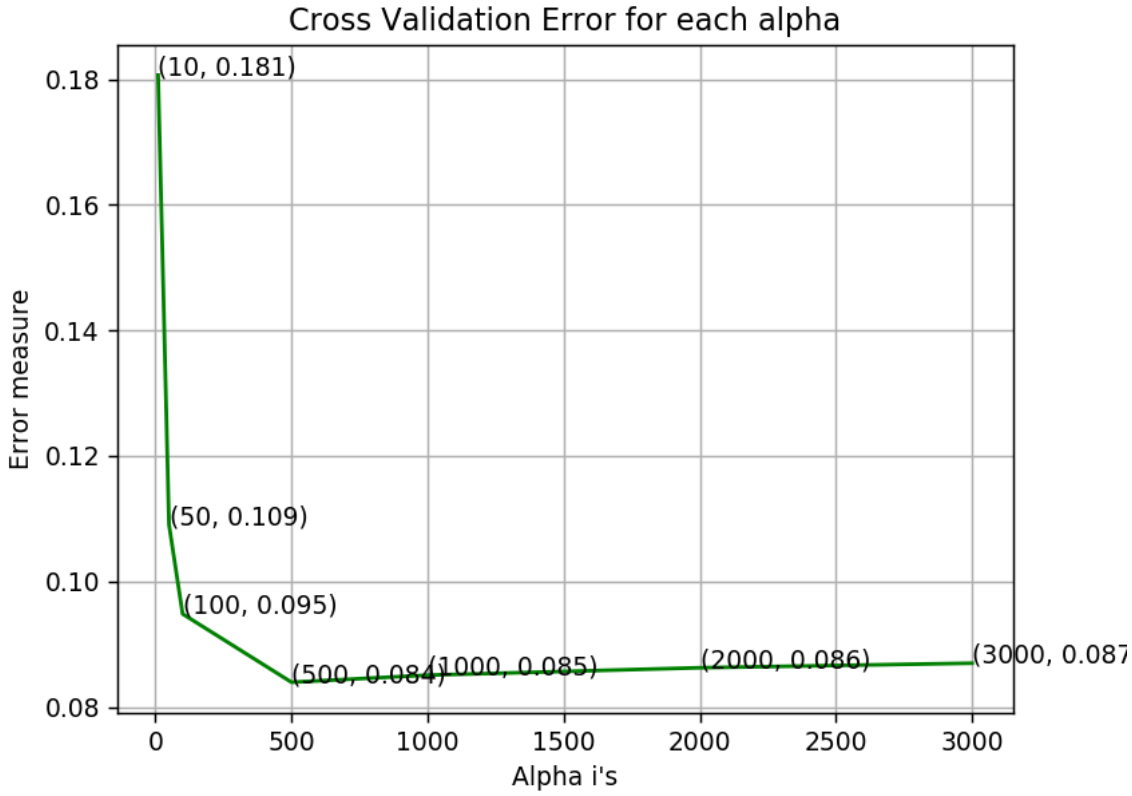
4.4.4 XgBoost Classifier

In [94]:

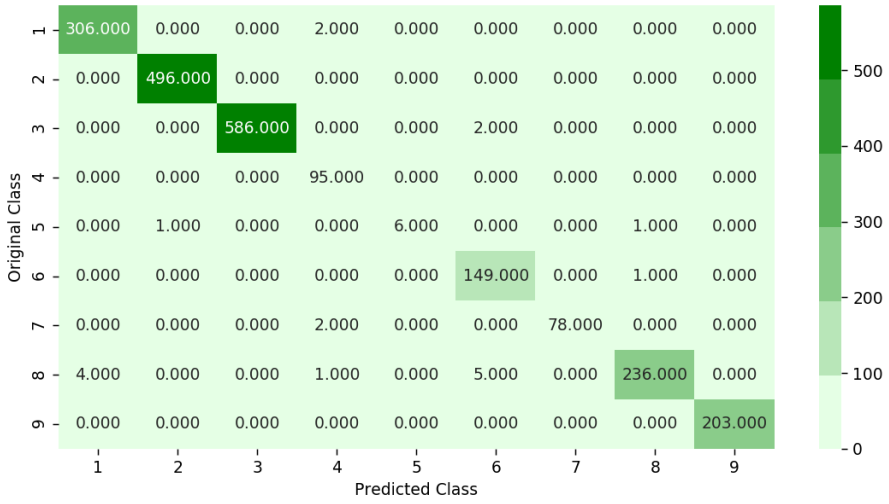
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(x_merge2_tr,y_train_merge2)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(x_merge2_tr, y_train_merge2)
    predict_y = sig_clf.predict_proba(x_merge2_cv)
    cv_log_error_array.append(log_loss(y_cv_merge2, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(x_merge2_tr,y_train_merge2)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_merge2_tr, y_train_merge2)
```

