Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

malware attack is to **identify whether a given piece of file/software is a malware.**

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning

Problem

2.1. Data

2.1.1. Data Overview

- **Source : https://www.kaggle.com/c/malware-classification/data**
- **For every malware, we have two files**
  1. **.asm file (read more: https://www.reviversoft.com/file-extensions/asm)**
  2. **.bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)**
- **Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:**
- **Lots of Data for a single-box/computer.**
- **There are total 10,868 .bytes files and 10,868 asm files total 21,736 files**
- **There are 9 types of malwares (9 classes) in our give data**
- **Types of Malware:**
  1. **Ramnit**
  2. **Lollipop**
  3. **Kelihos_ver3**
  4. **Vundo**
  5. **Simda**
  6. **Tracur**
  7. **Kelihos_ver1**
  8. **Obfuscator.ACY**
  9. **Gatak**

2.1.2. Example Data Point

.asm file

```
.text:00401000
assume es:nothing, ss:nothing, ds:_data,
fs:nothing, gs:nothing
.text:00401000 56
push    esi
.text:00401001 8D 44 24 08
lea     eax, [esp+8]
.text:00401005 50
push    eax
.text:00401006 8B F1
mov     esi, ecx
.text:00401008 E8 1C 1B 00 00
call    ??0exception@std@@QAE@ABQBD@Z ;
std::exception::exception(char const * c
onst &)
.text:0040100D C7 06 08 BB 42 00
mov     dword ptr [esi], offset off_42BB
08
.text:00401013 8B C6
mov     eax, esi
.text:00401015 5E
pop     esi
.text:00401016 C2 04 00
retn    4
.text:00401016
; ---------------------------------------
---------------------------------------
.text:00401019 CC CC CC CC CC CC CC
align 10h
.text:00401020 C7 01 08 BB 42 00
mov     dword ptr [ecx], offset off_42BB
08
.text:00401026 E9 26 1C 00 00
jmp     sub_402C51
.text:00401026
; ---------------------------------------
```

```
-------------------------------------
.text:0040102B CC CC CC CC CC
align 10h
.text:00401030 56
push    esi
.text:00401031 8B F1
mov     esi, ecx
.text:00401033 C7 06 08 BB 42 00
mov     dword ptr [esi], offset off_42BB
08
.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 delet
e(void *)
.text:0040104B 83 C4 04
add     esp, 4
.text:0040104E
.text:0040104E
loc_40104E:                 ; CODE XREF:
.text:00401043j
.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 0
0 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8
```

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

```
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6
E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C
8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C
8 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

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/AA
dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is us
ed 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 RandomizedS
earchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClass
ifierCV
from sklearn.neighbors import KNeighborsClassif
ier
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 LogisticRegres
sion
```

```
from sklearn.ensemble import RandomForestClassifier
```

In [3]:

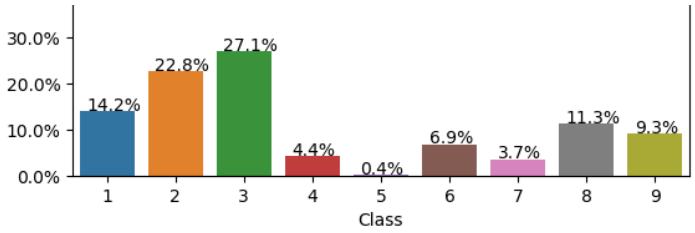
```
#separating byte files and asm files

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

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

# if we have folder called 'train' (train folder contains both .asm files and .bytes files) 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):
    data_files = os.listdir(source)
    for file in data_files:
        #print(file)
        if (file.endswith("bytes")):
            shutil.move(source+'\\'+file,destination_1)
        if (file.endswith("asm")):
```

```
            shutil.move(source+'\\'+file,destin
ation_2)
```

3.1. Distribution of malware classes in whole data set

In [2]:

```
Y=pd.read_csv(r"C:\Users\hp\Desktop\AppliedAI\i
python notes\microsoftmalware_data\trainLabels.
csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_
height()/total), (p.get_x()+0.1, p.get_height()
+5))

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

#adjust the ticklabel to the desired format, wi
thout changing the position of the ticks.
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 [3]:

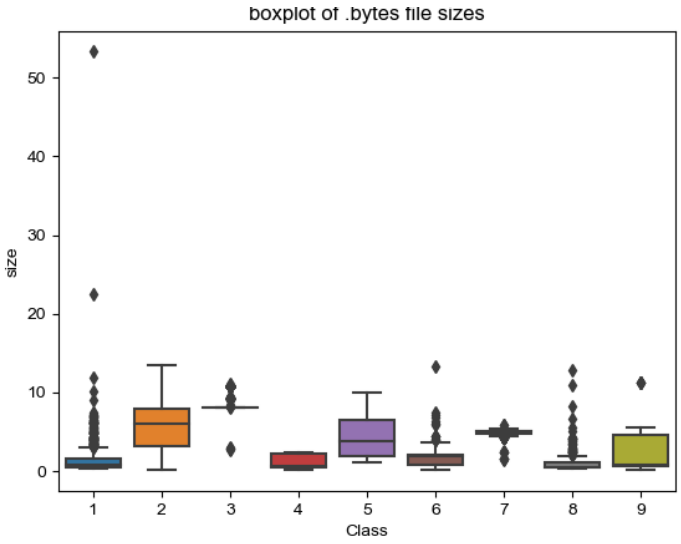
```
files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, 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('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
```

```
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

	ID	size	Class
0	01azqd4InC7m9JpocGv5	5.012695	9
1	01IsoiSMh5gxyDYTl4CB	6.556152	2
2	01jsnpXSAlgw6aPeDxrU	4.602051	9
3	01kcPWA9K2BOxQeS5Rju	0.679688	1
4	01SuzwMJEIXsK7A8dQbl	0.438965	8

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

In [4]:
```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



3.2.3 feature extraction from byte files

In [5]:

```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 0
0 C7 06 08
#-------------------
#we remove the starting address 00401000

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        file=file.split('.')[0]
        text_file = open('byteFiles/'+file+".tx
t", 'w+')
        file = file+'.bytes'
        with open('byteFiles/'+file,"r") as fp:
            lines=""
            for line in fp:
```

```
                a=line.rstrip().split(" ")[1:]
                b=' '.join(a)
                b=b+"\n"
                text_file.write(b)
            fp.close()
            os.remove('byteFiles/'+file)
        text_file.close()

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

In [7]:

```
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,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 files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byt
e_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(h
ex_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 [19]:

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

Out[19]:

	Unnamed: 0	ID	0	
0	0	01azqd4InC7m9JpocGv5	601905	390
1	1	01IsoiSMh5gxyDYTl4CB	39755	833

	Unnamed: 0	ID	0	
2	2	01jsnpXSAlgw6aPeDxrU	93506	954
3	3	01kcPWA9K2BOxQeS5Rju	21091	121
4	4	01SuzwMJEIXsK7A8dQbl	19764	71

5 rows × 263 columns

In [4]:
```
byte_features=pd.read_csv("result.csv")
print(byte_features.head())
```

```
   Unnamed: 0                   ID
0           0  01azqd4InC7m9JpocGv5  6019
05  3905  2816  3832  3345  3242
1           1  01IsoiSMh5gxyDYTl4CB   397
55  8337  7249  7186  8663  6844
2           2  01jsnpXSAlgw6aPeDxrU   935
06  9542  2568  2438  8925  9330
3           3  01kcPWA9K2BOxQeS5Rju   210
91  1213   726   817  1257   625
4           4  01SuzwMJEIXsK7A8dQbl   197
64   710   302   433   559   410

      6     7  ...    f9    fa    fb
fc    fd    fe    ff    ??  \
0  3650  3201  ...  3101  3211  3097  2
758  3099  2759  5753  1824
1  8420  7589  ...   439   281   302  7
639   518  17001  54902  8588
2  9007  2342  ...  2242  2885  2863  2
471  2786  2680  49144   468
3   550   523  ...   485   462   516  1
133   471   761  7998  13940

4   262   249  ...   350   209   239
```

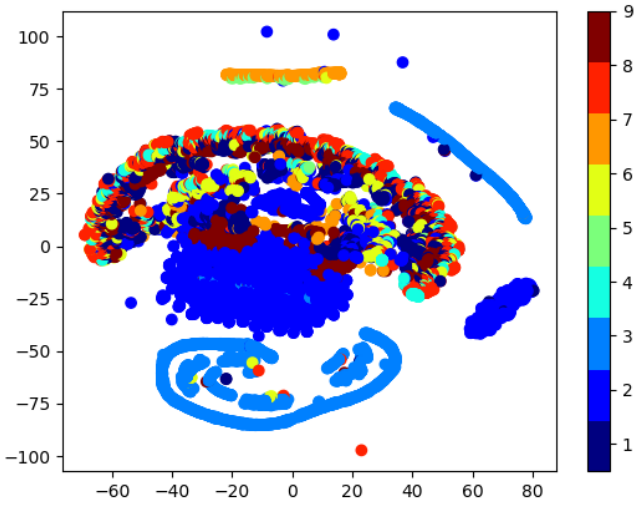
```
653   221    242   2199   9008

       size  Class
0  5.012695      9
1  6.556152      2
2  4.602051      9
3  0.679688      1
4  0.438965      8

[5 rows x 261 columns]
```

In [5]:
```
result=byte_features
# 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_value)
    return result1
result = normalize(result)
```

3.2.4 Multivariate Analysis

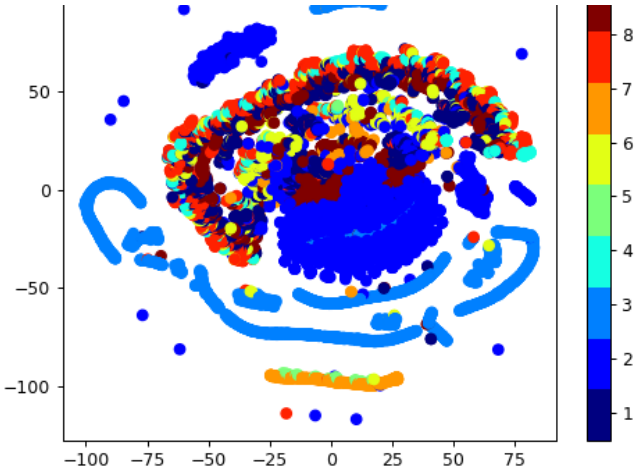
In [11]:
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID',
'Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
```

```
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 [15]:

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



Train Test split

In [9]:

```
data_y = result['Class']
# split the data into test and train by maintai
ning same distribution of output varaible 'y_tr
ue' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_s
plit(result.drop(['ID','Class'], axis=1), data_
y,stratify=data_y,test_size=0.20)
# split the train data into train and cross val
idation by maintaining same distribution of out
put 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 [10]:

```
print('Number of data points in train data:', X
_train.shape[0])
print('Number of data points in test data:', X_
test.shape[0])
print('Number of data points in cross validatio
n data:', X_cv.shape[0])
```

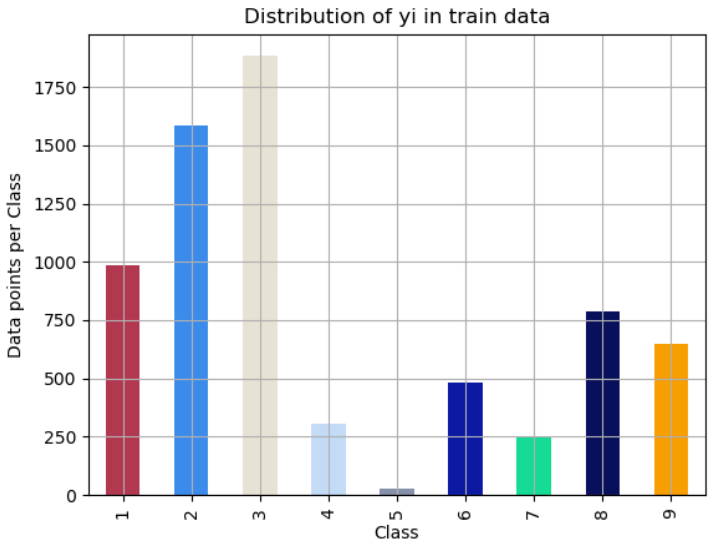
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [11]:

```
# it returns a dict, keys as class labels and values as the number of data points in that 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 = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95', '#080f5b', '#f79e02']
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], '(',
```

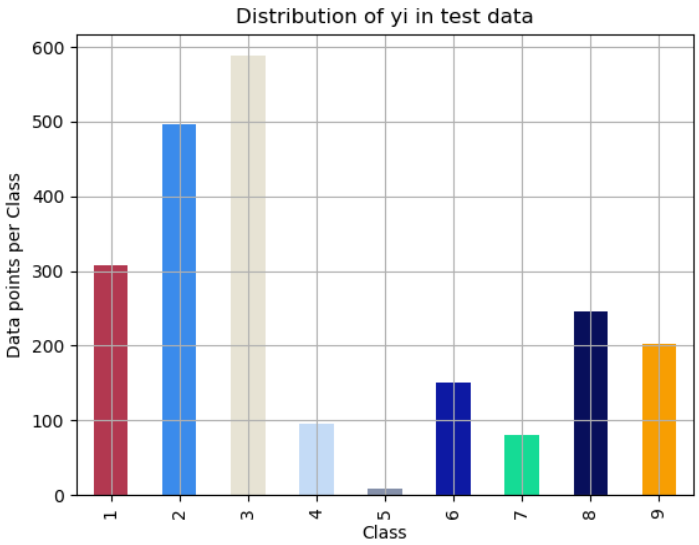
```
np.round((train_class_distribution.values[i]/y_
train.shape[0]*100), 3), '%)')


print('-'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4',
'#c4dbf6', '#8590aa', '#0d19a3', '#15db95', '#0
80f5b', '#f79e02']
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/nump
y/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], '(', n
p.round((test_class_distribution.values[i]/y_te
st.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4',
'#c4dbf6', '#8590aa', '#0d19a3', '#15db95', '#0
80f5b', '#f79e02']
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 validati
on data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/nump
y/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus
sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distributio
n.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.sha
pe[0]*100), 3), '%)')
```



