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**

1. Kelihos_ver3
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,
s:nothing, gs:nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24 08                          lea     eax, [esp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                                mov     esi, ecx
.text:00401008 E8 1C 1B 00 00                       call    ??
0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08 BB 42 00                    mov     dword ptr [esi], offset o
f_42BB08
.text:00401013 8B C6                                mov     eax, esi
.text:00401015 5E                               pop     esi
.text:00401016 C2 04 00                             retn    4
.text:00401016                              ; ---------------------------------------------
-------------------------
.text:00401019 CC CC CC CC CC CC CC                 align 10h
.text:00401020 C7 01 08 BB 42 00                    mov     dword ptr [ecx], offset o
f_42BB08
.text:00401026 E9 26 1C 00 00                       jmp     sub_402C51
.text:00401026                              ; ---------------------------------------------
-------------------------
.text:0040102B CC CC CC CC CC                       align 10h
.text:00401030 56                               push    esi
.text:00401031 8B F1                                mov     esi, ecx
.text:00401033 C7 06 08 BB 42 00                    mov     dword ptr [esi], offset o
f_42BB08
.text:00401039 E8 13 1C 00 00                       call    sub_402C51
.text:0040103E F6 44 24 08 01                       test    byte ptr [esp+8], 1
.text:00401043 74 09                                jz      short loc_40104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E 00 00                       call    ??3@YAXPAX@Z    ; operato
delete(void *)
.text:0040104B 83 C4 04                             add     esp, 4
.text:0040104E
.text:0040104E                              loc_40104E:                 ; CODE XREF:
.text:00401043 j
.text:0040104E 8B C6                                mov     eax, esi
.text:00401050 5E                               pop     esi
.text:00401051 C2 04 00                             retn    4
.text:00401051                              ; ---------------------------------------------
-------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
```

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

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

2.2.1. Type of Machine Learning Problem

There are nine different classes of malware that we need to 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/AAB6EeInEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."
https://github.com/saicharanarishanapally/microsoft-malware-detection/blob/master/MicrosoftMalwareDetection.ipynb
https://github.com/mayank171986/Microsoft-Malware-Detection

3. Exploratory Data Analysis

In [1]:

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

In [ ]:

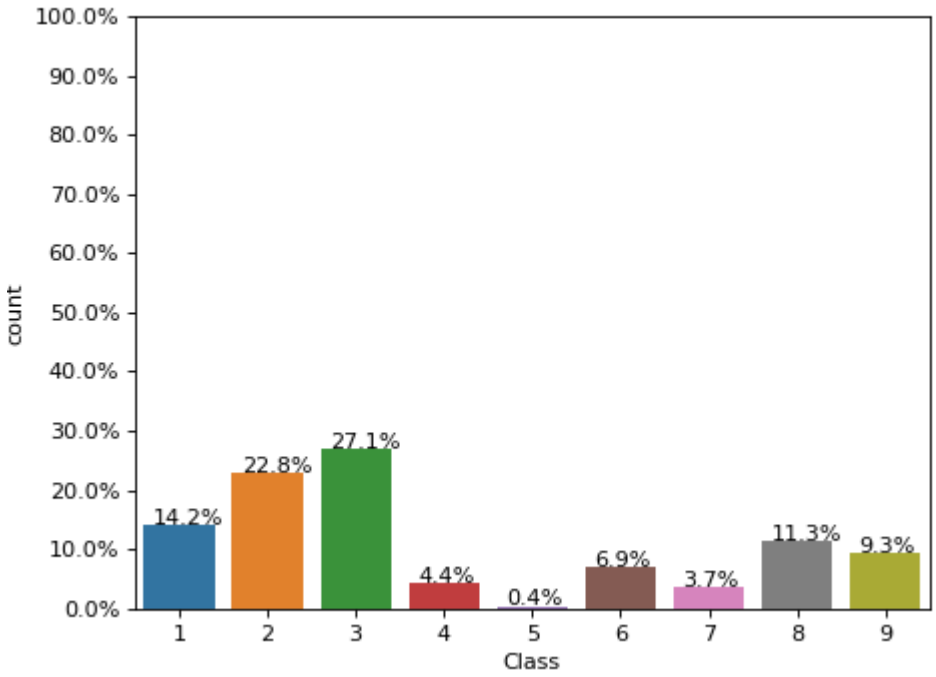
```
#separating byte files and asm files

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 wil
l 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,destination_2)
```

In [ ]:

Distribution of malware classes in whole data set

In [2]:

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

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

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



Feature extraction

File size of byte files as a feature

In [60]:

```
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_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
```

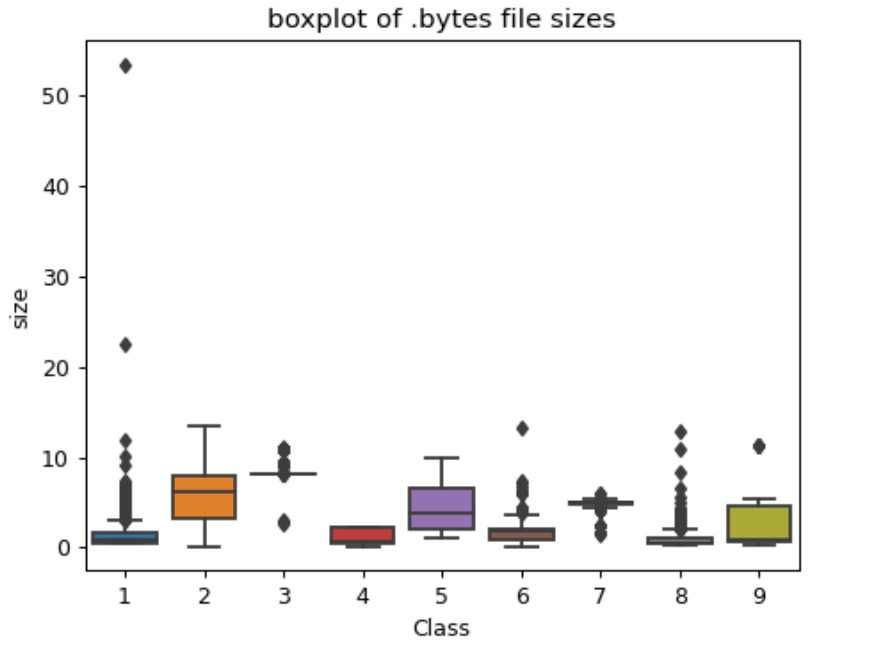
```
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting 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  CZvcR8GBrn5JIPdtqz1Y  8.941406      3
1  Al93Uy2JIpX5ikTNLj1d  0.339844      1
2  6ZY9Al0hSbQBGF1RcLVH  0.503906      8
3  6wQzVGgRDpW5ZuAMO0Us  5.496094      2
4  26Apnq34hwrYEbvSUoMV  3.585938      9
```

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()
```



feature extraction from byte files

In [ ]:

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

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        file=file.split('.')[0]
```

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

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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,
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,
e,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,5
,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
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,
5,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,c
,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
ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i, row in enumerate(feature_matrix[k]):
        if i!=len(feature_matrix[k])-1:
            byte_feature_file.write(str(row)+",")
        else:
            byte_feature_file.write(str(row))
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [61]:

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

Out[61]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8	f9	fa	fb	fc	fd
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687	3101	3211	3097	2758	3099
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536	439	281	302	7639	518

2 rows × 258 columns

In [62]:

```
data_size_byte.head(2)
```

Out[62]:

	ID	size	Class
0	CZvcR8GBrn5JlPdtqz1Y	8.941406	3
1	Al93Uy2JlpX5ikTNLj1d	0.339844	1

In [63]:

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

Out[63]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	f
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759	5753
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	17001	54902

2 rows × 260 columns

In [4]:

```
# 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(byte_features_with_size)
```

In [5]:

```
result.head(2)
```

Out[5]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.01356	0.
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.00192	0.

2 rows × 261 columns

In [7]:

```
result.shape
```

Out[7]:

```
(10868, 261)
```

In [7]:

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

Out[7]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9

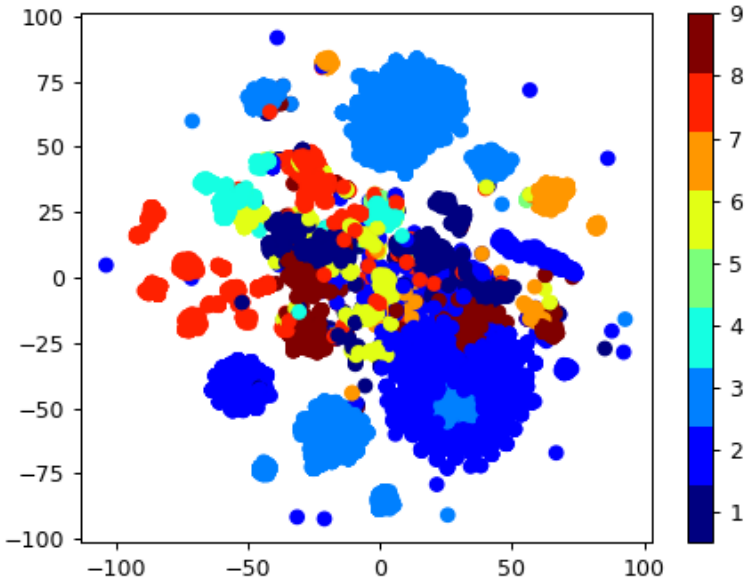
	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	9
0	0.000000	01azqd4lnC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.013560
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.001920
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	...	0.009804
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	...	0.002121
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	...	0.001530

5 rows × 261 columns

Multivariate Analysis

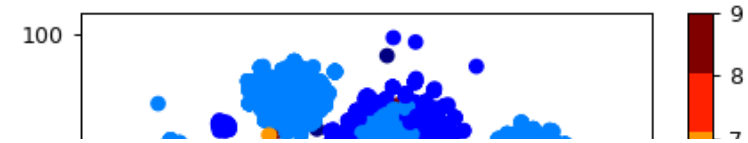
In [13]:

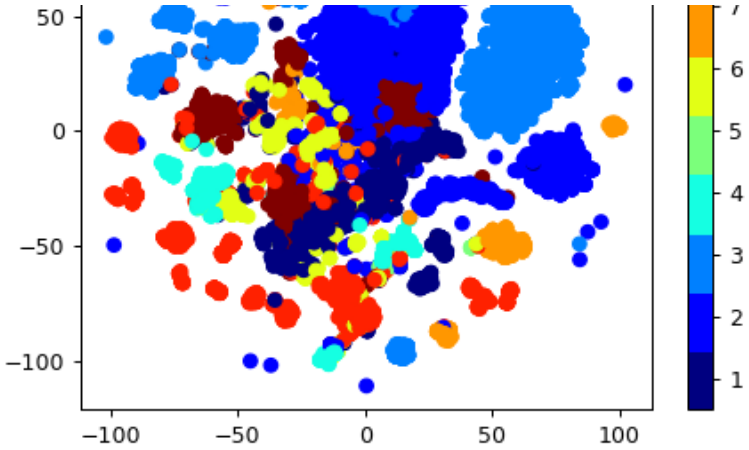
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))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



In [14]:

```
#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()
```





In [6]:

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

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

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

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

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

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

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

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

```
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

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 a ll 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

Feature extraction from asm files

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

In [ ]:

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

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

In [ ]:

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

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

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

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

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

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

```
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


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

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


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

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

In [7]:

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

Out[7]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	eb
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83	0	1
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12	0	1
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14	0	1
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	0	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	0	1

5 rows × 53 columns

Files sizes of each .asm file

In [7]:

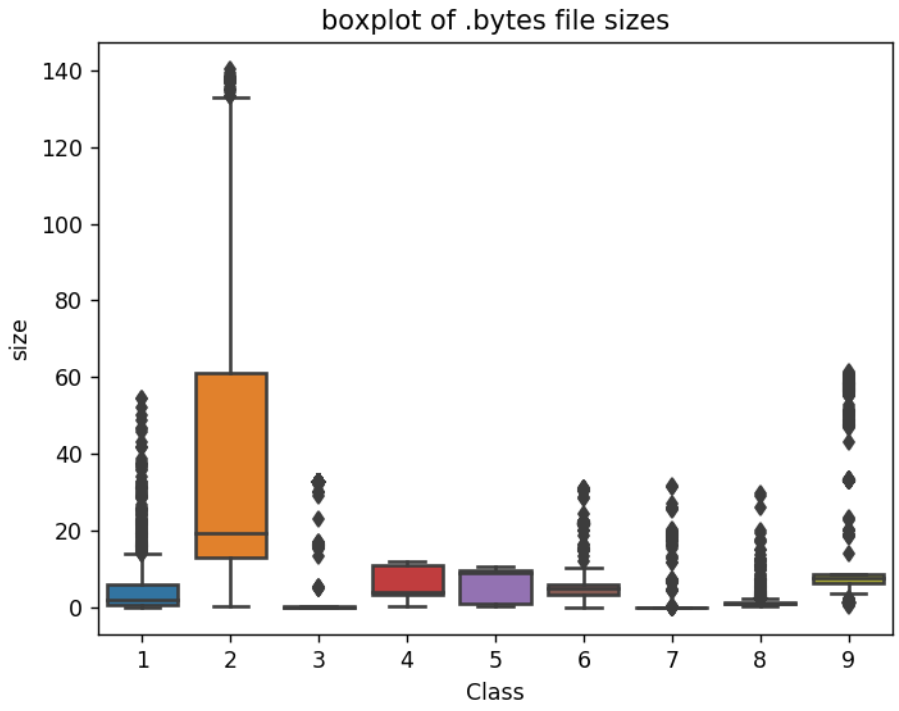
```
#file sizes of byte files