```
predict_y = sig_clf.predict_proba(x_merge2_tr)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
oss(y_train_merge2, predict_y))
predict_y = sig_clf.predict_proba(x_merge2_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv_merge2, predict_y))
predict_y = sig_clf.predict_proba(x_merge2_te)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test_merge2, predict_y))
plot_confusion_matrix(y_test_merge2,sig_clf.predict(x_merge2_te))
```

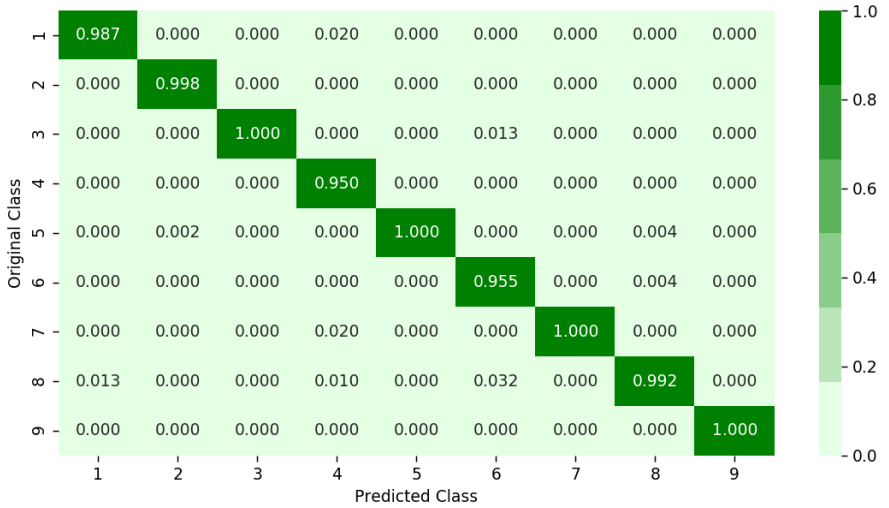
```
log_loss for c =  10 is 0.1806529562383966
log_loss for c =  50 is 0.10906659657760398
log_loss for c =  100 is 0.09483365045656128
log_loss for c =  500 is 0.08395048594620523
log_loss for c =  1000 is 0.08507814313809789
log_loss for c =  2000 is 0.08625822276555321
log_loss for c =  3000 is 0.08698300674066307
```



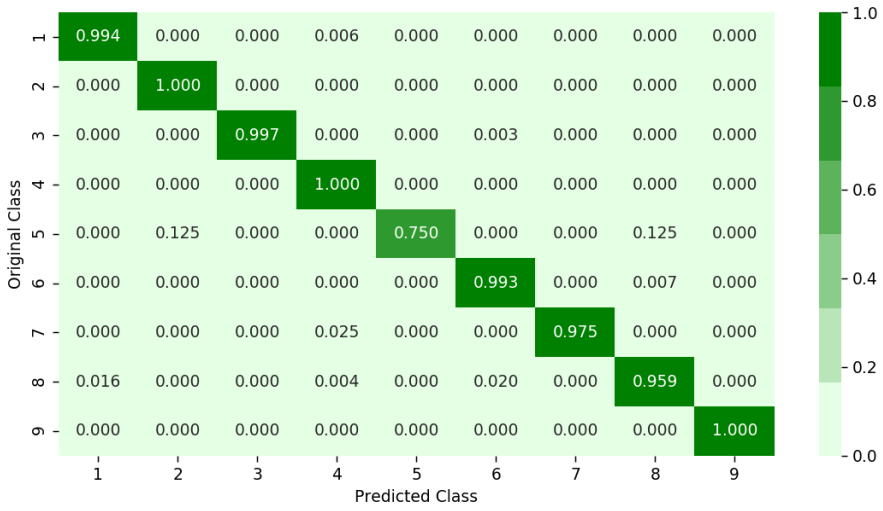

```
For values of best alpha =  500 The train log loss is: 0.03476808042415195
6
For values of best alpha =  500 The cross validation log loss is: 0.083950
48594620523
For values of best alpha =  500 The test log loss is: 0.05366273384150664
Number of misclassified points  0.8739650413983441
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



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



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



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

In [182]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1)
random_cfl.fit(x_merge2_tr, y_train_merge2)
```

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

[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done  11 out of  30 | elapsed:  1.0min remaining:
1.8min
[Parallel(n_jobs=-1)]: Done  15 out of  30 | elapsed:  1.6min remaining:
1.6min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  2.0min remaining:
1.1min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  2.8min remaining:
51.1s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  3.3min remaining:
22.0s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  3.6min finished
```

Out[182]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsam
ple_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=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], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5,
1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [183]:

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

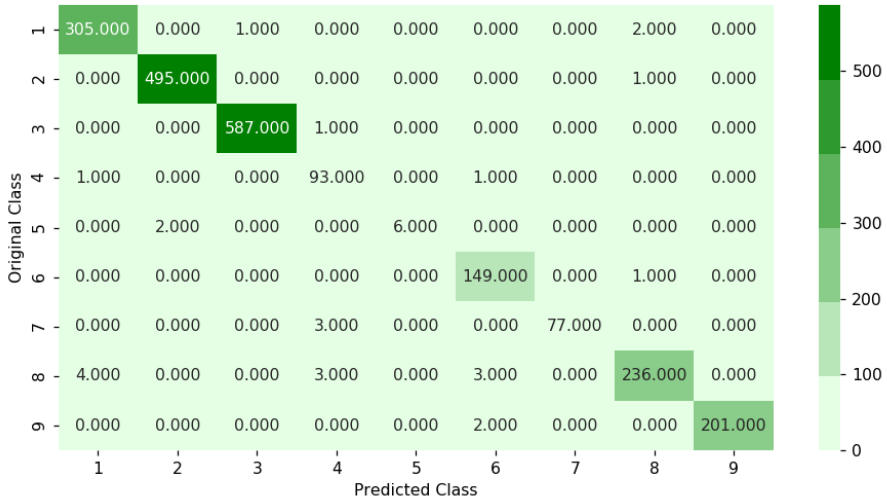
```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 10, 'learning_rate':
0.2, 'colsample_bytree': 0.5}
```

4.4.4 XgBoost Classifier with best hyperparamters

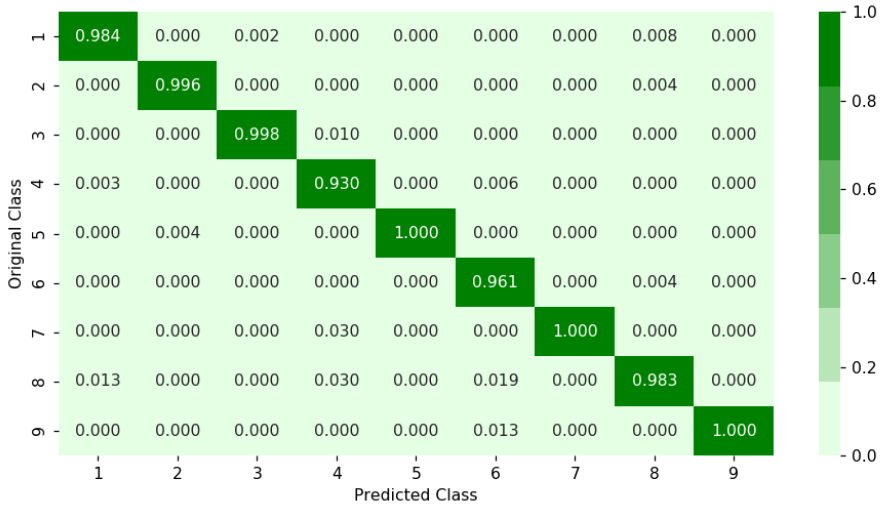
In [184]:

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

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

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.2,colsample_bytree=0.5,subsample=1,nthread=-1)
x_cfl.fit(x_merge2_tr,y_train_merge2,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_merge2_tr, y_train_merge2)

predict_y = sig_clf.predict_proba(x_merge2_tr)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge2, predict_y))
predict_y = sig_clf.predict_proba(x_merge2_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge2, predict_y))
predict_y = sig_clf.predict_proba(x_merge2_te)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge2, predict_y))
plot_confusion_matrix(y_test_merge2,sig_clf.predict(x_merge2_te))
```