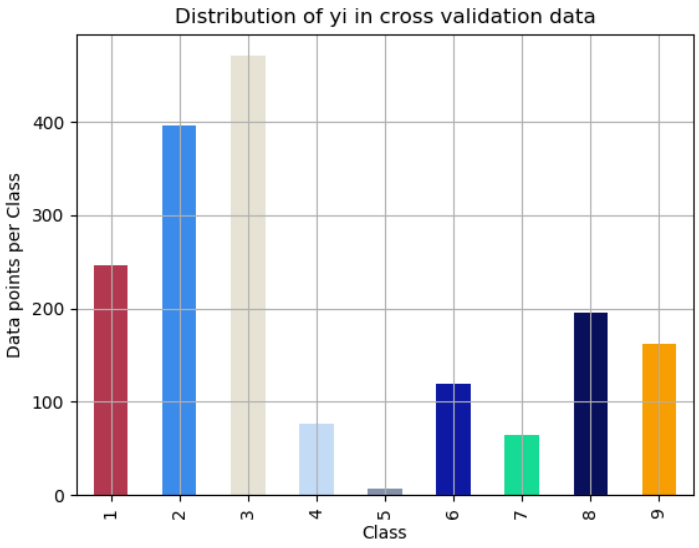

```
Number of data points in class 3 : 1883
( 27.074 %)
Number of data points in class 2 : 1586
( 22.804 %)
Number of data points in class 1 : 986
```

```
( 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 [13]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(le
n(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represent
s number of points of class i are predicted cla
ss 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 col
umns and axis=1 corresponds to rows in two diam
ensional 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 col
umns and axis=1 corresponds to rows in two diam
ensional 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=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

```
B.sum(axis=0))

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

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

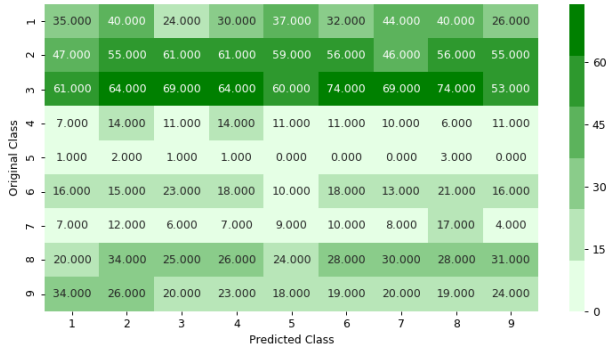
4.1.1. Random Model

In [20]:

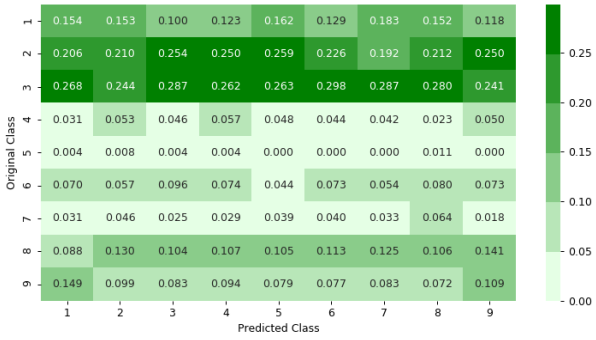
```
# we need to generate 9 numbers and the sum of
 numbers should be 1
# one solution is to genarate 9 numbers and div
ide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/408
4039

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

# we create a output array that has exactly sam
```

```
e 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(ra
nd_probs)))[0])
print("Log loss on Cross Validation Data using
 Random Model",log_loss(y_cv,cv_predicted_y, ep
s=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 Mode
l",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 R
andom Model 2.4987116946656167
Log loss on Test Data using Random Model
2.4553327958473936
Number of misclassified points  88.454461
82152715
---------------------------------------
--------- Confusion matrix --------------
---------------------------------------
```

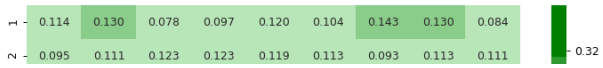


---------- Precision matrix ----------



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. 1.]
```

4.1.2. K Nearest Neighbour Classification

In [21]:

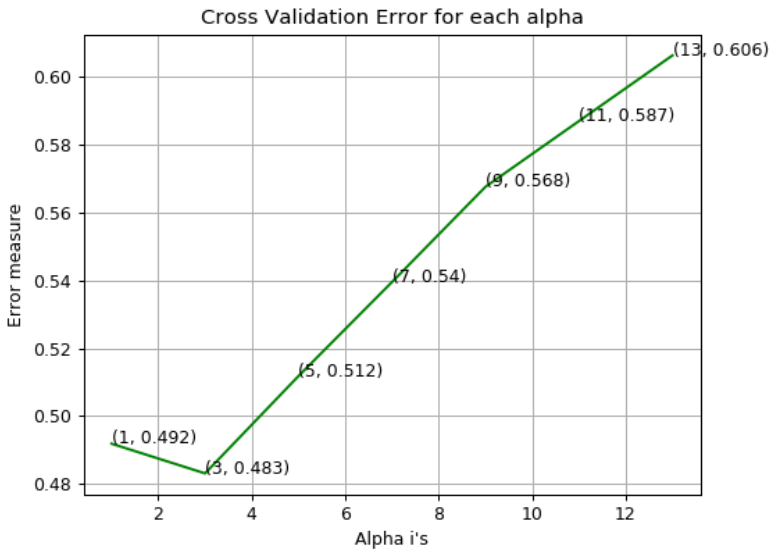
```
# find more about KNeighborsClassifier() here h
ttp://scikit-learn.org/stable/modules/generate
d/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights
='uniform', algorithm='auto', leaf_size=30, p=
2,
# metric='minkowski', metric_params=None, n_job
s=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 p
rovided data
# predict_proba(X):Return probability estimates
for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/c
ourse/applied-ai-course-online/lessons/k-neares
t-neighbors-geometric-intuition-with-a-toy-exam
ple-1/
#-----------------------------------
```

```
# find more about CalibratedClassifierCV here a
t http://scikit-learn.org/stable/modules/genera
ted/sklearn.calibration.CalibratedClassifierCV.
html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(ba
se_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV
()
# fit(X, y[, sample_weight])    Fit the calibra
ted model
# get_params([deep])    Get parameters for this
estimator.
# predict(X)    Predict the target of new sampl
es.
# 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, met
hod="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, pr
edict_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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[be
st_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[bes
t_alpha], "The train log loss is:",log_loss(y_t
rain, 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:",lo
g_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_loss(y_tes
```

```
t, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X
_test))
```

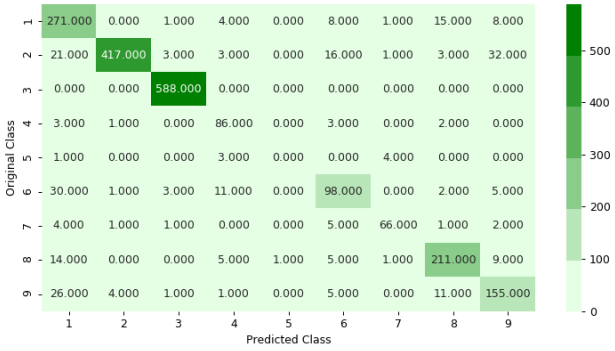
```
log_loss for k =  1 is 0.4918804536846319
6
log_loss for k =  3 is 0.483116902642161
log_loss for k =  5 is 0.5118350087441232
log_loss for k =  7 is 0.5395490778512431
log_loss for k =  9 is 0.5676371813660702
log_loss for k =  11 is 0.587017030836749
8
log_loss for k =  13 is 0.606375118318671
```



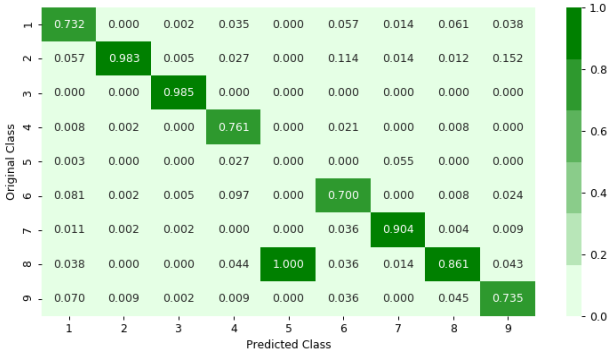

```
For values of best alpha =  3 The train l
og loss is: 0.29320594139515405
For values of best alpha =  3 The cross v
alidation log loss is: 0.483116902642161
For values of best alpha =  3 The test lo
g loss is: 0.4851954874286108
```

```
Number of misclassified points  12.971481
14075437
```

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

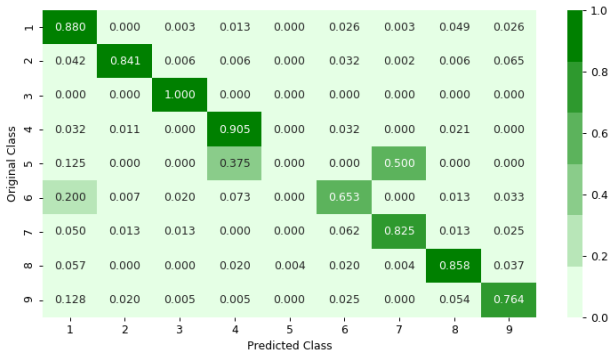


--------- Precision matrix ----------



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 [22]:

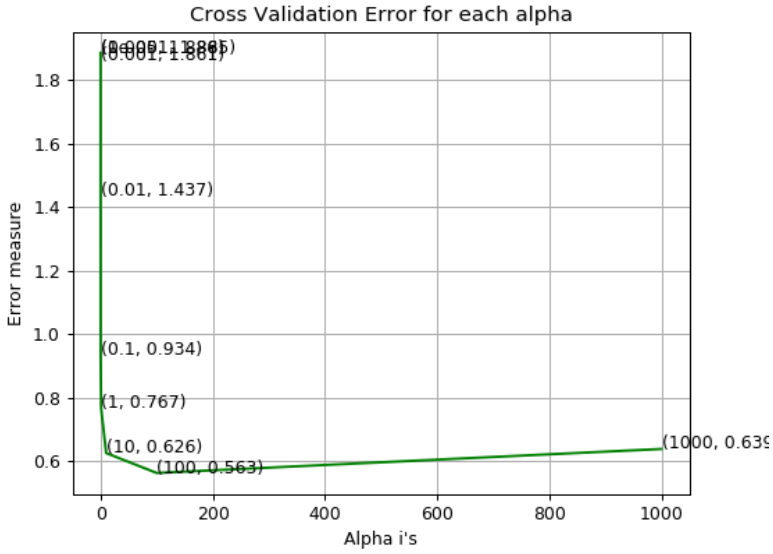
```
# read more about SGDClassifier() at http://sci
kit-learn.org/stable/modules/generated/sklearn.
linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alp
ha=0.0001, l1_ratio=0.15, fit_intercept=True, m
ax_iter=None, tol=None,
# 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]))
```

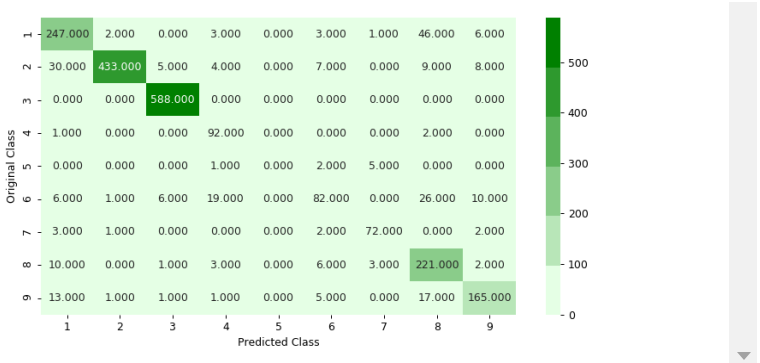
```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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 1.8861109311791922
log_loss for c =  0.0001 is 1.8845880471197165
log_loss for c =  0.001 is 1.8614285515198798

log_loss for c =  0.01 is 1.4374343790959
```

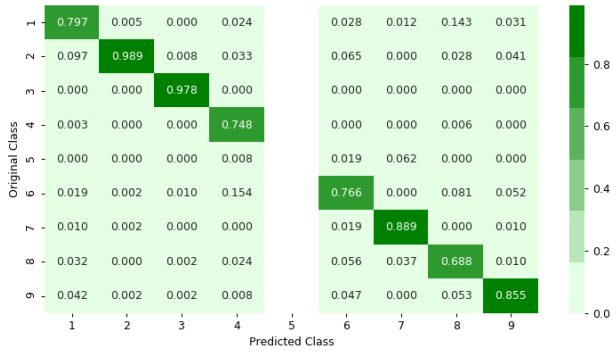
```
15
log_loss for c =  0.1 is 0.93379592046953
21
log_loss for c =  1 is 0.7667190017910965
log_loss for c =  10 is 0.625718517353697
8
log_loss for c =  100 is 0.56294262675526
log_loss for c =  1000 is 0.6385231825855
628
```