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 first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

```
                     ID       size  Class
0  85qIWFZNGAb6nLug14lH   0.214691      3
1  JNfrKZYIEmpVDBzn1kvL  66.865067      2
2  BCw58jIVumFW43RExqJk   0.174805      7
3  Iy6lQgsbZ9q0tfrT1hMe   1.283064      9
4  24NSkH5K03grqedBQM8T   0.215658      3
```

Distribution of .asm file sizes

In [ ]:

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



In [12]:

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

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

Out[12]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	es
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	4
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	

5 rows × 54 columns

In [8]:

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

Out[8]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000343	0.000746
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000343	0.000328
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000248	0.000475
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000114	0.000090

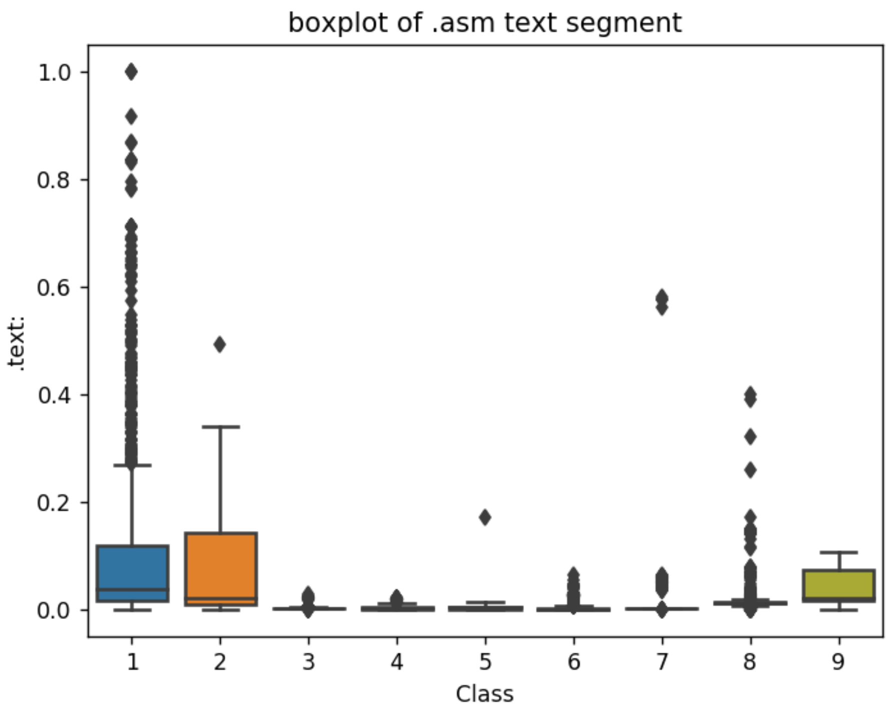
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	eax	esi
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000229	0.000102

5 rows × 53 columns

Univariate analysis on asm file features

In [ ]:

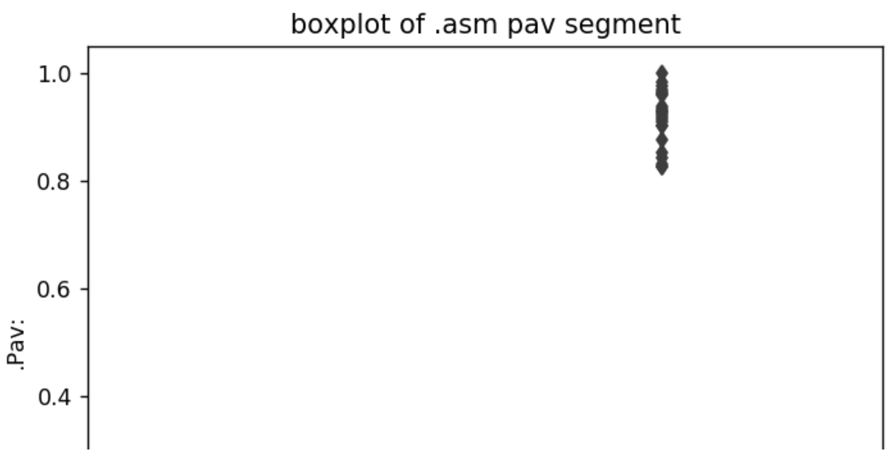
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```