```
For values of best alpha =  2000 The train log loss is: 0.0308444647803608
7
For values of best alpha =  2000 The cross validation log loss is: 0.06198
323196744861
For values of best alpha =  2000 The test log loss is: 0.06475730039107264
Number of misclassified points  1.1499540018399264
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```




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



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


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

In [98]:

```
conclusion_table.add_row(["asmgraph_asmsubroutines_merged",log_loss(y_train_merge2, pre
dict_y),log_loss(y_test_merge2, predict_y)])
```

Compute images from asm files

In [161]:

```
### This will take two long to compute so instead we used byte images as features
```

In [ ]:

```
import re
import sys
import os
from math import log
import numpy as np
import scipy as sp
from PIL import Image
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook as tqdm
def saveimg(array,name):
#     print (name)
    if array.shape[1]!=4:
        assert(False)
    b=int((array.shape[0]*4)**(0.5))
    b=2**(int(log(b)/log(2))+1)
    a=int(array.shape[0]*4/b)
    #print a,b,array.shape
    array=array[:int(a*b/4),:]
    array=np.reshape(array,(a,b))
    #print array.shape
    im = Image.fromarray(np.uint8(array))
    im.save('train/trainImageAsm/'+name+'.jpg', "BMP")
folders= ['first','second','third','fourth','fifth']
for folder in folders:

    files=os.listdir('train/asmFiles_spilt/{}/'.format(folder))
    c=0
    length = []
    for x in tqdm(files):
        # all of the asm files which highes number of bytes code per line is 4 we found
by some analysis
        # taking above decrese no. of rows per image hence decresing res. too low
        if '.asm' != x[-4:]:
            continue
        f=open('train/asmFiles_spilt/{}/'.format(folder)+x,encoding='cp1252',errors ='r
eplace')
        array=[]
        c+=1
        for row in f:

            row = row.rstrip('\r\n')  # get rid of newlines they are annoying.
            if ';' in row:
                row = row.split(';')[0] # get rid of comments they are annoying.
                #print(row)

            # get rid of all these things they are annoying.
    #         row = row.replace('short','').replace('ds:',' ')
    #         row = row.replace('dword','').replace('near','')
    #         row = row.replace('ptr','').replace(':',' ').replace(',',' ') #.replace
('??',' ')
    #         row = row.replace('@','').replace('?','')
            bytes_code = re.findall("[0-9][a-fA-F]", row)
            if len(bytes_code)!=4:
                continue
            array.append([int(i,16) if i!='??' else 0 for i in bytes_code ])
        saveimg(np.array(array),x)
        del array
        f.close()
```

Construction bytes file image features

In [41]:

```
from PIL import ImageFile
from PIL import Image

imagebyte_feature = np.zeros((10868,1000))
imagebyte_feature_file = []

# taking only startin 1000 pixels
files=os.listdir('train/trainImageBytes/')
index=0
for file in tqdm(files):
    f = open('train/trainImageBytes/'+file, "rb")
    imagebyte_feature_file.append(file.split('.')[0])
    p = ImageFile.Parser()
    while 1:
        s = f.read(1024)
        if not s:
            break
        p.feed(s)

    im = p.close()

    f.close()
    pixels = list(im.getdata())
    imagebyte_feature[index,:] = np.array(pixels[:200])
    index+=1
#     break
```

In [77]:

```
imagebyte_df = pd.DataFrame(imagebyte_feature)
imagebyte_df['ID'] = imagebyte_feature_file
imagebyte_df = pd.merge(imagebyte_df,Y,on='ID', how='left')
imagebyte_df.head()
```

Out[77]:

	0	1	2	3	4	5	6	7	8	9	...	992	993	9
0	232.0	11.0	0.0	0.0	0.0	233.0	22.0	0.0	0.0	0.0	...	0.0	131.0	250
1	199.0	1.0	36.0	4.0	92.0	0.0	233.0	214.0	74.0	0.0	...	36.0	133.0	136
2	203.0	203.0	203.0	203.0	0.0	133.0	255.0	203.0	255.0	232.0	...	0.0	203.0	203
3	106.0	255.0	104.0	163.0	22.0	0.0	16.0	100.0	161.0	0.0	...	96.0	187.0	16
4	164.0	172.0	74.0	0.0	172.0	79.0	0.0	0.0	81.0	236.0	...	164.0	47.0	241

5 rows × 1002 columns

In [81]:

```
# to csv
imagebyte_df.to_csv('imagebyte_features.csv',index=False)
```

In [83]:

```
# read csv
imagebyte_df = pd.read_csv('imagebyte_features.csv')
```

machine learning model on bytes file image features

In [85]:

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

Out[85]:

	0	1	2	3	4	5	6	7	8	9	...	990	991	9
0	232.0	11.0	0.0	0.0	0.0	233.0	22.0	0.0	0.0	0.0	...	0.0	0.0	0
1	199.0	1.0	36.0	4.0	92.0	0.0	233.0	214.0	74.0	0.0	...	81.0	32.0	36
2	203.0	203.0	203.0	203.0	0.0	133.0	255.0	203.0	255.0	232.0	...	203.0	5.0	0
3	106.0	255.0	104.0	163.0	22.0	0.0	16.0	100.0	161.0	0.0	...	236.0	4.0	96
4	164.0	172.0	74.0	0.0	172.0	79.0	0.0	0.0	81.0	236.0	...	165.0	164.0	164

5 rows × 1000 columns

In [86]:

```
X_train_image, X_test_image, y_train_image, y_test_image = train_test_split(result_x,re
sult_y ,stratify=result_y,test_size=0.20)
X_train_image, X_cv_image, y_train_image, y_cv_image = train_test_split(X_train_image,
y_train_image,stratify=y_train_image,test_size=0.20)
```