```
log loss for train data 0.487143730187876
1
log loss for cv data 0.56294262675526
log loss for test data 0.5294168099375685
Number of misclassified points  12.603495
860165593
----------------------------------------
--------- 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 [23]:

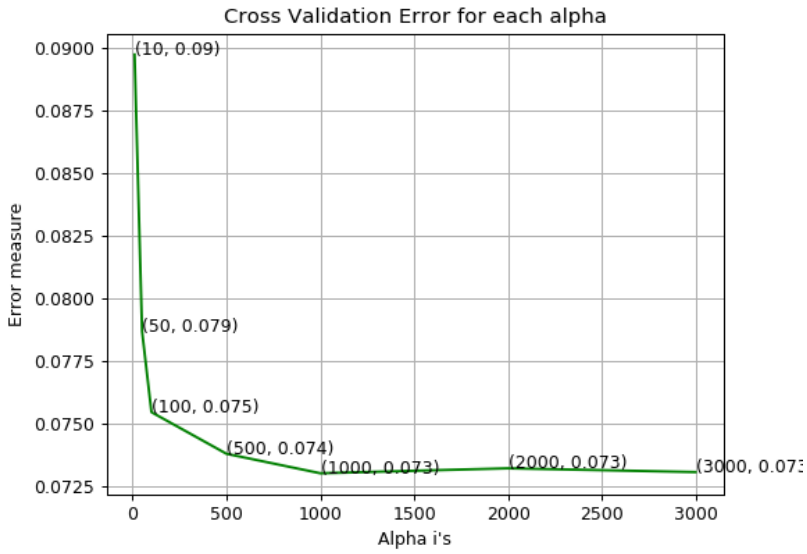
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_est
imators=10, criterion='gini', max_depth=None, m
in_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=
0.0, max_features='auto', max_leaf_nodes=None,
 min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_
score=False, n_jobs=1, random_state=None, verbo
se=0, warm_start=False,
# class_weight=None)

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

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

```
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-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-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]))
```

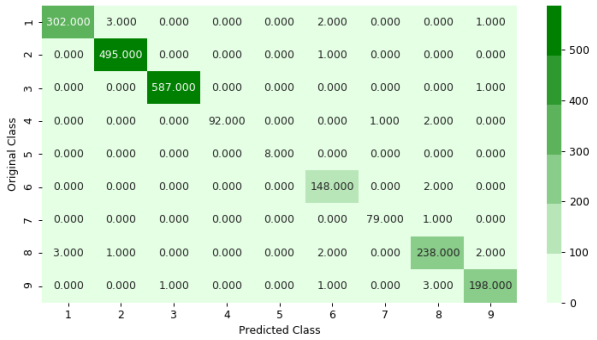
```
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best
_alpha], "The train log loss is:",log_loss(y_tr
ain, 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:",lo
g_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_loss(y_tes
t, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X
_test))
```

```
log_loss for c =  10 is 0.089752727963568
77
log_loss for c =  50 is 0.078693746810576
25
log_loss for c =  100 is 0.07546292664044
586
```

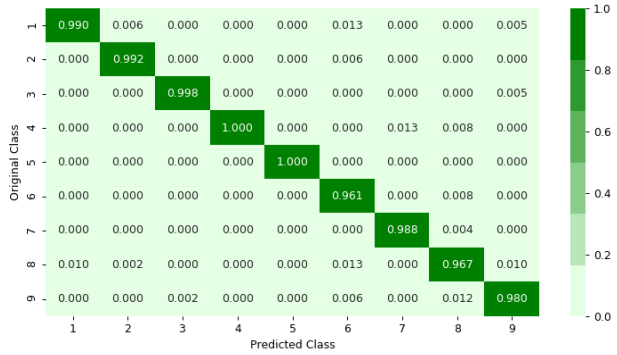
```
log_loss for c =  500 is 0.07379883728342
362
log_loss for c =  1000 is 0.0730207707851
6724
log_loss for c =  2000 is 0.0732181357402
0479
log_loss for c =  3000 is 0.0730681328644
14
```




```
For values of best alpha =  1000 The trai
n log loss is: 0.02941015229514485
For values of best alpha =  1000 The cros
s validation log loss is: 0.0730207707851
6724
For values of best alpha =  1000 The test
log loss is: 0.06623869322452512
Number of misclassified points  1.2419503
219871204
----------------------------------------
--------- 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

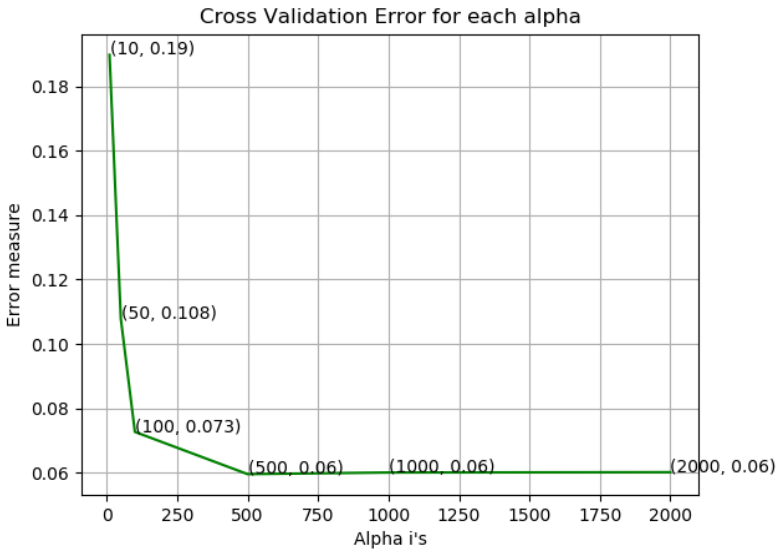
In [14]:

```
# 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-ar
```

```
e-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, met
hod="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, pr
edict_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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

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

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(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_loss(y_test, predict_y))
```

```
log_loss for c =  10 is 0.18972194520294353
log_loss for c =  50 is 0.10788109266220497
log_loss for c =  100 is 0.07274508299721164
log_loss for c =  500 is 0.059635927131908094
log_loss for c =  1000 is 0.06014538270053144
log_loss for c =  2000 is 0.06024938906225305
```

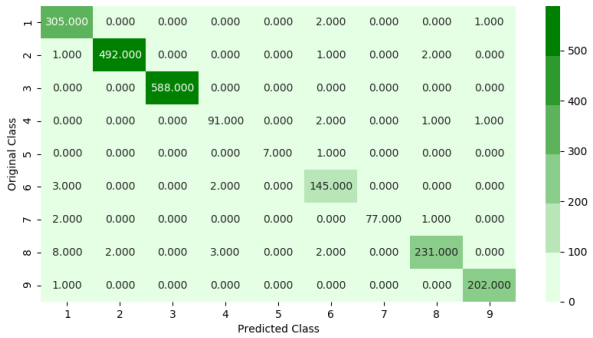



```
For values of best alpha =  500 The train
log loss is: 0.02468009568654092
For values of best alpha =  500 The cross
validation log loss is: 0.059635927131908
094
For values of best alpha =  500 The test
log loss is: 0.07847700799402009
```

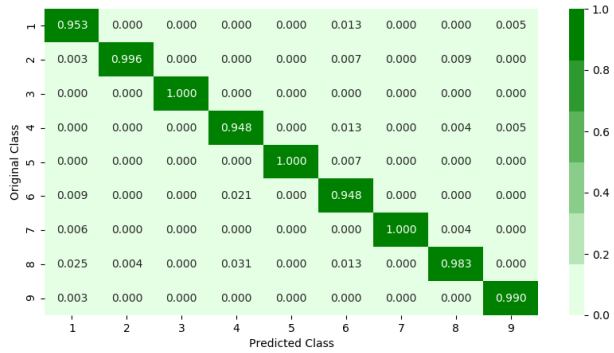
In [15]:

```
plot_confusion_matrix(y_test, sig_clf.predict(X
_test))
```

```
Number of misclassified points  1.6559337
626494939
----------------------------------------
--------- 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 [22]:

```
# 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 candidate
s, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend Loky

Backend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks
| elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  10 tasks
| elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done  17 tasks
```

```
| elapsed: 31.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 48.9min remaining:  5.4min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 54.2min finished
```

Out[22]:

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

In [ ]:

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

```
{'subsample': 1, 'n_estimators': 2000, 'm
```

```
{'subsample': 1, 'n_estimators': 2000, 'm
ax_depth': 5, 'learning_rate': 0.01, 'col
sample_bytree': 0.5}
```

In [16]:

```
x_cfl=XGBClassifier(n_estimators=2000, learning
_rate=0.01, colsample_bytree=0.5, max_depth=5,s
ubsample=1)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigm
oid')
c_cfl.fit(X_train,y_train)

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

```
train loss 0.024348854759529453
cv loss 0.06210692336009718
test loss 0.07717065282799755
```

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :

1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs

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

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

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

4.2.1 Feature extraction from asm files

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

In [ ]:
```
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='first'
folder_2 ='second'
folder_3 ='third'
```

```
folder_4 ='fourth'
folder_5 ='fifth'
folder_6 = 'output'
for i in [folder_1,folder_2,folder_3,folder_4,f
older_5,folder_6]:
    if not os.path.isdir(i):
        os.makedirs(i)

source='train/'
files = os.listdir('train')
ID=df['Id'].tolist()
data=range(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'firs
t')
    elif i%5==1:
        shutil.move(source+files[data[i]],'seco
nd')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'thri
d')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'four
th')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fift
h')
```

In [ ]:

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

def firstprocess():
```

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

    prefixes = ['HEADER:','.text:','.Pav:','.id
ata:','.data:','.bss:','.rdata:','.edata:','.rs
rc:','.tls:','.reloc:','.BSS:','.CODE']
    #this are opcodes that are used to get best
results
    #https://en.wikipedia.org/wiki/X86_instruct
ion_listings

    opcodes = ['jmp', 'mov', 'retf', 'push', 'p
op', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea'
,'movzx']
    #best keywords that are taken from differen
t 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','e
di','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),dt
ype=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 a
nd ('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:','.id
ata:','.data:','.bss:','.rdata:','.edata:','.rs
rc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'p
op', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea'
```

```
,'movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','e
di','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dt
ype=int)
        opcodescount=np.zeros(len(opcodes),dtyp
e=int)
        keywordcount=np.zeros(len(keywords),dty
pe=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 l
i in line):
                        features.append(opcodes
[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
```

```
                        if registers[i] in li a
nd ('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:','.id
ata:','.data:','.bss:','.rdata:','.edata:','.rs
rc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'p
op', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea'
,'movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','e
di','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dt
```

```
ype=int)
        opcodescount=np.zeros(len(opcodes),dtyp
e=int)
        keywordcount=np.zeros(len(keywords),dty
pe=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='c
p1252',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 l
i in line):
                        features.append(opcodes
[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li a
nd ('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:','.id
ata:','.data:','.bss:','.rdata:','.edata:','.rs
rc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'p
op', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea'
,'movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','e
di','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dt
ype=int)
        opcodescount=np.zeros(len(opcodes),dtyp
e=int)
        keywordcount=np.zeros(len(keywords),dty
pe=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 l
i in line):
                        features.append(opcodes
[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li a
nd ('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:','.id
ata:','.data:','.bss:','.rdata:','.edata:','.rs
rc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'p
op', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea'
,'movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','e
di','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
    files = os.listdir('fifth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dt
ype=int)
        opcodescount=np.zeros(len(opcodes),dtyp
e=int)
        keywordcount=np.zeros(len(keywords),dty
pe=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='c
p1252',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 l
i in line):
                        features.append(opcodes
[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li a
nd ('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 multiprogrammin
g
    #the number of process depends upon the num
```

```
ber 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 exe
cution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joine
d
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

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

In [14]:

```
# asmoutputfile.csv(output genarated from the a
bove two cells) will contain all the extracted
 features from .asm files
# this file will be uploaded in the drive, you
 can directly use this
dfasm=pd.read_csv("asmoutputfile (1).csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='le
```

```
ft')
result_asm.head()
```

Out[14]:

	ID	HEADER:	.text:	.Pav:
0	01kcPWA9K2BOxQeS5Rju	19	744	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0
3	3X2nY7iQaPBIWDrAZqJe	17	227	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [15]:

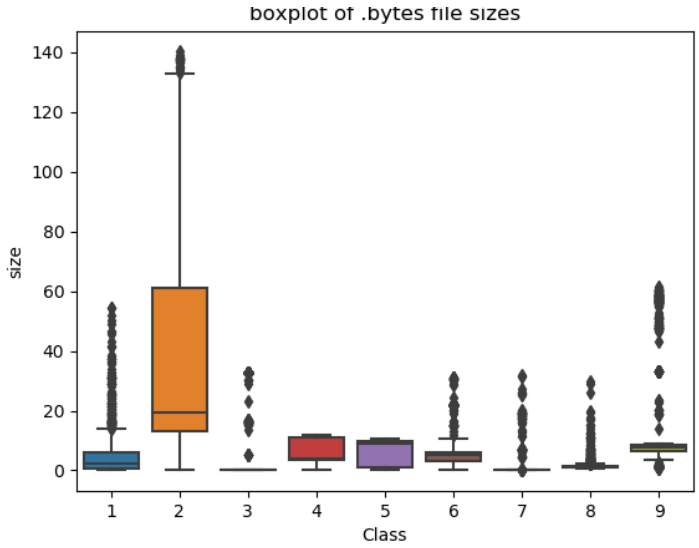
```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_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('asmFiles/'+file)
    # split the file name at '.' and take the f
irst part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in fil
enames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(102
4.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':
sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

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

4.2.1.2 Distribution of .asm file sizes

In [15]:
```
#boxplot of asm files
ax = sns.boxplot(x="Class", y="size", data=asm_
size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