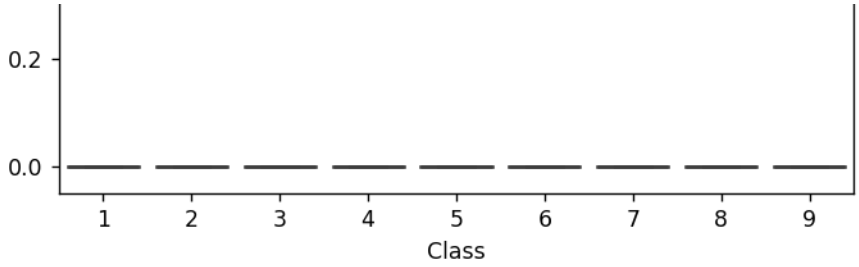



```
The plot is between Text and class
Class 1,2 and 9 can be easly separated
```

In [ ]:

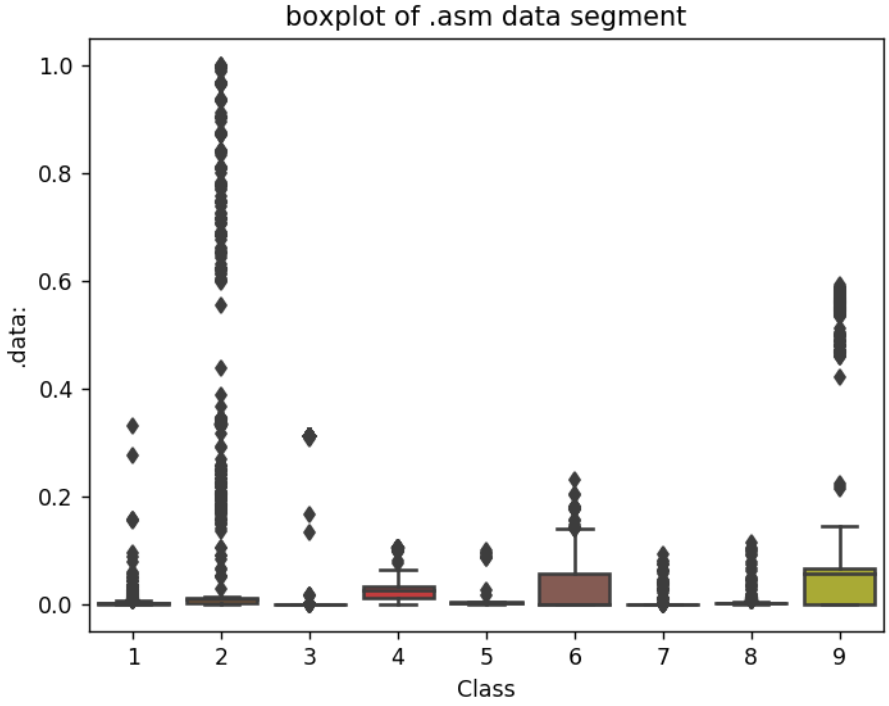
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```





In [ ]:

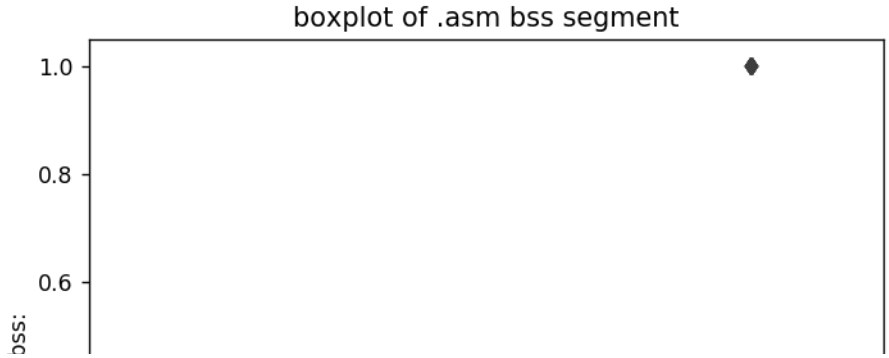
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```