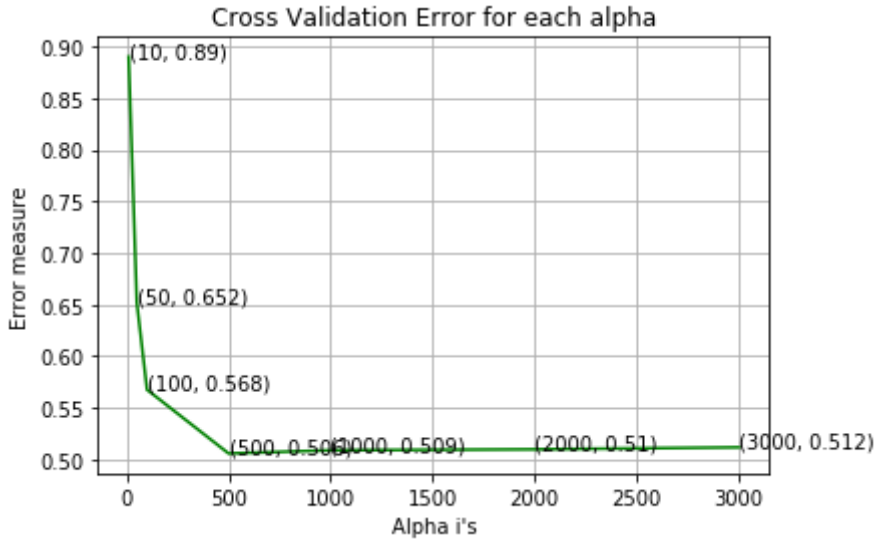
4.4.4 XgBoost Classifier

In [87]:

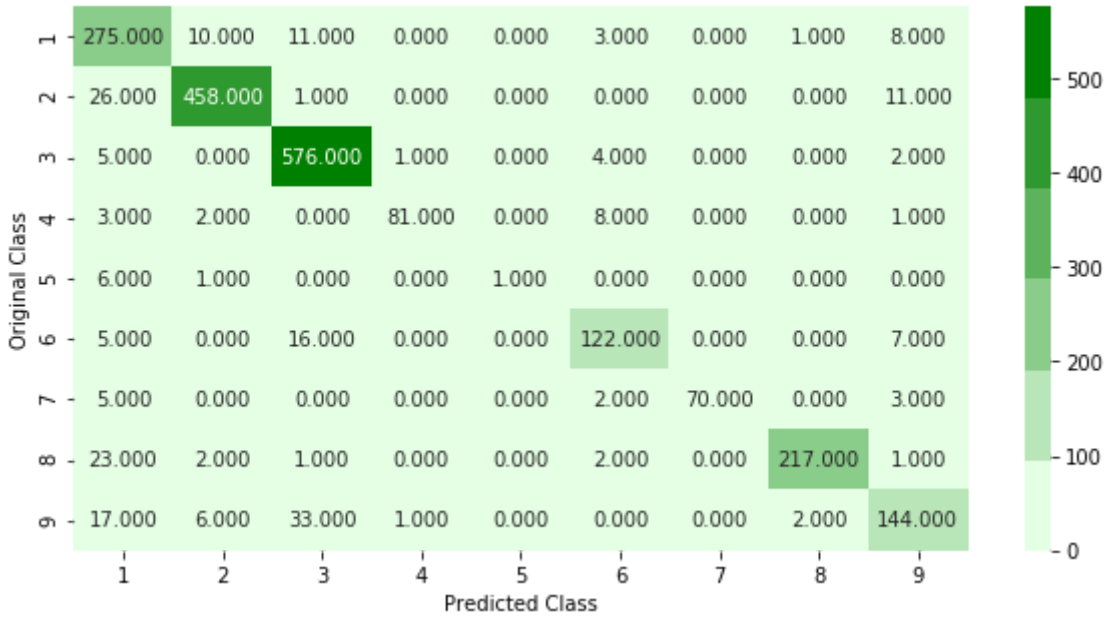
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
predict_y = sig_clf.predict_proba(X_train_image)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
oss(y_train_image, predict_y))
predict_y = sig_clf.predict_proba(X_cv_image)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv_image, predict_y))
predict_y = sig_clf.predict_proba(X_test_image)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test_image, predict_y))
plot_confusion_matrix(y_test_image,sig_clf.predict(X_test_image))
```

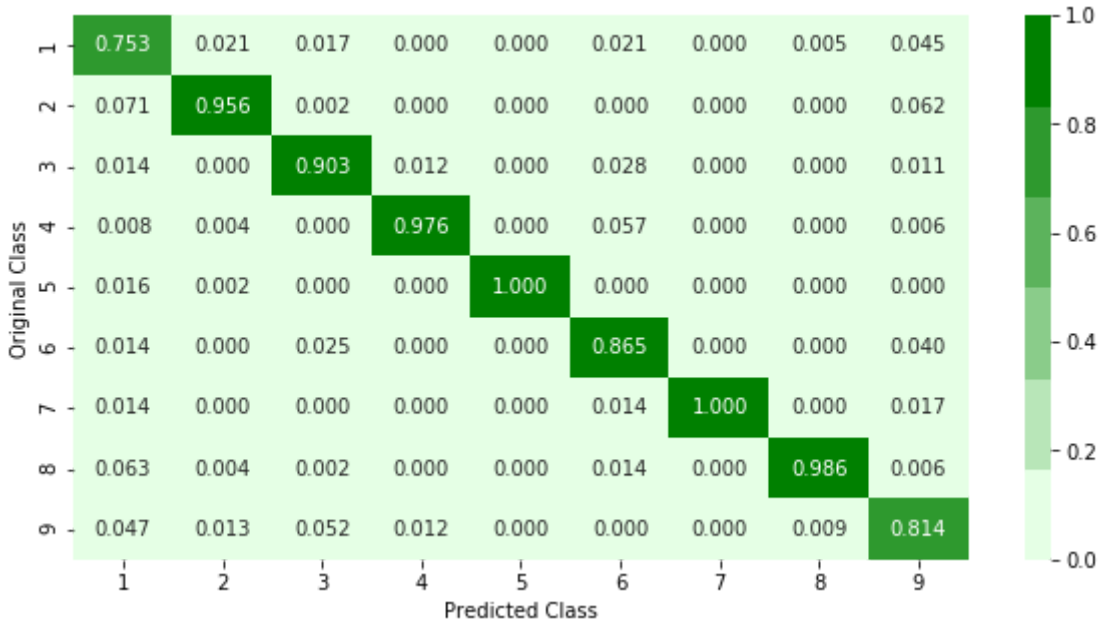
```
log_loss for c =  10 is 0.8902556035573416
log_loss for c =  50 is 0.652259004651175
log_loss for c =  100 is 0.5675560577508992
log_loss for c =  500 is 0.5056837906859262
log_loss for c =  1000 is 0.5087747851557956
log_loss for c =  2000 is 0.5096984238120955
log_loss for c =  3000 is 0.5116276946327601
```




```
For values of best alpha =  500 The train log loss is: 0.16743346108907153
For values of best alpha =  500 The cross validation log loss is: 0.505683
7906859262
For values of best alpha =  500 The test log loss is: 0.4391914125132858
Number of misclassified points  10.579576816927323
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```