In [16]:

```
# 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='left')
result_asm.head()
```

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

Out[16]:

	ID	HEADER:	.text:	.Pav:
0	01kcPWA9K2BOxQeS5Rju	19	744	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0
3	3X2nY7iQaPBIWDrAZqJe	17	227	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0

5 rows × 54 columns

In [17]:

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

Out[17]:

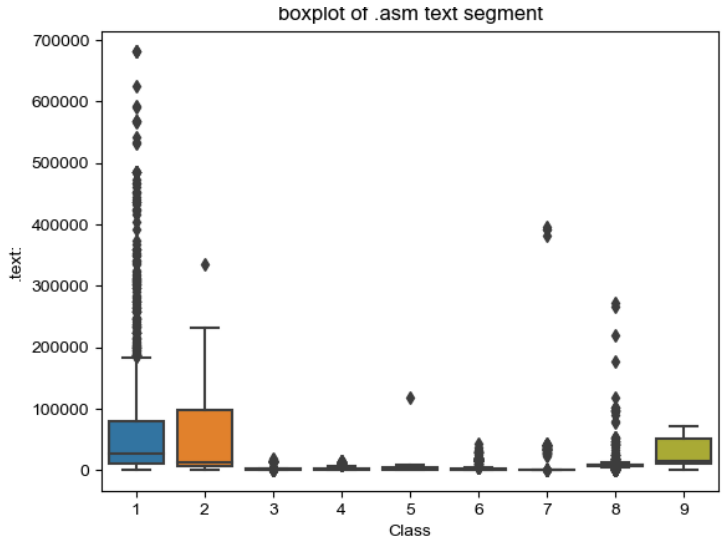
	ID	HEADER:	.text:	.Pav:
0	01kcPWA9K2BOxQeS5Rju	19	744	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0
3	3X2nY7iQaPBIWDrAZqJe	17	227	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

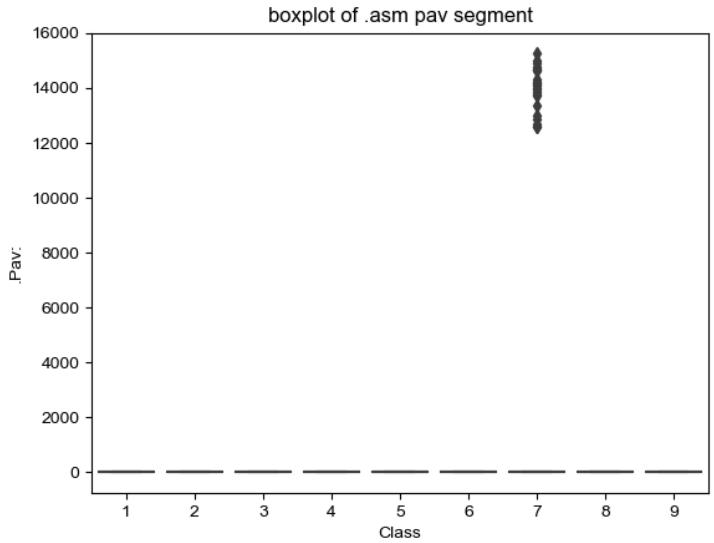
In [18]:

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

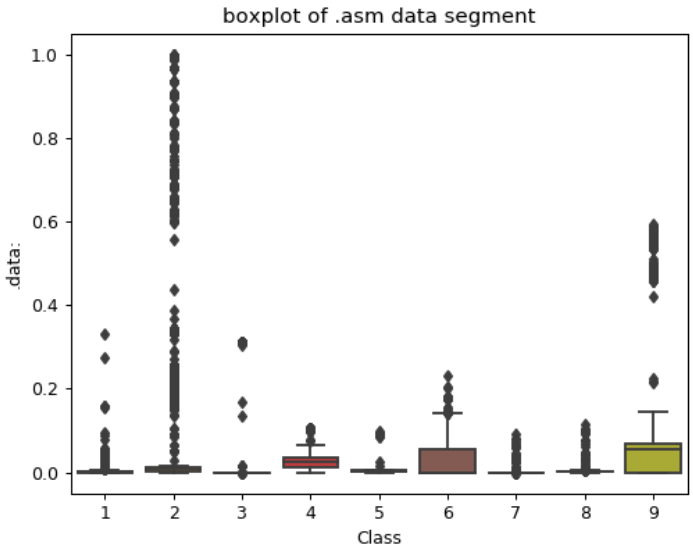


The plot is between Text and class
Class 1,2 and 9 can be easly separated

In [19]:
```
ax = sns.boxplot(x="Class", y=".Pav:", data=res
ult_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```

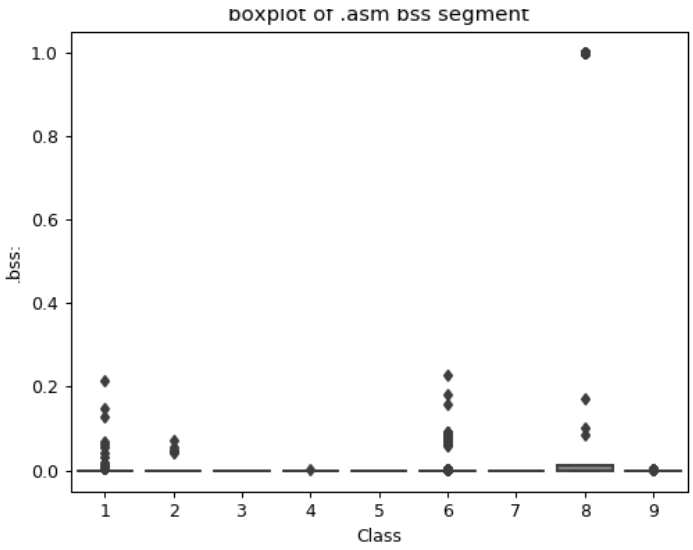


In [19]:
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



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

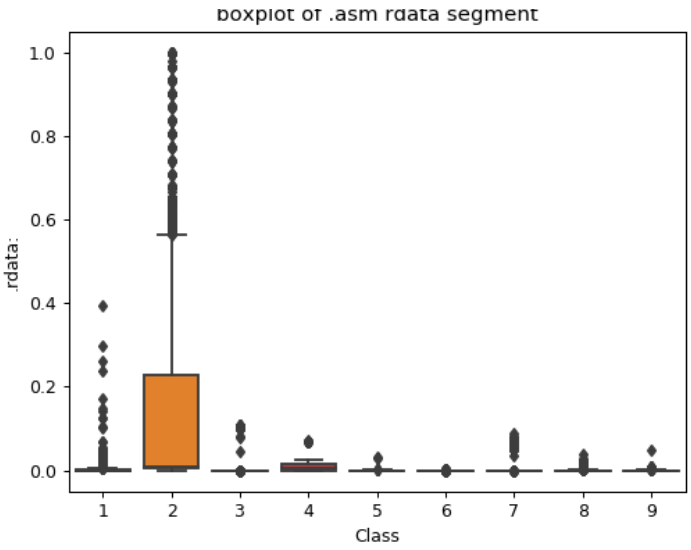
In [20]:
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

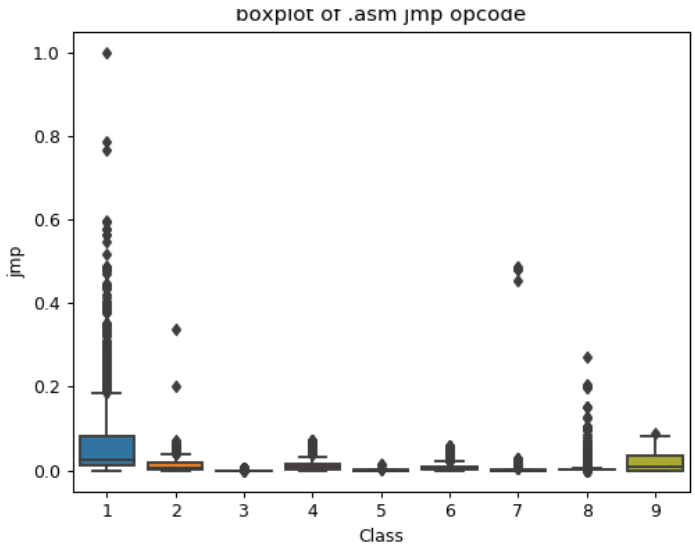
In [21]:

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



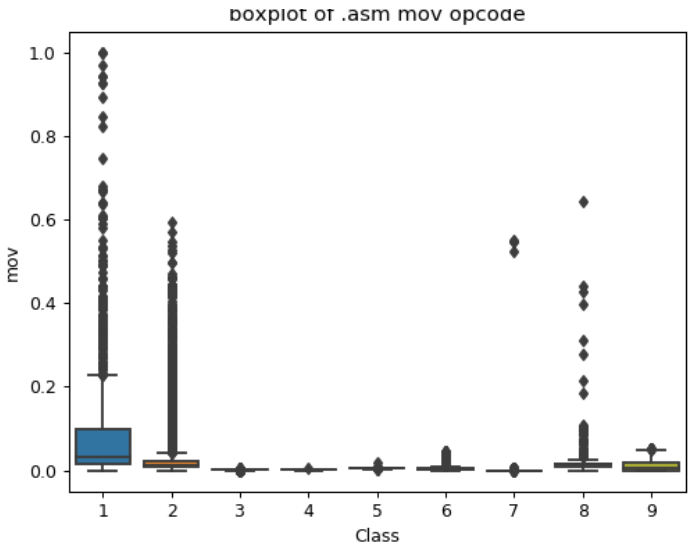
Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [22]:
```
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 appr ox in 75 perentile of files

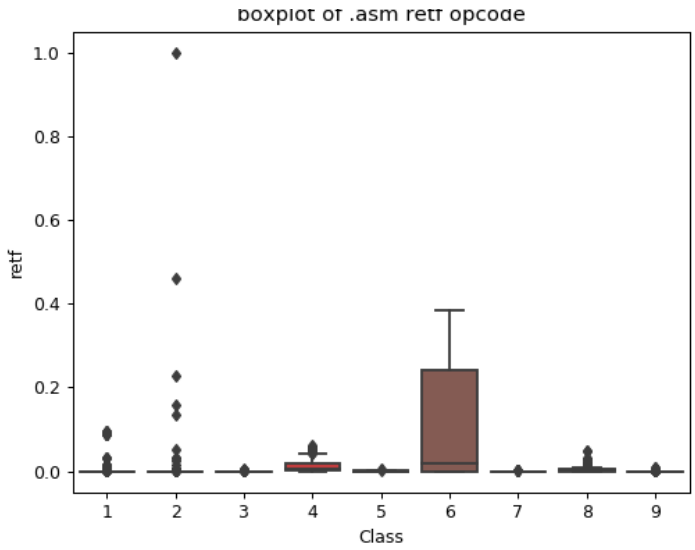
In [23]:
```
ax = sns.boxplot(x="Class", y="mov", data=resul
t_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



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

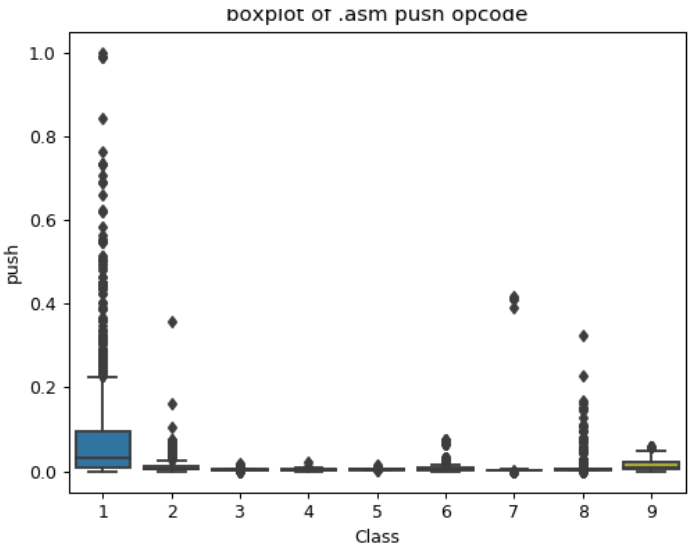
In [24]:

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



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

In [25]:
```
ax = sns.boxplot(x="Class", y="push", data=resu
lt_asm)
plt.title("boxplot of .asm push opcode")
plt.show()
```



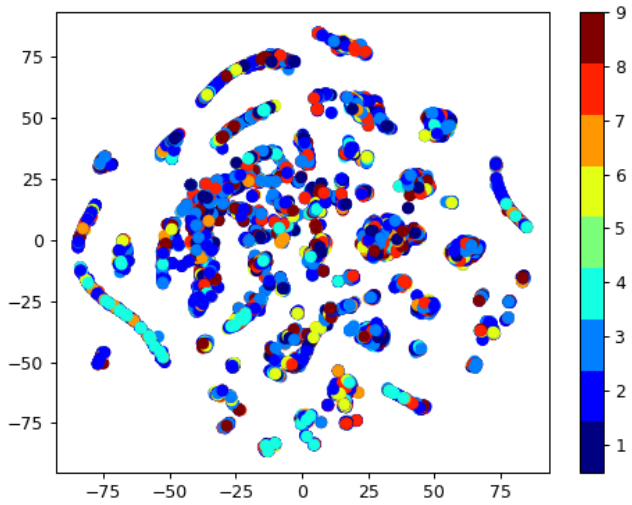
plot between push opcode and Class label
Class 1 is having 75 precentile files wi
th push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

In [16]:

```
# check out the course content for more explant
ion on tsne algorithm
# https://www.appliedaicourse.com/course/applie
d-ai-course-online/lessons/t-distributed-stocha
stic-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(['I
D','Class'], axis=1).fillna(0))
vis_x = results[:, 0]
vis_y = results[:, 1   ]
```

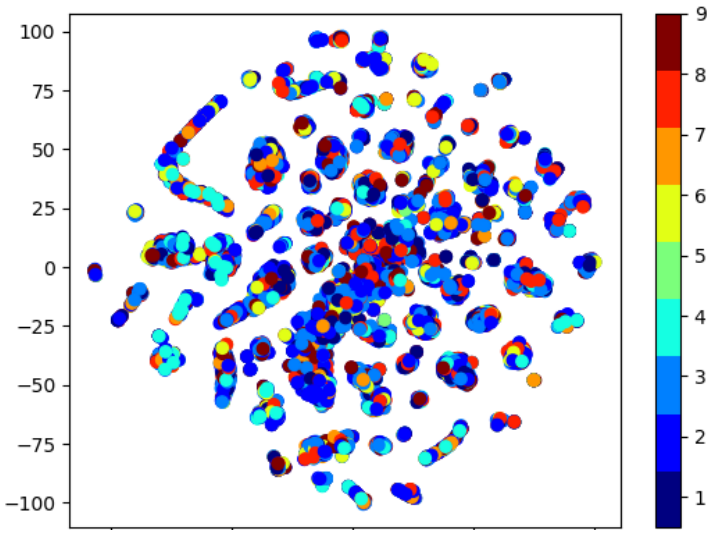
```
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm
.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



In [30]:

```
# by univariate analysis on the .asm file featu
res we are getting very negligible information
 from
# 'rtn', '.BSS:' '.CODE' features, so heare we
 are trying multivariate analysis after removin
g those features
# the plot looks very messy

xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['I
D','Class', 'rtn', '.BSS:', '.CODE','size'], ax
is=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 [18]:
```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:',
'rtn','.CODE'], axis=1)
```

In [19]:
```
X_train_asm, X_test_asm, y_train_asm, y_test_as
m = train_test_split(asm_x,asm_y ,stratify=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_asm,strat
ify=y_train_asm,test_size=0.20)
```

In [20]:
```
print( X_cv_asm.isnull().all())
```

```
HEADER:     False
.text:      False
.Pav:       False
.idata:     False
.data:      False
.bss:       False
.rdata:     False
.edata:     False
.rsrc:      False
.tls:       False
.reloc:     False
jmp         False
mov         False
retf        False
push        False
pop         False
xor         False
retn        False
```

```
nop         False
sub         False
inc         False
dec         False
add         False
imul        False
xchg        False
or          False
shr         False
cmp         False
call        False
shl         False
ror         False
rol         False
jnb         False
jz          False
lea         False
movzx       False
.dll        False
std::       False
:dword      False
edx         False
esi         False
eax         False
ebx         False
ecx         False
edi         False
ebp         False
esp         False
eip         False
size        False
dtype: bool
```

4.4. Machine Learning models on

features of .asm files

4.4.1 K-Nearest Neigbors

In [35]:

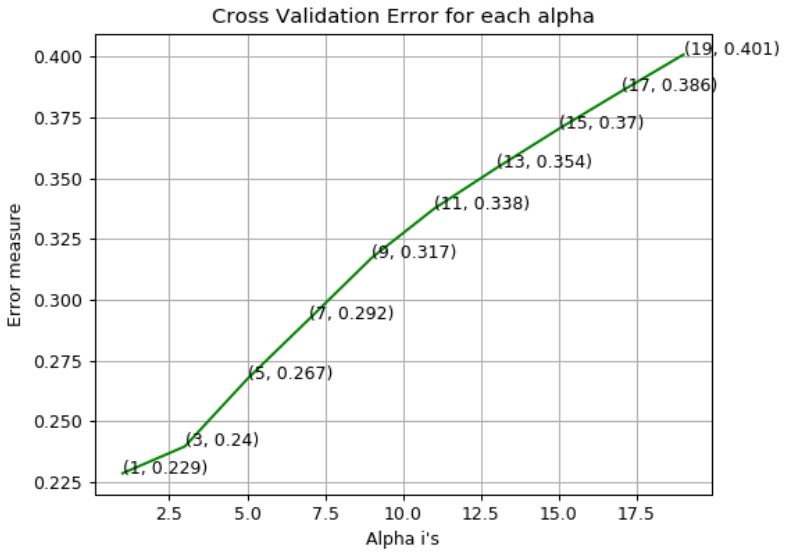
```
# find more about KNeighborsClassifier() here h
ttp://scikit-learn.org/stable/modules/generate
d/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights
='uniform', algorithm='auto', leaf_size=30, p=
2,
# metric='minkowski', metric_params=None, n_job
s=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 p
rovided data
# predict_proba(X):Return probability estimates
for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/c
ourse/applied-ai-course-online/lessons/k-neares
t-neighbors-geometric-intuition-with-a-toy-exam
ple-1/
#-----------------------------------


# find more about CalibratedClassifier
#CV here at http://scikit-learn.org/stable/modu
les/generated/sklearn.calibration.CalibratedCla
ssifierCV.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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[be
st_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_tra
in_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.predic
t(X_test_asm))
```

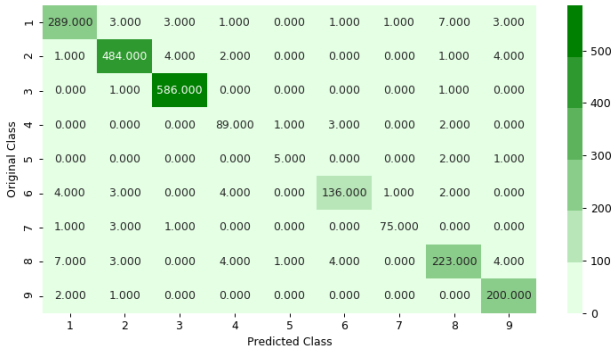
```
log_loss for k =  1 is 0.2286951008264786
```

```
log_loss for k =  3 is 0.2397460490976792
1
log_loss for k =  5 is 0.2674376718256929
5
log_loss for k =  7 is 0.2922812711849662
log_loss for k =  9 is 0.3173517943176062
log_loss for k =  11 is 0.337527230134397
3
log_loss for k =  13 is 0.354258171718433
4
log_loss for k =  15 is 0.370356725235185
4
log_loss for k =  17 is 0.385757047159040
1
log_loss for k =  19 is 0.400903349169395
35
```

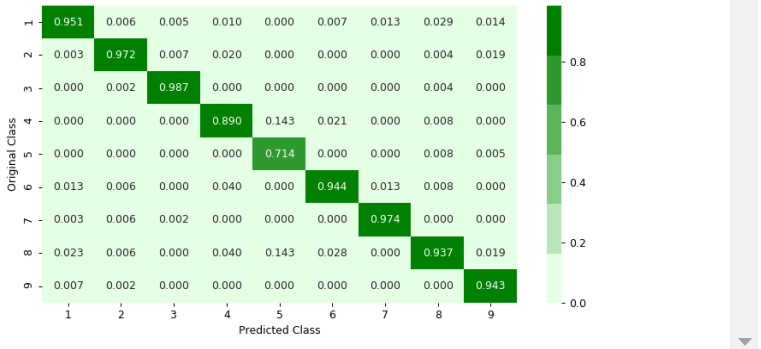


log loss for train data 0.073718205506760
85
log loss for cv data 0.2286951008264786
log loss for test data 0.2135354369723588
Number of misclassified points 4.0018399
26402944

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

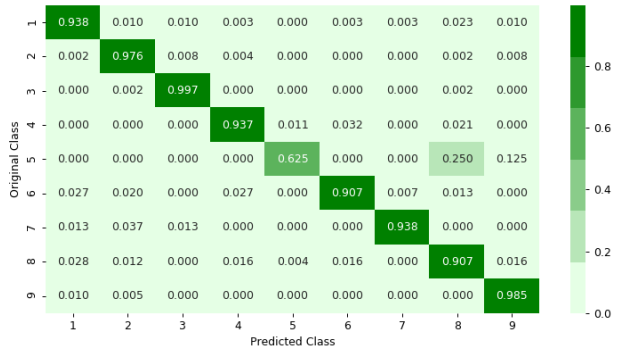


--------- Precision matrix --------------



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 [36]:

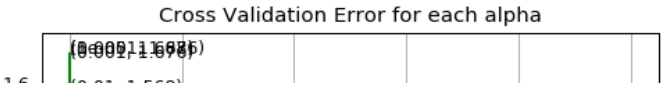
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
```

```
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alp
ha=0.0001, l1_ratio=0.15, fit_intercept=True, m
ax_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 Desce
nt.
# predict(X)    Predict class labels for sample
s in X.

#------------------------------
# video link: https://www.appliedaicourse.com/c
ourse/applied-ai-course-online/lessons/geometri
c-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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alp
ha[best_alpha],class_weight='balanced')
logisticR.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(logisticR, met
hod="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_tr
ain_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_as
m, predict_y, labels=logisticR.classes_, eps=1e
-15)))
```

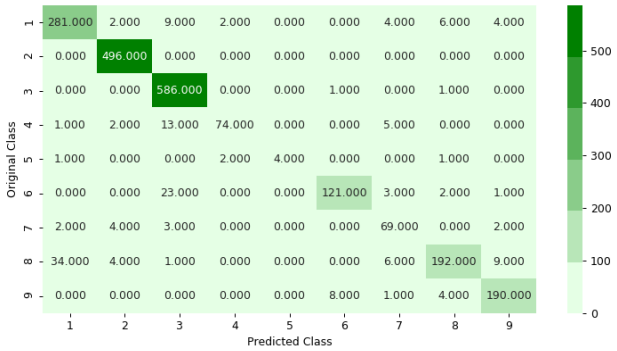
```
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_tes
t_asm, predict_y, labels=logisticR.classes_, ep
s=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predic
t(X_test_asm))
```

```
log_loss for c =  1e-05 is 1.686985986895
7804
log_loss for c =  0.0001 is 1.68550101924
72757
log_loss for c =  0.001 is 1.675578156215
2487
log_loss for c =  0.01 is 1.5677273322121
714
log_loss for c =  0.1 is 1.30025731163389
27
log_loss for c =  1 is 0.856048258533692
log_loss for c =  10 is 0.573568764987986
4
log_loss for c =  100 is 0.44312147180989
47
log_loss for c =  1000 is 0.4315735323228
3385
```



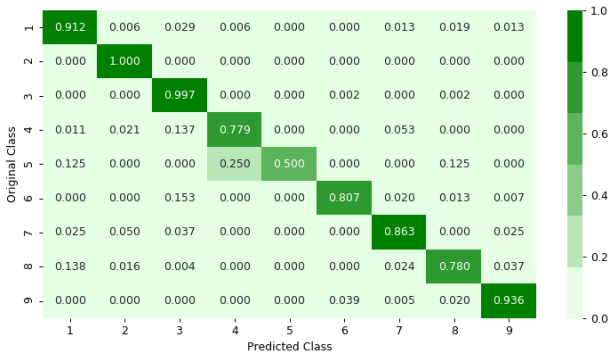
log loss for train data 0.381505481961809
33
log loss for cv data 0.43157353232283385
log loss for test data 0.3830892601338432
6
Number of misclassified points 7.4057037
71849126

--------- 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 [37]:

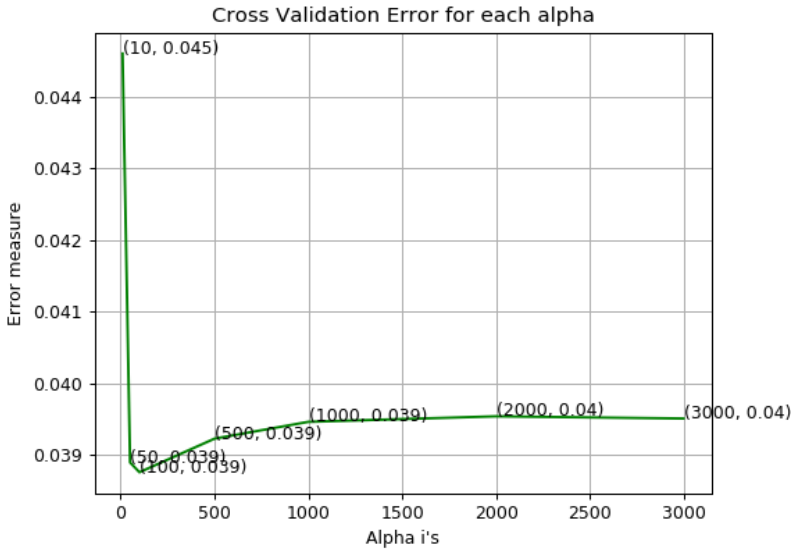
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_est
imators=10, criterion='gini', max_depth=None, m
in_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=
0.0, max_features='auto', max_leaf_nodes=None,
 min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_
score=False, n_jobs=1, random_state=None, verbo
se=0, warm_start=False,
# class_weight=None)
```

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

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

# ---------------------------------
# video link: https://www.appliedaicourse.com/c
ourse/applied-ai-course-online/lessons/random-f
orest-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, met
hod="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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
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_tr
ain_asm, predict_y, labels=sig_clf.classes_, ep
s=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_as
m, predict_y, labels=sig_clf.classes_, eps=1e-1
5)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_tes
t_asm, predict_y, labels=sig_clf.classes_, eps=
1e-15)))
```

```
plot_confusion_matrix(y_test_asm,sig_clf.predic
t(X_test_asm))
```

```
log_loss for c =  10 is 0.044604000720488
056
log_loss for c =  50 is 0.038892329851189
296
log_loss for c =  100 is 0.03875524544813
011
log_loss for c =  500 is 0.03922444080580
9314
log_loss for c =  1000 is 0.0394594179078
3839
log_loss for c =  2000 is 0.0395365912328
6974
log_loss for c =  3000 is 0.0395060858773
2239
```