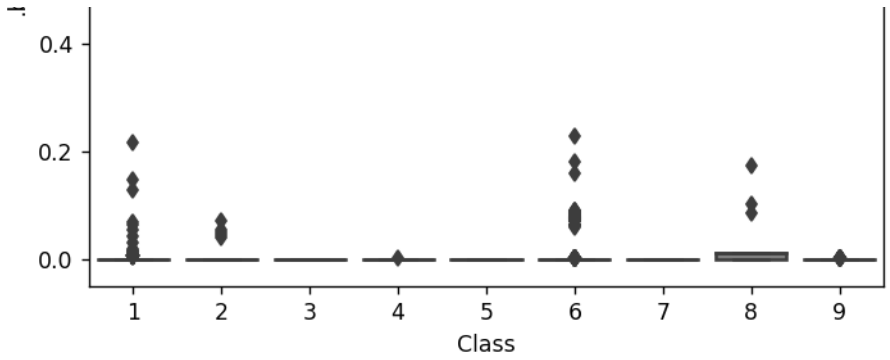


The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [ ]:

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

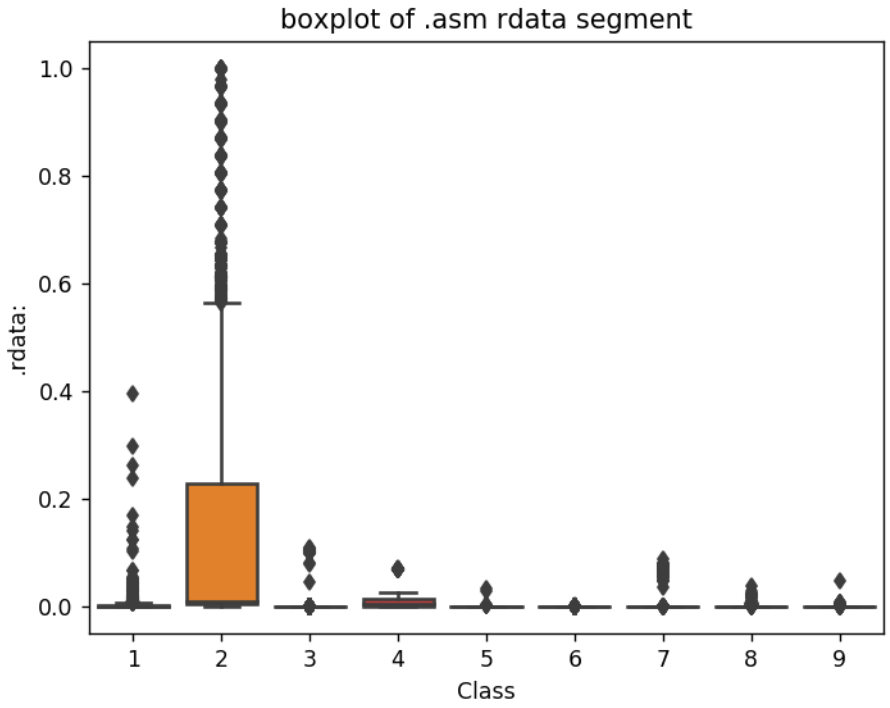




plot between bss segment and class label
very less number of files are having bss segment

In [ ]:

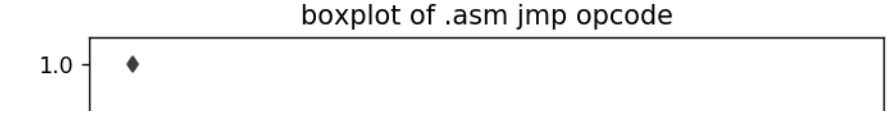
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```