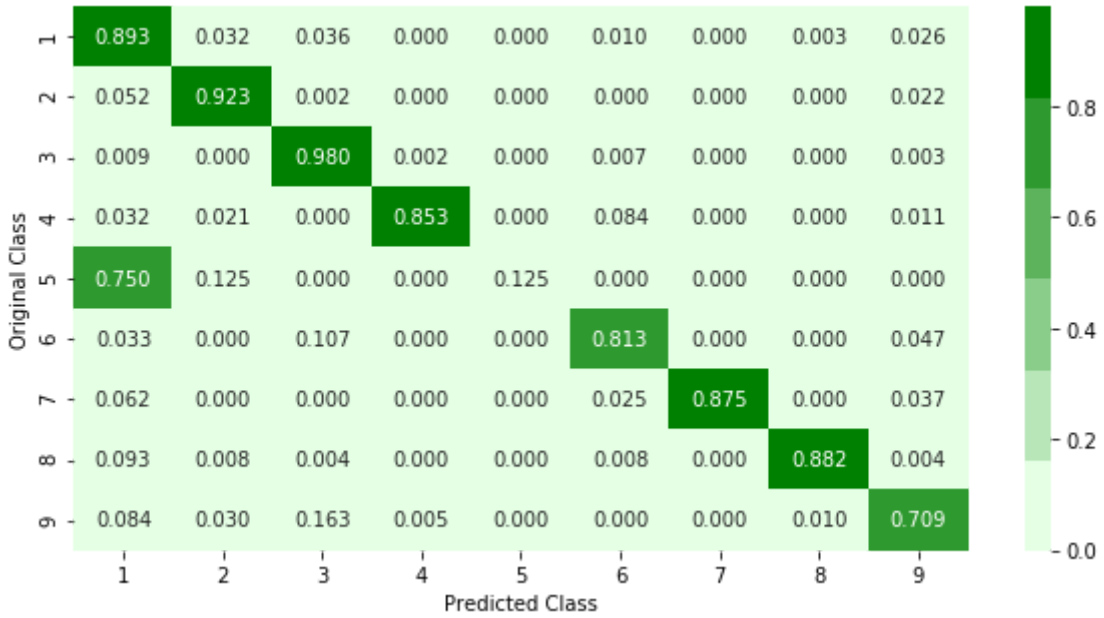



```
---------------------------------------------------- Precision matrix ------
-----------------------------------------
```



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

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



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

In [ ]:

```
x_cfl=XGBClassifier()

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

In [ ]:

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

In [ ]:

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

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

x_cfl=XGBClassifier(n_estimators=2000,max_depth=5,learning_rate=0.15,colsample_bytree=1,subsample=0.3,nthread=-1)
x_cfl.fit(X_train_image,y_train_image,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_image, y_train_image)

predict_y = sig_clf.predict_proba(X_train_image)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_image, predict_y))
predict_y = sig_clf.predict_proba(X_cv_image)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_image, predict_y))
predict_y = sig_clf.predict_proba(X_test_image)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_graph, predict_y))
plot_confusion_matrix(y_test_image,sig_clf.predict(X_test_image))
```

In [ ]:

```
conclusion_table.add_row(["byte_image",log_loss(y_train_image, predict_y),log_loss(y_test_graph, predict_y)])
```

Machine learning model on both asm_graph and asm_subroutines features

merging both asm_graph and asm_subroutines features

In [18]:

```
result.head()
```

Out[18]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.0
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0

5 rows × 260 columns

In [19]:

```
result_asm.head()
```

Out[19]:

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

5 rows × 54 columns

In [20]:

```
result_asm_graph.head()
```

Out[20]:

	ID	has_program_entry_point	nodes	edges	avg_in_degree	a
0	01azqd4InC7m9JpocGv5	0	0.010336	0.047084	0.769626	
1	01IsoiSMh5gxyDYTI4CB	0	0.008469	0.007154	0.142723	
2	01jsnpXSAlgw6aPeDxrU	0	0.001600	0.002472	0.260980	
3	01kcPWA9K2BOxQeS5Rju	0	0.002734	0.001461	0.090271	
4	01SuzwMJEIXsK7A8dQbl	0	0.006402	0.004795	0.126533	

5 rows × 21 columns

In [21]:

```
result_asm_subroutines.head()
```

Out[21]:

	ID	subroutines	Class
0	01azqd4InC7m9JpocGv5	atexit findwindowa isvalidlocale virtualalloc ...	9
1	01IsoiSMh5gxyDYTl4CB	3yaxpaxz eax 2yapaxiz 0exceptionstdqaeabqbdz c...	2
2	01jsnpXSAlgw6aPeDxrU	virtualalloc loadlibrarya 545279 winmain16 3	9
3	01kcPWA9K2BOxQeS5Rju	0basicstringguchartraitsgstdvallocatorg2stdqae...	1
4	01SuzwMJEIXsK7A8dQbl	ecx exitprocess ebp+var10 processtrace createt...	8

In [22]:

```
print(result.shape)
print(result_asm.shape)
print(result_asm_subroutines.shape)
print(result_asm_graph.shape)
print("Total features we have here are: ", 330)
```

```
(10868, 260)
(10868, 54)
(10868, 3)
(10868, 21)
Total features we have here are:  330
```

In [23]:

```
result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='left')
result_x = pd.merge(result_x,result_asm_graph.drop(['Class'], axis=1),on='ID', how='lef
t')
result_x = pd.merge(result_x,result_asm_subroutines.drop(['Class'], axis=1),on='ID', ho
w='left')
# result_x = pd.merge(result_x,imagebyte_df.drop(['Class'], axis=1),on='ID', how='lef
t')
result_y = result_x['Class']
result_x = result_x.drop(['Class','ID'], axis=1)
result_x.head()
```

Out[23]:

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

5 rows × 330 columns

In [24]:

```
X_train_merge3, X_test_merge3, y_train_merge3, y_test_merge3 = train_test_split(result_
x,result_y ,stratify=result_y,test_size=0.20)
X_train_merge3, X_cv_merge3, y_train_merge3, y_cv_merge3 = train_test_split(X_train_mer
ge3, y_train_merge3,stratify=y_train_merge3,test_size=0.20)
```

Tfidf with least 2000 idf_ values

In [26]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer =  TfidfVectorizer(min_df=0.0015,max_features=300,binary='false')
tfidf_subrotine_tr = vectorizer.fit_transform(X_train_merge3['subroutines'])
print("Number of features:",len(vectorizer.get_feature_names()))
```