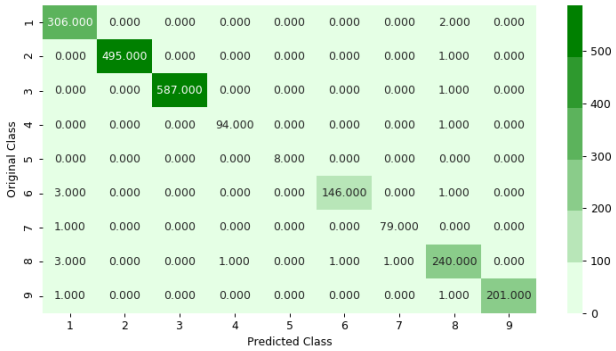



```
log loss for train data 0.012379247850927
044
log loss for cv data 0.03875524544813011
log loss for test data 0.0394283789368753
```

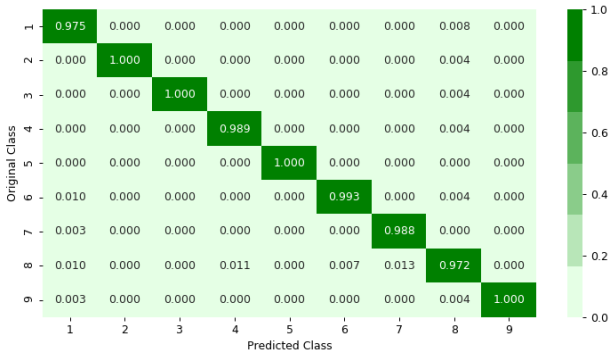
76

Number of misclassified points 0.8279668813247469

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

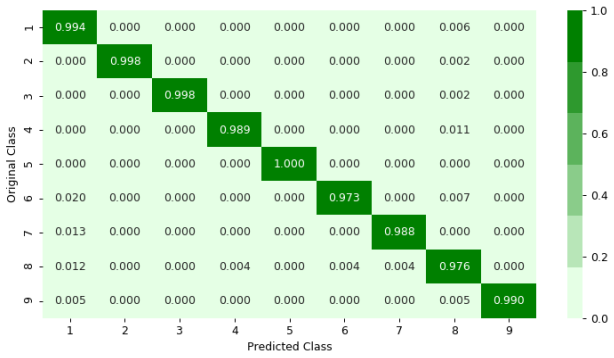


--------- Precision matrix ----------



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 [38]:

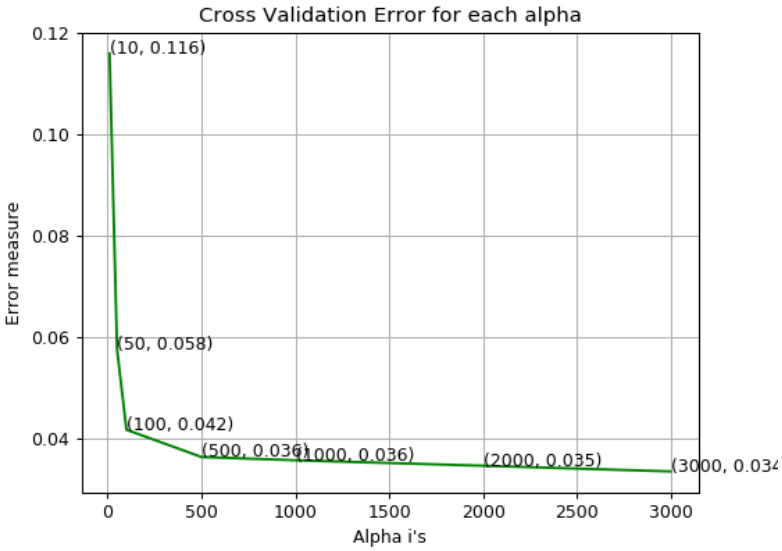
```
# Training a hyper-parameter tuned Xg-Boost reg
ressor on our train data

# find more about XGBClassifier function here h
ttp://xgboost.readthedocs.io/en/latest/python/p
ython_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, lear
ning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtre
e', n_jobs=1, nthread=None, gamma=0, min_child_
weight=1,
# max_delta_step=0, subsample=1, colsample_bytr
ee=1, colsample_bylevel=1, reg_alpha=0, reg_lam
bda=1,
# scale_pos_weight=1, base_score=0.5, random_st
ate=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_limi
t=0) : Predict with data. NOTE: This function i
s not thread safe.
# get_score(importance_type='weight') -> get th
e feature importance
# ----------------------
# video link2: https://www.appliedaicourse.com/
course/applied-ai-course-online/lessons/what-ar
e-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, met
hod="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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alp
ha],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[bes
t_alpha], "The train log loss is:",log_loss(y_t
rain_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:",lo
g_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_loss(y_tes
t_asm, predict_y))
```

```
plot_confusion_matrix(y_test_asm,sig_clf.predic
t(X_test_asm))
```

```
log_loss for c =  10 is 0.115881053383402
65
log_loss for c =  50 is 0.057658250882591
494
log_loss for c =  100 is 0.04186141305711
363
log_loss for c =  500 is 0.03649854125696
994
log_loss for c =  1000 is 0.0358596195193
93905
log_loss for c =  2000 is 0.0347823675220
7586
log_loss for c =  3000 is 0.0336673034374
09195
```




```
For values of best alpha =  3000 The trai
n log loss is: 0.009827260187420 22
For values of best alpha =  3000 The cros
s validation log loss is: 0.0336673034374
```

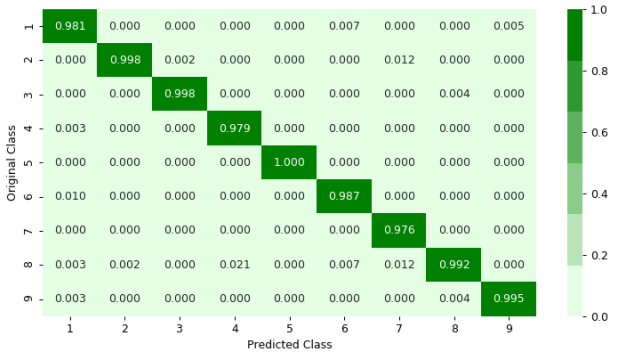
09195

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

Number of misclassified points 0.7819687 2125115

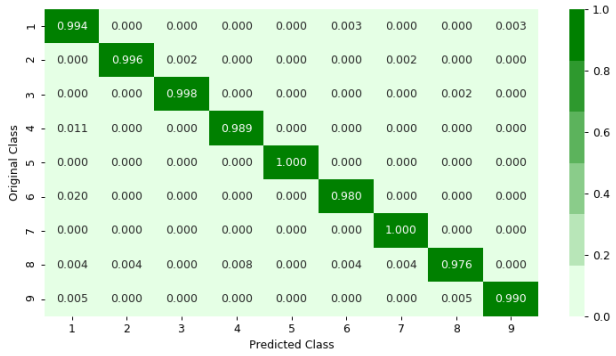
--------- Confusion matrix ----------

--------- 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 [39]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.
2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distr
ibutions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_asm,y_train_asm)
```

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

```
[Parallel(n_jobs=-1)]: Using backend Loky
Backend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks
| elapsed:  5.6min
[Parallel(n_jobs=-1)]: Done  10 tasks
| elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done  17 tasks
| elapsed:  9.3min
[Parallel(n_jobs=-1)]: Done  27 out of  3
0 | elapsed: 15.6min remaining:  1.7min
[Parallel(n_jobs=-1)]: Done  30 out of  3
0 | elapsed: 16.7min finished
```

Out[39]:

```
RandomizedSearchCV(cv='warn', error_score
='raise-deprecating',
          estimator=XGBClassifier(base_sc
ore=0.5, booster='gbtree', colsample_byle
vel=1,
       colsample_bytree=1, gamma=0, learn
ing_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, m
issing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective
='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_p
os_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_
iter=10, n_jobs=-1,
          param_distributions={'learning_

rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.
2], 'n_estimators': [100, 200, 500, 1000,
2000], 'max_depth': [3, 5, 10], 'colsampl
e_bytree': [0.1, 0.3, 0.5, 1], 'subsampl
e': [0.1, 0.3, 0.5, 1]},
```

```
          pre_dispatch='2*n_jobs', random
_state=None, refit=True,
          return_train_score='warn', scor
ing=None, verbose=10)
```

In [40]:
```
print (random_cfl.best_params_)
```

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

In [42]:
```
# Training a hyper-parameter tuned Xg-Boost reg
ressor on our train data

# find more about XGBClassifier function here h
ttp://xgboost.readthedocs.io/en/latest/python/p
ython_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, lear
ning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtre
e', n_jobs=1, nthread=None, gamma=0, min_child_
weight=1,
# max_delta_step=0, subsample=1, colsample_bytr
ee=1, colsample_bylevel=1, reg_alpha=0, reg_lam
bda=1,
# scale_pos_weight=1, base_score=0.5, random_st
ate=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_limi
t=0) : Predict with data. NOTE: This function i
s not thread safe.
# get_score(importance_type='weight') -> get th
e feature importance
# ----------------------
# video link2: https://www.appliedaicourse.com/
course/applied-ai-course-online/lessons/what-ar
e-ensembles/
# ----------------------

x_cfl=XGBClassifier(n_estimators=2000,subsample
=0.5,learning_rate=0.01,colsample_bytree=0.5,ma
x_depth=10)
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigm
oid')
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, predi
ct_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict
_y))
```

```
train loss 0.010626478719576738
cv loss 0.033016089856804966
test loss 0.04082419520278345
```

4.5. Machine Learning models on

features of both .asm and .bytes files

4.5.1. Merging both asm and byte file features

In [21]:
```
result.head()
```

Out[21]:

	Unnamed: 0	ID	0	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0

5 rows × 261 columns

In [22]:
```
result_asm.head()
```

Out[22]:

	ID	HEADER:	.text:	.Pav:
0	01kcPWA9K2BOxQeS5Rju	19	744	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0
3	3X2nY7iQaPBIWDrAZqJe	17	227	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0

5 rows × 54 columns

In [23]:
```
print(result.shape)
print(result_asm.shape)
```

```
(10868, 261)
(10868, 54)
```

In [24]:
```
result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[24]:

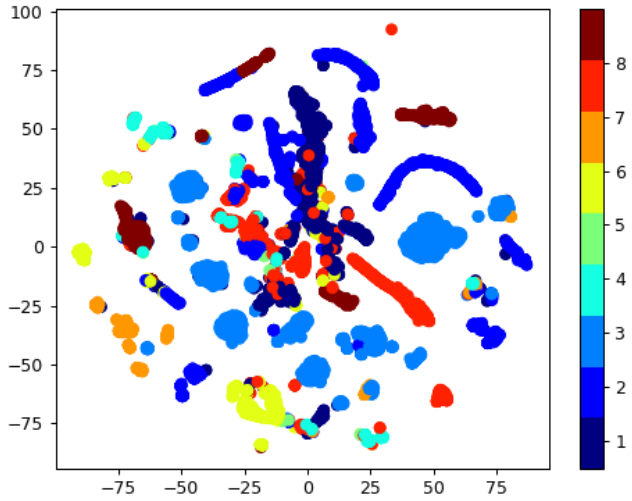
	Unnamed: 0	0	1	2	
0	0.000000	0.262806	0.005498	0.001567	0.00206
1	0.000092	0.017358	0.011737	0.004033	0.00387
2	0.000184	0.040827	0.013434	0.001429	0.00131
3	0.000276	0.009209	0.001708	0.000404	0.00044
4	0.000368	0.008629	0.001000	0.000168	0.00023

5 rows × 308 columns

4.5.2. Multivariate Analysis on final fearures

In [25]:
```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
```

```
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```



4.5.3. Train and Test split

In [26]:
```
X_train, X_test_merge, y_train, y_test_merge =
train_test_split(result_x, result_y,stratify=re
sult_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_
merge = train_test_split(X_train, y_train,strat
ify=y_train,test_size=0.20)
```

4.5.4. Random Forest Classifier on final features

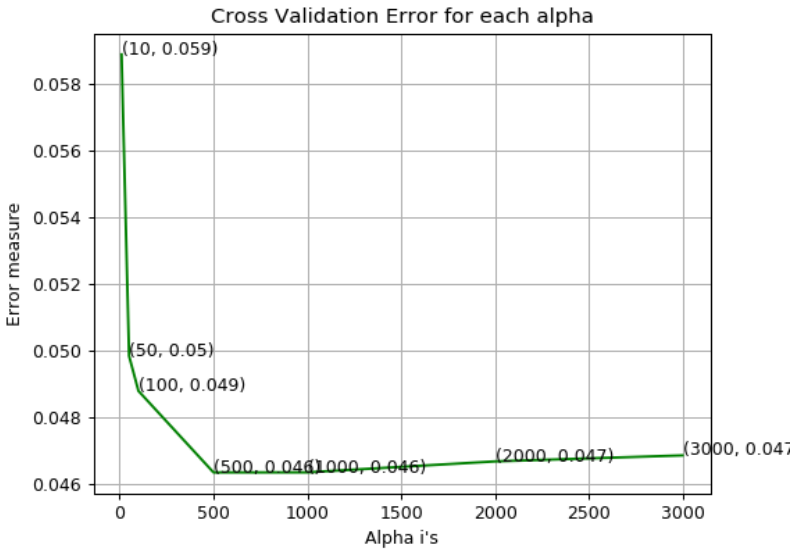
In [34]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_est
```

```
imators=10, criterion='gini', max_depth=None, m
in_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=None, verbo
se=0, warm_start=False,
# class_weight=None)

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/c
ourse/applied-ai-course-online/lessons/random-f
orest-and-their-construction-2/
# ------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassi
fier
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, met
hod="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merg
e)
    cv_log_error_array.append(log_loss(y_cv_mer
ge, 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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
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[bes
t_alpha], "The train log loss is:",log_loss(y_t
rain_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:",lo
g_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_tes
t_merge, predict_y))
```

```
log_loss for c =  10 is 0.058864988008900
165
log_loss for c =  50 is 0.049821713523895
83
log_loss for c =  100 is 0.04877439563993
806
log_loss for c =  500 is 0.04633136949419
593
log_loss for c =  1000 is 0.0463328266984
2955
log_loss for c =  2000 is 0.0466614893130
4081
log_loss for c =  3000 is 0.0468416173343
0787
```




```
For values of best alpha =  500 The train
log loss is: 0.015045746557915482
For values of best alpha =  500 The cross
validation log loss is: 0.046331369494195
93
For values of best alpha =  500 The test
log loss is: 0.0419437056294099
```