```
Number of features: 300
```

In [27]:

```
tfidf_subrotine_cv = vectorizer.transform(X_cv_merge3['subroutines'])
tfidf_subrotine_te = vectorizer.transform(X_test_merge3['subroutines'])
```

find least idf_values

In [28]:

```
idf_features = pd.DataFrame(list(zip(vectorizer.get_feature_names(),vectorizer.idf_)),
columns=['features','idf'])
```

In [29]:

```
features = vectorizer.get_feature_names()
idf_features_n = idf_features.sort_values(['idf'], ascending=True)[:1000]
idf_features_n.head()
```

Out[29]:

	features	idf
71	eax	1.454823
75	edi	1.620252
84	esi	1.739640
187	loadlibrarya	1.838662
72	ebp	1.936036

filter train, test , cv with onlt selected idf values features

In [30]:

```
# Only Tfidf with least 2000 idf_ values
index = [features.index(feature) for feature in idf_features_n['features']]
tfidf_subrotine_tr = tfidf_subrotine_tr[:,index]
tfidf_subrotine_cv = tfidf_subrotine_cv[:,index]
tfidf_subrotine_te = tfidf_subrotine_te[:,index]
```

combine features

In [31]:

```
import scipy.sparse as sparse
x_merge3_tr = sparse.hstack([sparse.csr_matrix(X_train_merge3.drop(['subroutines'], axi
s=1)),tfidf_subrotine_tr])
x_merge3_cv = sparse.hstack([sparse.csr_matrix(X_cv_merge3.drop(['subroutines'], axis=1
)),tfidf_subrotine_cv])
x_merge3_te = sparse.hstack([sparse.csr_matrix(X_test_merge3.drop(['subroutines'], axis
=1)),tfidf_subrotine_te])

print("Train size",x_merge3_tr.shape)
print("Train size",x_merge3_cv.shape)
print("Train size",x_merge3_te.shape)
```

```
Train size (6955, 629)
Train size (1739, 629)
Train size (2174, 629)
```

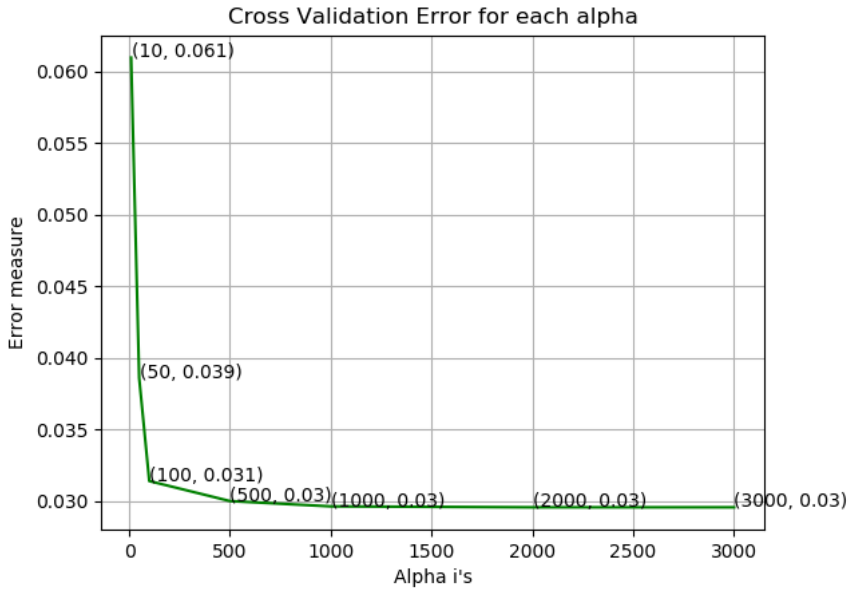
4.4.4 XgBoost Classifier

In [32]:

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

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

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(x_merge3_tr,y_train_merge3)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(x_merge3_tr, y_train_merge3)
    predict_y = sig_clf.predict_proba(x_merge3_cv)
    cv_log_error_array.append(log_loss(y_cv_merge3, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(x_merge3_tr,y_train_merge3)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_merge3_tr, y_train_merge3)
```

```
predict_y = sig_clf.predict_proba(x_merge3_tr)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
oss(y_train_merge3, predict_y))
predict_y = sig_clf.predict_proba(x_merge3_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv_merge3, predict_y))
predict_y = sig_clf.predict_proba(x_merge3_te)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test_merge3, predict_y))
plot_confusion_matrix(y_test_merge3,sig_clf.predict(x_merge3_te))
```

```
log_loss for c =  10 is 0.060981690631279936
log_loss for c =  50 is 0.03860874543048204
log_loss for c =  100 is 0.031418154520838064
log_loss for c =  500 is 0.030021070406249534
log_loss for c =  1000 is 0.029649914863680793
log_loss for c =  2000 is 0.029584385338149954
log_loss for c =  3000 is 0.029584602501473007
```



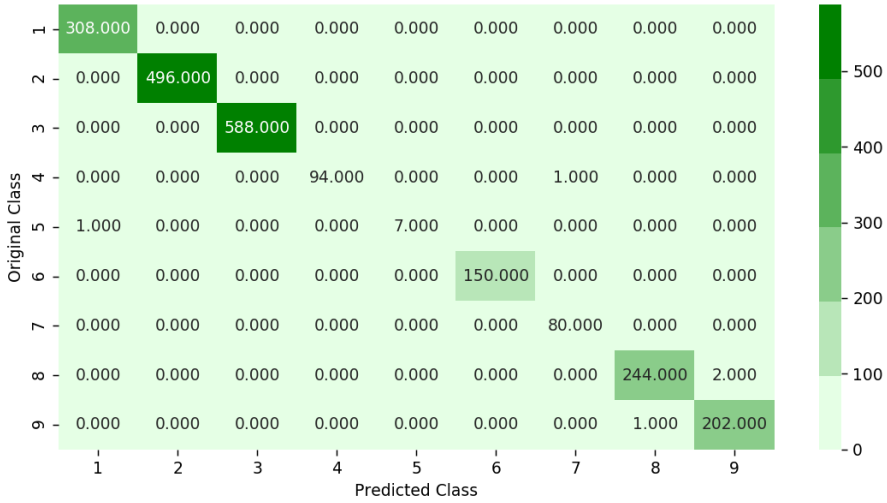

```
For values of best alpha =  2000 The train log loss is: 0.0119484274067644
98
For values of best alpha =  2000 The cross validation log loss is: 0.02958
4385338149954
For values of best alpha =  2000 The test log loss is: 0.02105046593293742
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-32-1835c1eb8d37> in <module>
     56 predict_y = sig_clf.predict_proba(x_merge3_te)
     57 print('For values of best alpha = ', alpha[best_alpha], "The test
 log loss is:",log_loss(y_test_merge3, predict_y))
---> 58 plot_confusion_matrix(y_test_merge3,sig_clf.predict(x_merge3_te))

NameError: name 'plot_confusion_matrix' is not defined
```

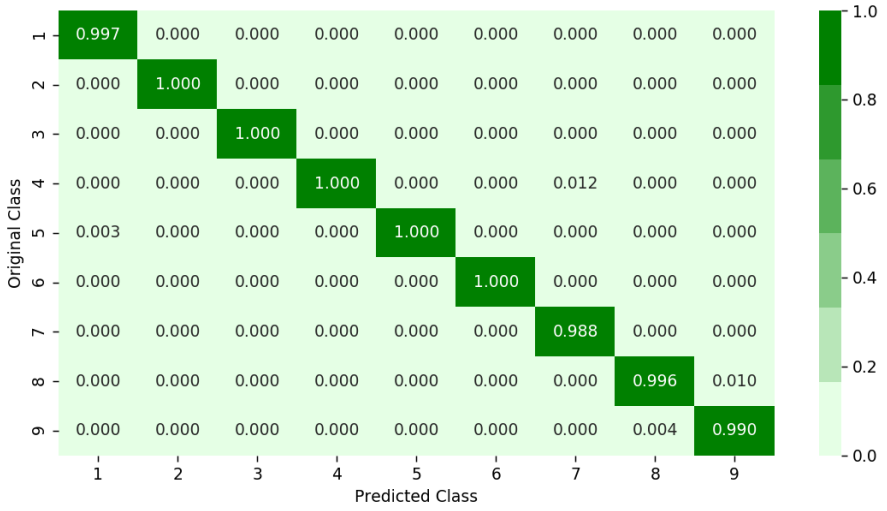
In [34]:

```
plot_confusion_matrix(y_test_merge3,sig_clf.predict(x_merge3_te))
```

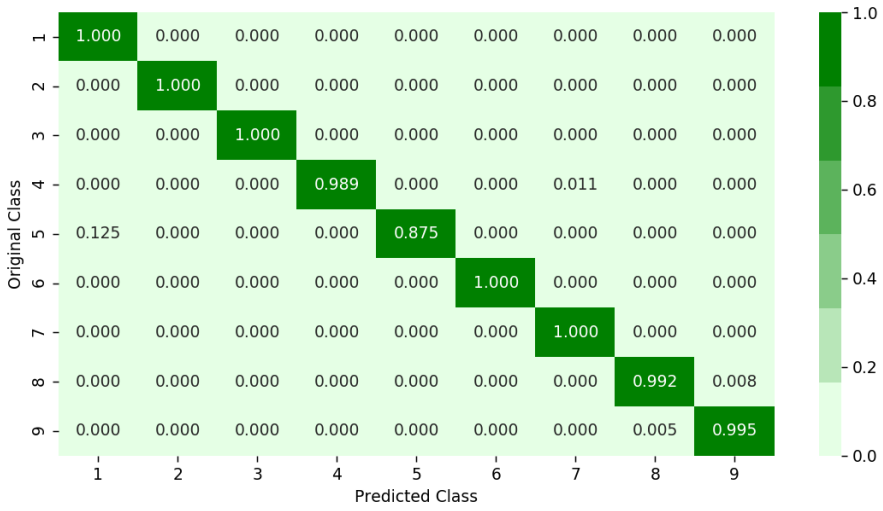
```
Number of misclassified points  0.22999080036798528
---------------------------------------------------- Confusion matrix ------
------------------------------------------
```




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




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



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

In [36]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(x_merge3_tr, y_train_merge3)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done  11 out of  30 | elapsed:  5.6min remaining:
9.6min
[Parallel(n_jobs=-1)]: Done  15 out of  30 | elapsed: 11.5min remaining: 1
1.5min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed: 19.6min remaining: 1
1.3min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 21.0min remaining:
6.4min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 27.9min remaining:
3.1min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 45.7min finished
```

Out[36]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtre
e',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_st
ep=0,
                                           max_depth=3, min_child_weight=
1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsamp
le=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.
5, 1],
                                        'learning_rate': [0.01, 0.03, 0.0
5, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 10
00,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [37]:

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

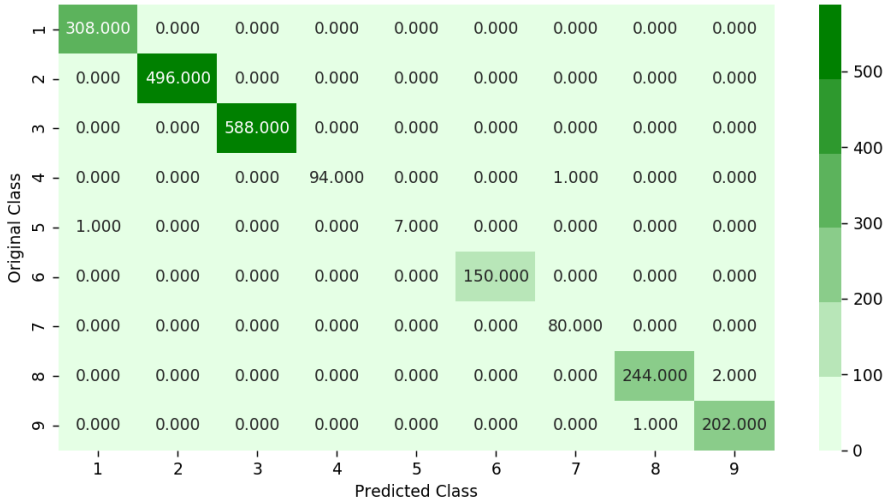
```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.
15, 'colsample_bytree': 1}
```

4.4.4 XgBoost Classifier with best hyperparamters

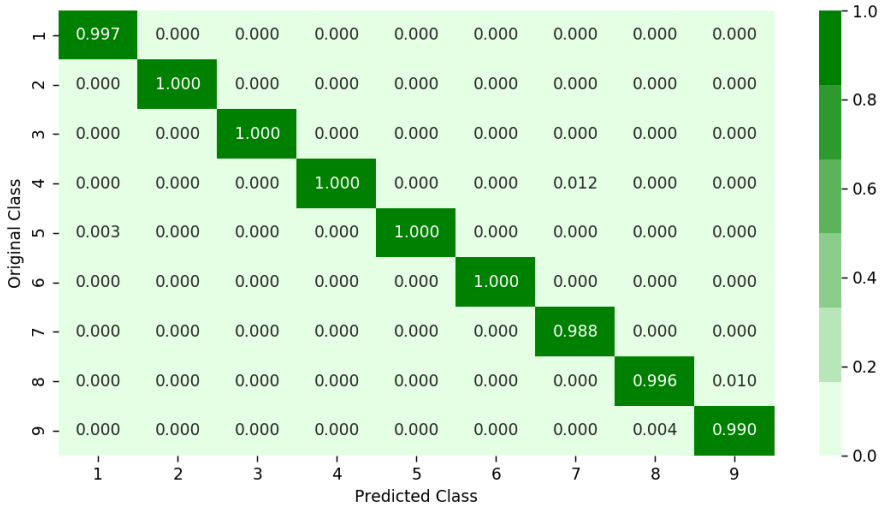
In [39]:

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

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

x_cfl=XGBClassifier(n_estimators=2000,max_depth=3,learning_rate=0.15,colsample_bytree=1,subsample=1,nthread=-1)
x_cfl.fit(x_merge3_tr,y_train_merge3,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_merge3_tr, y_train_merge3)

predict_y = sig_clf.predict_proba(x_merge3_tr)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge3, predict_y))
predict_y = sig_clf.predict_proba(x_merge3_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge3, predict_y))
predict_y = sig_clf.predict_proba(x_merge3_te)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge3, predict_y))
plot_confusion_matrix(y_test_merge3,sig_clf.predict(x_merge3_te))
```