4.5.5. XgBoost Classifier on final features

In [35]:

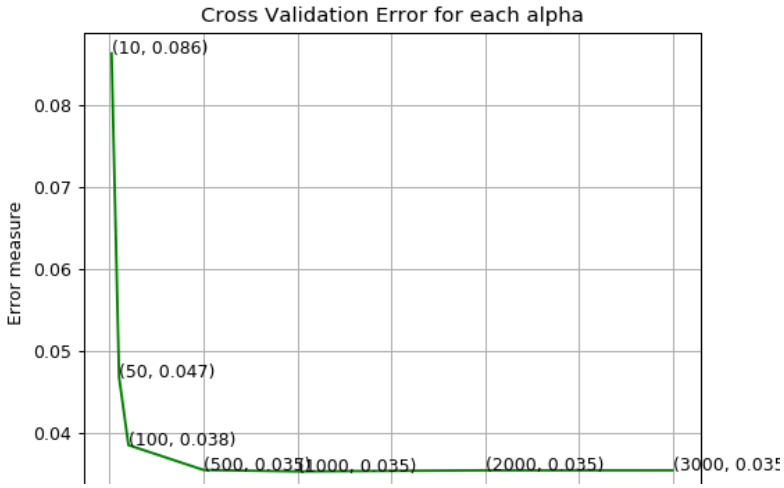
```
# Training a hyper-parameter tuned Xg-Boost reg
ressor on our train data

# find more about XGBClassifier function here h
ttp://xgboost.readthedocs.io/en/latest/python/p
ython_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, lear
ning_rate=0.1, n_estimators=100, silent=True,
```

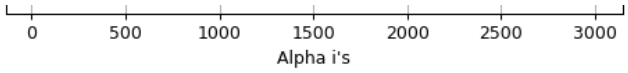
```
# objective='binary:logistic', booster='gbtre
e', n_jobs=1, nthread=None, gamma=0, min_child_
weight=1,
# max_delta_step=0, subsample=1, colsample_bytr
ee=1, colsample_bylevel=1, reg_alpha=0, reg_lam
bda=1,
# scale_pos_weight=1, base_score=0.5, random_st
ate=0, seed=None, missing=None, **kwargs)

# 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_limi
t=0) : Predict with data. NOTE: This function i
s not thread safe.
# get_score(importance_type='weight') -> get th
e feature importance
# ----------------------
# video link2: https://www.appliedaicourse.com/
course/applied-ai-course-online/lessons/what-ar
e-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, met
hod="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merg
e)
```

```
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
rain_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:",lo
g_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_tes
t_merge, predict_y))
```

```
log_loss for c =  10 is 0.086344102591976
68
log_loss for c =  50 is 0.046796220027048
7
log_loss for c =  100 is 0.03846464669244
138
log_loss for c =  500 is 0.03542509345482
663
log_loss for c =  1000 is 0.0352479011374
5623
log_loss for c =  2000 is 0.0353782044873
6872
log_loss for c =  3000 is 0.0353841592455
50155
```






```
For values of best alpha =  1000 The trai
n log loss is: 0.010771162453744454
For values of best alpha =  1000 The cros
s validation log loss is: 0.0353841592455
50155
For values of best alpha =  1000 The test
log loss is: 0.024834218493213808
```

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

In [36]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.
2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distr
ibutions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_merge, y_train_merge)
```

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

```
[Parallel(n_jobs=-1)]: Using backend Loky
Backend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks
```

```
| elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done  10 tasks
| elapsed: 11.0min
[Parallel(n_jobs=-1)]: Done  17 tasks
| elapsed: 17.5min
[Parallel(n_jobs=-1)]: Done  27 out of  3
0 | elapsed: 28.6min remaining:  3.2min
[Parallel(n_jobs=-1)]: Done  30 out of  3
0 | elapsed: 37.0min finished
```

Out[36]:

```
RandomizedSearchCV(cv='warn', error_score
='raise-deprecating',
          estimator=XGBClassifier(base_sc
ore=0.5, booster='gbtree', colsample_byle
vel=1,
       colsample_bytree=1, gamma=0, learn
ing_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, m
issing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective
='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_p
os_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_
iter=10, n_jobs=-1,
          param_distributions={'learning_
rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.
2], 'n_estimators': [100, 200, 500, 1000,
2000], 'max_depth': [3, 5, 10], 'colsampl
e_bytree': [0.1, 0.3, 0.5, 1], 'subsampl
e': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random
_state=None, refit=True,
          return_train_score='warn', scor
ing=None, verbose=10)
```

In [37]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 200, 'ma
x_depth': 5, 'learning_rate': 0.1, 'colsa
mple_bytree': 0.5}
```

In [39]:
```
# find more about XGBClassifier function here h
ttp://xgboost.readthedocs.io/en/latest/python/p
ython_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, lear
ning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtre
e', n_jobs=1, nthread=None, gamma=0, min_child_
weight=1,
# max_delta_step=0, subsample=1, colsample_bytr
ee=1, colsample_bylevel=1, reg_alpha=0, reg_lam
bda=1,
# scale_pos_weight=1, base_score=0.5, random_st
ate=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_limi
t=0) : Predict with data. NOTE: This function i
s not thread safe.
# get_score(importance_type='weight') -> get th
e feature importance
# ----------------------
```

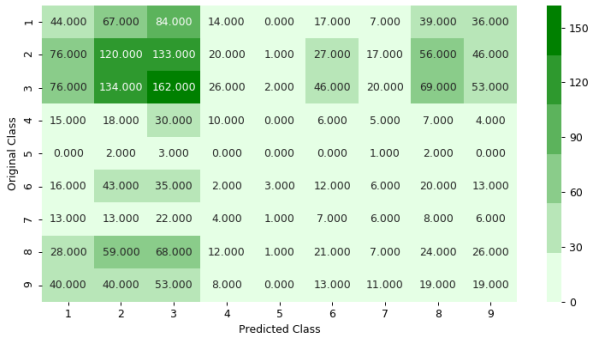
```
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# ----------------------

x_cfl=XGBClassifier(n_estimators=1000,max_depth=5,learning_rate=0.1,colsample_bytree=0.5,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 =  1000 The train log loss is: 0.010849938040281054
For values of best alpha =  1000 The cross validation log loss is: 0.032691088389142 83
For values of best alpha =  1000 The test
```

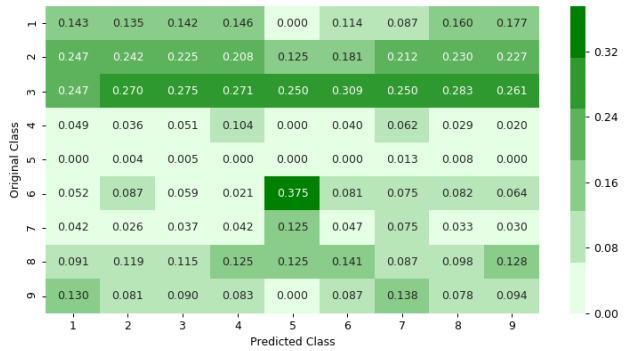
log loss is: 0.02814277993749233

Number of misclassified points 81.73873045078197

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

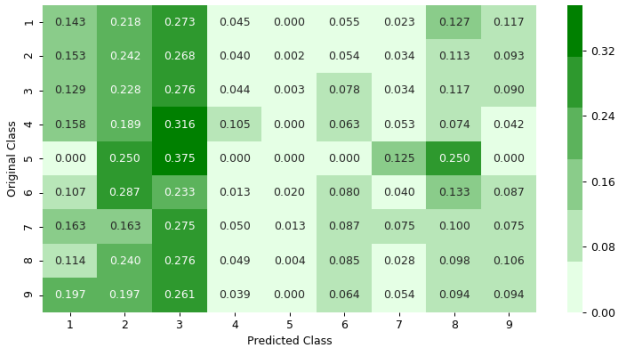


--------- Precision matrix ----------



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.]
```

5. Assignments

1. Add bi-grams on byte files and asm files and improve the log-loss to 0.01
2. Watch the video (video) and include pixel intensity features to improve the logloss

Creating bi-grams and n-gram features on byte files and asm files

On Byte files

In [ ]:

```
'''#http://www.albertauyeung.com/post/generating-ngrams-python/
def generate_ngrams(s, n):
    # Convert to lowercases
    s = s.lower()

    # Break sentence in the token, remove empty tokens
    tokens = [token for token in s.split(",") if token != ""]

    # Use the zip function to help us generate n-grams
    # Concatentate the tokens into ngrams and return
    ngrams = zip(*[tokens[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]'''
```

In [23]:

```
byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,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,f
3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [26]:
```
byte_bigram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for j in range(0, len(byte_vocab.split(','
))):
        byte_bigram_vocab.append(v + ' ' +byte_
vocab.split(',')[j])
len(byte_bigram_vocab)
```

Out[26]:
```
66049
```

In [27]:
```
byte_bigram_vocab[:5]
```

Out[27]:
```
['00 00', '00 01', '00 02', '00 03', '00
04']
```

In [6]:
```
byte_trigram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for j in range(0, len(byte_vocab.split(','
))):
        for k in range(0, len(byte_vocab.split(
',',))):
            byte_trigram_vocab.append(v + ' ' +
byte_vocab.split(',')[j]+' '+byte_vocab.split(
',')[k])
len(byte_trigram_vocab)
```

Out[6]:
```
16974593
```

In [7]:
```
byte_trigram_vocab[:5]
```

Out[7]:
```
['00 00 00', '00 00 01', '00 00 02', '00
00 03', '00 00 04']
```

In [28]:
```
import scipy
from sklearn.feature_extraction.text import Cou
ntVectorizer
vector = CountVectorizer(lowercase=False,ngram_
range=(2,2), vocabulary=byte_bigram_vocab)
bytebigram_vect = scipy.sparse.csr_matrix((1086
8, 66049))
for i, file in enumerate(os.listdir('byteFiles'
)):
    f = open('byteFiles/' + file)
    bytebigram_vect[i,:]+= scipy.sparse.csr_mat
rix(vect.fit_transform([f.read().replace('\n',
' ').lower()]))
    f.close()
```

In [39]:
```
from sklearn.preprocessing import normalize
byte_bigram_vect = normalize(byte_bigram_vect)
bytebigram_vect
```

Out[39]:
```
<10868x66049 sparse matrix of type '<clas
s 'numpy.float64'>'
        with 0 stored elements in Compres
sed Sparse Row format>
```

On asm files {N-Gram(2-gram, 3-gram, 4-gram) Opcodes}

In [31]:
```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop',
'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'ad
d','imul', 'xchg', 'or', 'shr', 'cmp', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','mo
vzx']
```

In [32]:
```
asmopcodebigram = []
```

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

Out[32]:
```
676
```

In [33]:
```
asmopcodetrigram = []
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        for k in range(0, len(opcodes)):
            asmopcodetrigram.append(v + ' ' + o
pcodes[j] + ' ' + opcodes[k])
len(asmopcodetrigram)
```

Out[33]:
```
17576
```

In [34]:
```
asmopcodetetragram = []
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        for k in range(0, len(opcodes)):
            for l in range(0, len(opcodes)):
                asmopcodetetragram.append(v + '
' + opcodes[j] + ' ' + opcodes[k] + ' ' + opcod
es[l])
len(asmopcodetetragram)
```

Out[34]:
```
456976
```

In [ ]:
```
op_file = open("opcode_file.txt", "w+")
for asmfile in os.listdir('asmFiles'):
    opcode_str = ""
    with codecs.open('asmFiles/' + asmfile, enc
oding='cp1252', errors ='replace') as fli:
        for lines in fli:
```

```
            line = lines.rstrip().split()
            for li in line:
                if li in opcodes:
                    opcode_str += li + ' '
    op_file.write(opcode_str + "\n")
op_file.close()
```