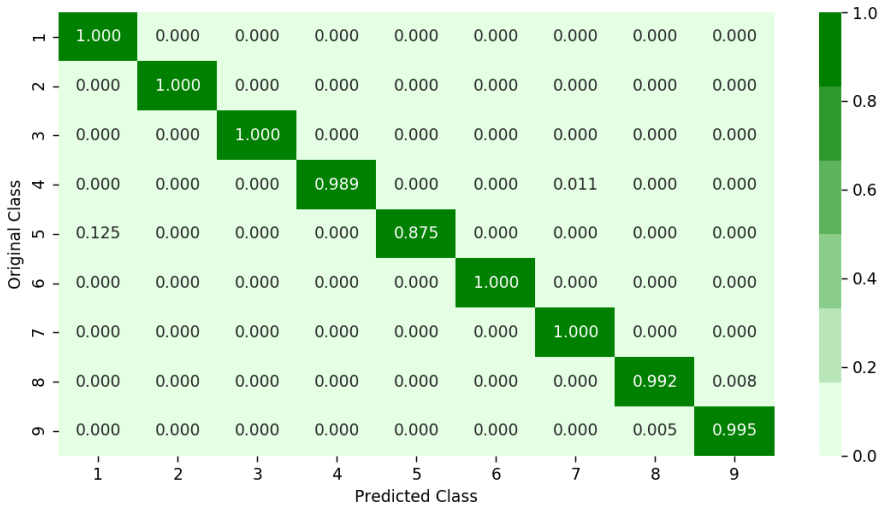
```
For values of best alpha =  2000 The train log loss is: 0.0119417539713528
47
For values of best alpha =  2000 The cross validation log loss is: 0.02978
5713218488885
For values of best alpha =  2000 The test log loss is: 0.02022262164892204
4
Number of misclassified points  0.22999080036798528
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```




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




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



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

In [ ]:

```
conclusion_table.add_row(["all_features",log_loss(y_train_merge2, predict_y),log_loss(y_test_merge2, predict_y)])
```

In [51]:

```
print(conclusion_table)
```

```
+--------------------------------+----------------------+----------------
-----+
|     feature_name(Xgboost)      |    train_Logloss     |     Test_Logloss
|
+--------------------------------+----------------------+----------------
-----+
|           bytes_Bow            |    0.022540976086    |   0.078268858709
8    |
|           asm_codes            |   0.0102661325822    |   0.048390876439
7    |
|    bytesbow_asmcodes_merged    |   0.0121922832297    |   0.031704113244
2    |
|           asm_Graph            | 0.061729138713408044 |  0.1618956619696
228  |
|        asm_subroutines         | 0.05531057965840496  | 0.09410721669170
027  |
| asmgraph_asmsubroutines_merged | 0.03084446478036087  | 0.06475730039107
264  |
|       combined[1,2,4,5]        | 0.011941753971352847 | 0.02022262164892
2044 |
+--------------------------------+----------------------+----------------
-----+
```

In [ ]:

```
# Conclusion

# Apart from using bytes_bow and asm_codes features in this case study i performed some
more deep and complex analysis to
# acquire much lower log loss using xgboost with in total of ~600 features

# note - i have not used image feature for asm file as it took very long time to comput
e and didnot use image feature because
# it will add more complexity to the xgboost and the code will take much longer to run,
it possible to drop as low as
# 0.005 logloss with image features and asm_external_subroutines feature with around 10
00 features

# --- asmg_graph and asm_subroutines ----

# What i have done apart from bytes_bow and asm_codes

# 1) Used networkx to create nodes from asm call_graph and handcrafted different graph
 features

# 2) Used asm external subroutines with tfidf(max_feaures =300 only)

# What have i not included in conclusion

# Too achieve much less log loss apart from using image features

# --- local_subroutines ----
# we can use local subroutines similarities i.e programs with similar subroutinees appe
ars to be same class
# but the problem with local subroutines are that same subroutines from different progr
ams have same name
# therefore for this we have to callulate similarity matrix first using 5 different met
hods explained in this paper ->
#http://www.cs.sjsu.edu/faculty/stamp/papers/topics/topic5/10.1007-s11416-012-0175-y.pd
f

# Compute similarity matrix requires much much time, aside from this i could not find a
ny kernel which provided similarity
# pre-computed for this dataset so , i dropped this method

# Note - i achieved log loss of 0.02 without using any of two image features

# refrences

#https://pdfs.semanticscholar.org/8db2/69106ea6e1f59e4dac0889665dd3336ee9b1.pdf

#http://www.cs.sjsu.edu/faculty/stamp/papers/topics/topic5/10.1007-s11416-012-0175-y.pd
f

#https://www.quora.com/What-is-the-difference-between-opcode-operand-and-instruction

#graph similarity features https://sites.cs.ucsb.edu/~xyan/papers/tods06_similarity.pdf
```