In [47]:

```
vect_bi = CountVectorizer(ngram_range=(2, 2), v
ocabulary = asmopcodebigram)
opcodebivect = scipy.sparse.csr_matrix((10868,
len(asmopcodebigram)))
for i in range(10868):
    raw_opcode = open('opcode_file.txt').read()
.split('\n')
    opcodebivect[i, :] += scipy.sparse.csr_matr
ix(vect_bi.transform([raw_opcode[i]]))

vect_tri = CountVectorizer(ngram_range=(3, 3),
vocabulary = asmopcodetrigram)
opcodetrivect = scipy.sparse.csr_matrix((10868,
len(asmopcodetrigram)))
for i in range(10868):
    opcodetrivect[i, :] += scipy.sparse.csr_mat
rix(vect_tri.transform([raw_opcode[i]]))

vect_tra = CountVectorizer(ngram_range=(4, 4),
vocabulary = asmopcodetetragram)
opcodetetravect = scipy.sparse.csr_matrix((1086
8, len(asmopcodetetragram)))
for i in range(10868):
    opcodetetravect[i, :] += scipy.sparse.csr_m
atrix(vect_tra.transform([raw_opcode[i]]))
```

In [48]:

```
opcodebivect
```

Out[48]:
```
<10868x676 sparse matrix of type '<class
'numpy.float64'>'
        with 1877309 stored elements in C
ompressed Sparse Row format>
```

In [52]:
```
opcodetrivect
```

Out[52]:
```
<10868x17576 sparse matrix of type '<clas
s 'numpy.float64'>'
        with 7332672 stored elements in C
ompressed Sparse Row format>
```

In [55]:
```
opcodetetravect
```

Out[55]:
```
<10868x456976 sparse matrix of type '<cla
ss 'numpy.float64'>'
        with 16605229 stored elements in
Compressed Sparse Row format>
```

Feature extraction as image from asmfiles

In [35]:
```
#ref : https://www.youtube.com/watch?v=VLQTR1LG
z5Y#t=13m11s
import array
for asmfile in os.listdir("asmFiles"):
    filename = asmfile.split('.')[0]
    file = codecs.open("asmFiles/" + asmfile,
'rb')
    filelen = os.path.getsize("asmFiles/" + asm
file)
    width = int(filelen ** 0.5)
    rem = int(filelen / width)
    arr = array.array('B')
```

```
    arr.frombytes(file.read())
    file.close()
    reshaped = np.reshape(arr[:width * width],
(width, width))
    reshaped = np.uint8(reshaped)
    scipy.misc.imsave('asm_image/' + filename +
'.png',reshaped)
```

extracting the first 200 Image Pixels

In [38]:
```
import cv2
imagefeatures = np.zeros((10868, 200)) # Creati
ng a null matrix of 200 size vectors.
for i, asmfile in enumerate(os.listdir("asmFile
s")):
    img = cv2.imread("asm_image/" + asmfile.spl
it('.')[0] + '.png')
    img_arr = img.flatten()[:200] #therefore ex
tracting only 200 features.
    imagefeatures[i, :] += img_arr
```

In [68]:
```
#extracting the column names of first 200 pixel
s.
imgfeatures_name = []
for i in range(200):
    img_features_name.append('pix' + str(i))
#Normalizing the features.
imgdf = pd.DataFrame(normalize(imagefeatures, a
xis = 0), columns = imgfeatures_name)
imgdf['ID'] = result.ID
```

In [70]:
```
imgdf.head()
```

Out[70]:

	pix0	pix1	pix2	pix3	pix4
0	0.010268	0.010268	0.010268	0.008033	0.008033
1	0.006560	0.006560	0.006560	0.013504	0.013504
2	0.010268	0.010268	0.010268	0.008033	0.008033
3	0.010268	0.010268	0.010268	0.008033	0.008033
4	0.010268	0.010268	0.010268	0.008033	0.008033

5 rows × 201 columns

extracting the important Features Using Random Forest

In [38]:
```
#ref:https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e
def imp_features_indices(data, features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    return imp_feature_indx[:keep]
```

In [ ]:
```
sns.set()
plt.figure(figsize = (10, 5))
ax = sns.barplot(x = imp_feature_name, y = imp_value)
ax.set_xticklabels(labels = imp_feature_name, r
```

```
otation = 45)
sns.set_palette(reversed(sns.color_palette("hus
l", 10)), 10)
plt.title('Important Features')
plt.xlabel('Feature Names')
plt.ylabel('Importance')
```

extracting Important Feature on Opcode N_grams

In [44]:
```
op_bi_indxes = imp_features_indices(normalize(o
pcodebivect, axis = 0), asmopcodebigram, 200)
op_tri_indxes = imp_features_indices(normalize(
opcodetrivect, axis = 0), asmopcodetrigram, 200
)
op_tetra_indxes = imp_features_indices(normaliz
e(opcodetetravect, axis = 0), asmopcodetetragra
m, 200)
```

In [45]:
```
op_bi_df = pd.SparseDataFrame(normalize(opcodeb
ivect, axis = 0), columns = asmopcodebigram)
op_bi_df = op_bi_df.loc[:, np.intersect1d(op_bi
_df.columns, np.take(asmopcodebigram, op_bi_ind
xes))]

op_tri_df = pd.SparseDataFrame(normalize(opcode
trivect, axis = 0), columns = asmopcodetrigram)
op_tri_df = op_tri_df.loc[:, np.intersect1d(op_
tri_df.columns, np.take(asmopcodetrigram, op_tr
i_indxes))]

op_tetra_df = pd.SparseDataFrame(normalize(opco
detetravect, axis = 0), columns = asmopcodetetr
agram)
```

```
op_tetra_df = op_tetra_df.loc[:, np.intersect1d
(op_tetra_df.columns, np.take(asmopcodetetragra
m, op_tetra_indxes))]
```

In [48]:
```
op_bi_df['ID'] = result.ID
op_bi_df.head()
```

Out[48]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop
0	0.031815	0.003894	0.000000	0.00042	0.000000
1	0.000000	0.000649	0.000000	0.00021	0.000374
2	0.000000	0.000000	0.000000	0.00000	0.000000
3	0.000000	0.000101	0.000000	0.00007	0.000000
4	0.000362	0.001156	0.001467	0.00028	0.000374

5 rows × 201 columns

In [43]:
```
op_tri_df['ID'] = result.ID
op_tri_df.head()
```

Out[43]:

	add cmp jmp	add mov add	add mov cmp	add mov jmp	add mov mov
0	0.000000	0.002183	0.001340	0.001563	0.003593
1	0.000000	0.001364	0.000670	0.000625	0.002705
2	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.001292	0.001091	0.004914	0.002814	0.014009

5 rows × 201 columns

In [53]:
```
op_tetra_df['ID'] = result.ID
op_tetra_df.head()
```

Out[53]:

	add mov add mov	add mov add pop	add mov cmp jnb	add mov mov add	add mov mov mov
0	0.001593	0.007668	0.000000	0.002031	0.002517
1	0.000000	0.007668	0.000000	0.001625	0.002760
2	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.002125	0.000000	0.023352	0.023558	0.006657

5 rows × 201 columns

extracting Important Feature on Byte Bi-Gram

In [54]:
```
byte_bi_indxes = imp_features_indices(normalize
(bytebigram_vect, axis = 0), byte_bigram_vocab,
300)

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

byte_bi_df = pd.SparseDataFrame(top_byte_bi, co
lumns = np.take(byte_bigram_vocab, byte_bi_indx
es))
```

Advanced features

Concating 300 bytebigram,200 opcode bigram,200 opcode trigram,200 opcode tetragram ,first 200 image pixels

In [74]:
```
final_data = pd.concat([result_x, op_bi_df, op_
tri_df, op_tetra_df, byte_bi_df,img_df], axis =
1, join = 'inner')
#final_data.head()
```

Train cv and test split 80% and 20%

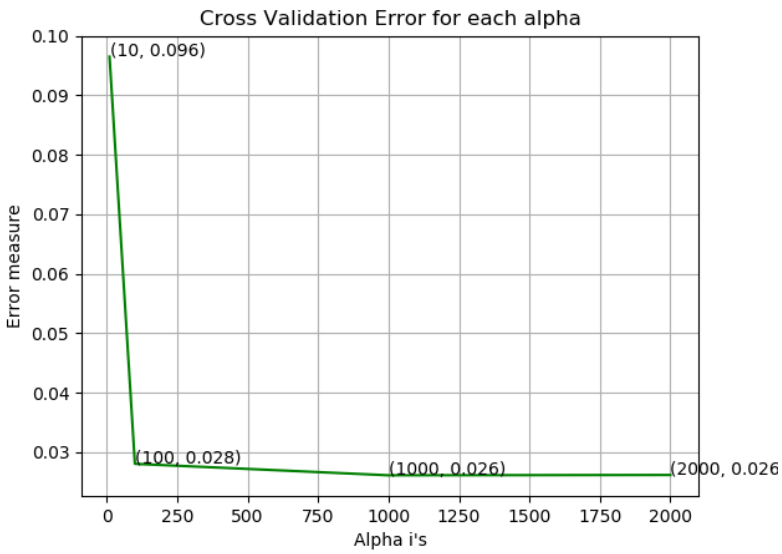
In [37]:
```
x_train_final, x_test_final, y_train_final, y_t
est_final = train_test_split(final_data, result
_y, stratify = result_y, test_size = 0.20)
x_trn_final, x_cv_final, y_trn_final, y_cv_fina
l = train_test_split(x_train_final, y_train_fin
al, stratify = y_train_final, test_size = 0.20)
```

Xgboost modelling with all the features

In [29]:
```
alpha=[10,100,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(x_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(x_cfl, met
hod="sigmoid")
    sig_clf.fit(x_trn_final, y_trn_final)
```

```
    predict_y = sig_clf.predict_proba(x_cv_fina
l)
    cv_log_error_array.append(log_loss(y_cv_fin
al, 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_a
rray,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
log_loss for c =  10 is 0.096496484676351
32
log_loss for c =  100 is 0.02802699487589
2948
log_loss for c =  1000 is 0.0261010230172
4636
log_loss for c =  2000 is 0.0261557646431
62237
```



In [84]:

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

predict_y = sig_clf.predict_proba(x_trn_final)
print ('For values of best alpha = ', alpha[bes
t_alpha], "The train log loss is:",log_loss(y_t
rn_final, predict_y))
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best
_alpha], "The cross validation log loss is:",lo
g_loss(y_cv_final, predict_y))
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best
_alpha], "The test log loss is:",log_loss(y_tes
t_final, predict_y))
```

```
For values of best alpha =  0.01 The trai
n log loss is: 0.010187974436441512
For values of best alpha =  0.01 The cros
```

```
s validation log loss is: 0.0239576285661
4576
For values of best alpha =  0.01 The test
log loss is: 0.018309505637434106
```

Conclusion:

In [8]:

```
from prettytable import PrettyTable
p = PrettyTable()
p.field_names = ["Multiclass Classification Mod
els",'Type of Feature_Files','Test Logloss']
p.add_row(["random","bytefiles","2.45"])
p.add_row(["Knn","bytefiles","0.48"])
p.add_row(["Logistic Regression","bytefiles",
"0.52"])
p.add_row(["Random Forest Classifier ","bytefil
es","0.06"])
p.add_row(["XgBoost ","bytefiles","0.07"])
p.add_row(["knn","asmfiles","0.21"])
p.add_row(["Logistic Regression","asmfiles","0.
38"])
p.add_row(["Random Forest Classifier ","asmfile
s","0.03"])
p.add_row(["XgBoost ","asmfiles","0.04"])
p.add_row(["Random Forest Classifier ","bytefil
es+asmfiles","0.04"])
p.add_row(["XgBoost ","bytefiles+asmfiles","0.0
2"])
p.add_row(["XgBoost ","bytefiles+asmfiles+advan
ced features","0.01"])
print(p)
```

```
+---------------------------------+-----
---------------------------------+-------
```

```
-------+
| Multiclass Classification Models |
Type of Feature_Files          | Test Logl
oss |
+---------------------------------+-----
---------------------------------+-------
-------+
|               random               |
bytefiles                 |     2.45     |
|                Knn                 |
bytefiles                 |     0.48     |
|       Logistic Regression          |
bytefiles                 |     0.52     |
|    Random Forest Classifier        |
bytefiles                 |     0.06     |
|             XgBoost                |
bytefiles                 |     0.07     |
|                knn                 |
asmfiles                  |     0.21     |
|       Logistic Regression          |
asmfiles                  |     0.38     |
|    Random Forest Classifier        |
asmfiles                  |     0.03     |
|             XgBoost                |
asmfiles                  |     0.04     |
|    Random Forest Classifier        |
bytefiles+asmfiles           |     0.04
|
|             XgBoost                |
bytefiles+asmfiles           |     0.02
|
|             XgBoost                | byte
files+asmfiles+advanced features |     0.
01     |
+---------------------------------+-----
---------------------------------+-------
```

```
-------+
```

- **since the dataset is nearly 200GB, it is highly computational expensive(It took me nearly 8 days to get the above results with 8gb RAM laptop.).**
- **The whole dataset is having 2 types of files(ie. Byte files and asm files).Thus we seperated the both files and preprocessed with Unigram vectorization and finally trained models individually to find the impact of each feature on our problem.**
- **Then combined the both files and applied the ML models, results stated above.**
- **Now we extracted bigram vectorization on bytefiles and N_gram(2,3,4-grams) vectorization on important opcodes of asmfiles.**
- **Applied the ML models with xgboost multiclass classification. Finally got logloss 0.01**

In [9]:
```
#references:
#https://medium.com/datadriveninvestor/what-ive-learned-microsoft-malware-prediction-competition-on-kaggle-3c8189dcc850
# https://www.youtube.com/watch?v=VLQTRlLGz5Y
#https://github.com/dchad/malware-detection
#https://github.com/sai977/microsoft-malware-detection
#https://www.kaggle.com/c/microsoft-malware-prediction/discussion/74639
#https://www.kaggle.com/c/microsoft-malware-prediction/discussion/74638
```

In [ ]:
```

```