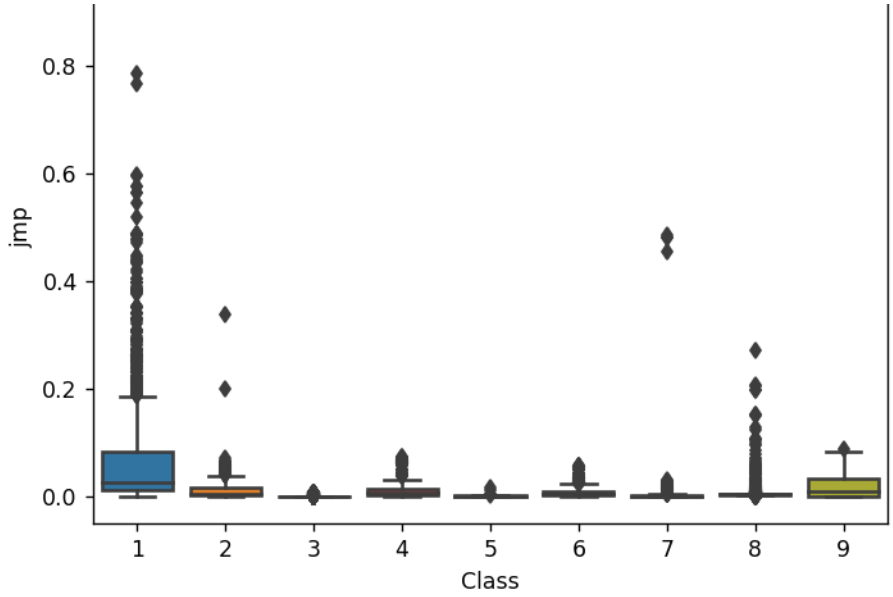


Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [ ]:

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

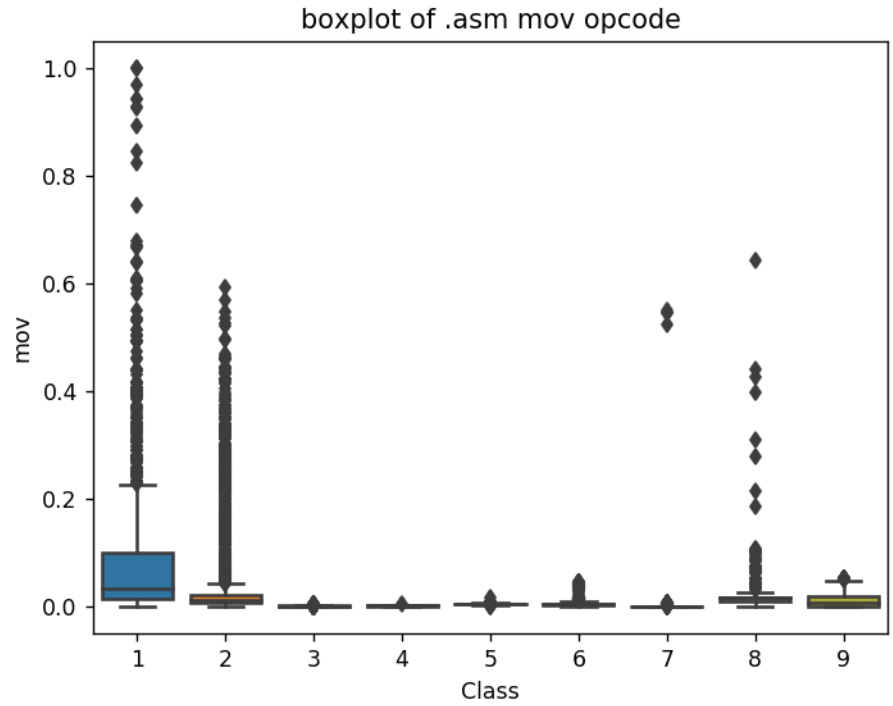




plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [ ]:

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

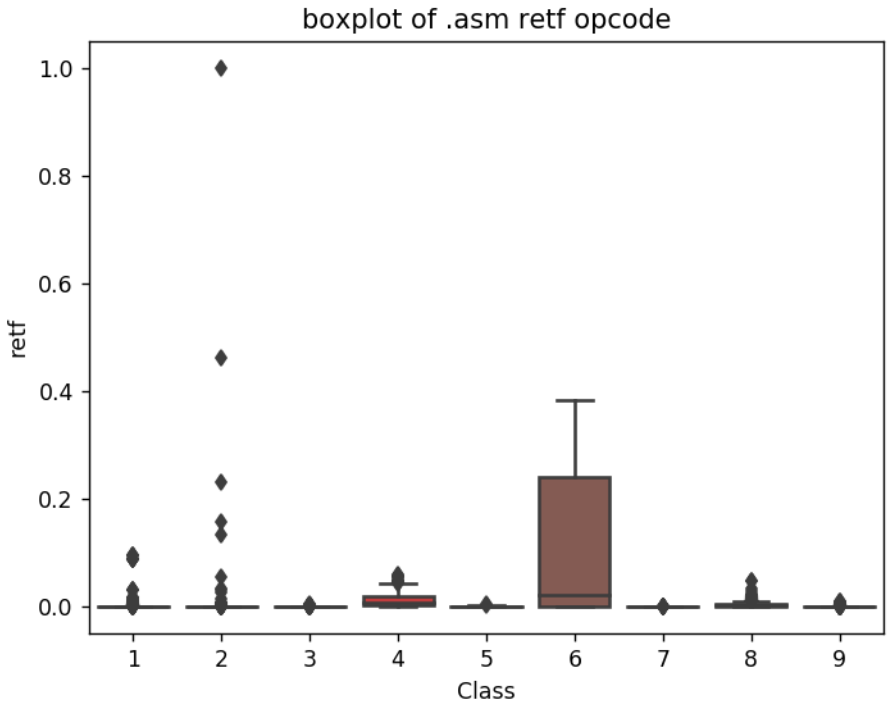


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [ ]:

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

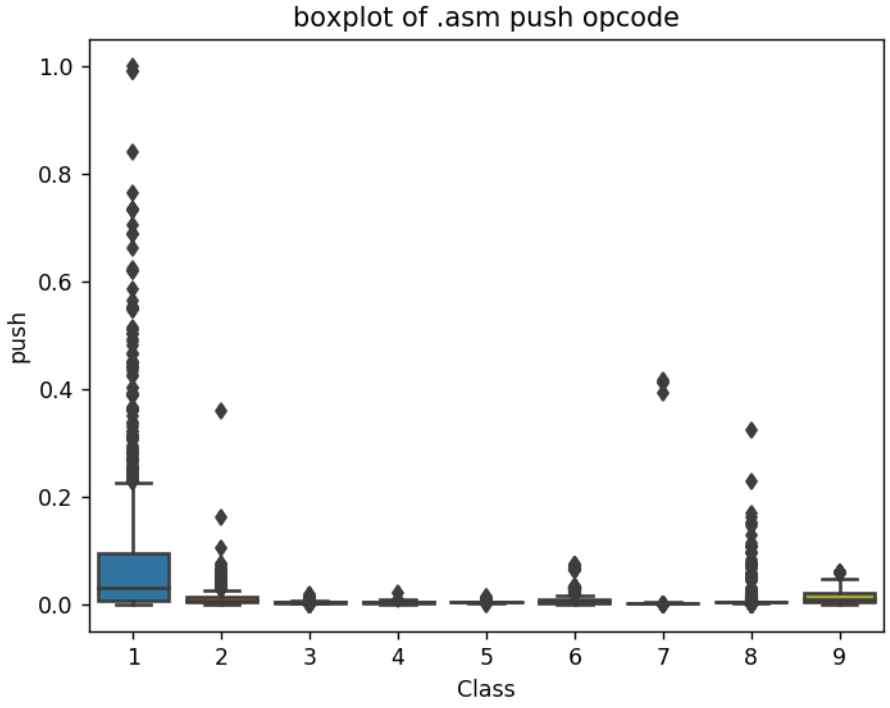
```
plt.show()
```



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

In [ ]:

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

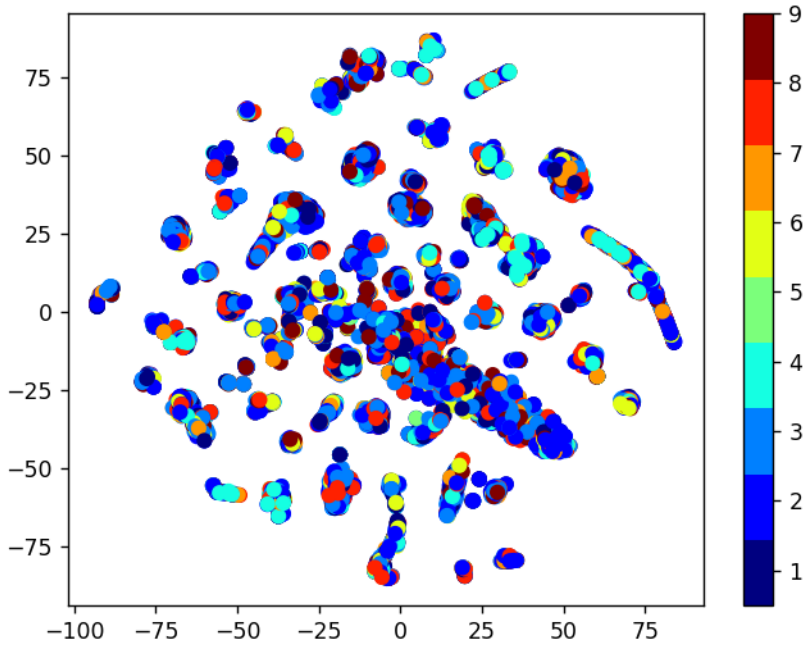


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

Multivariate Analysis on .asm file features

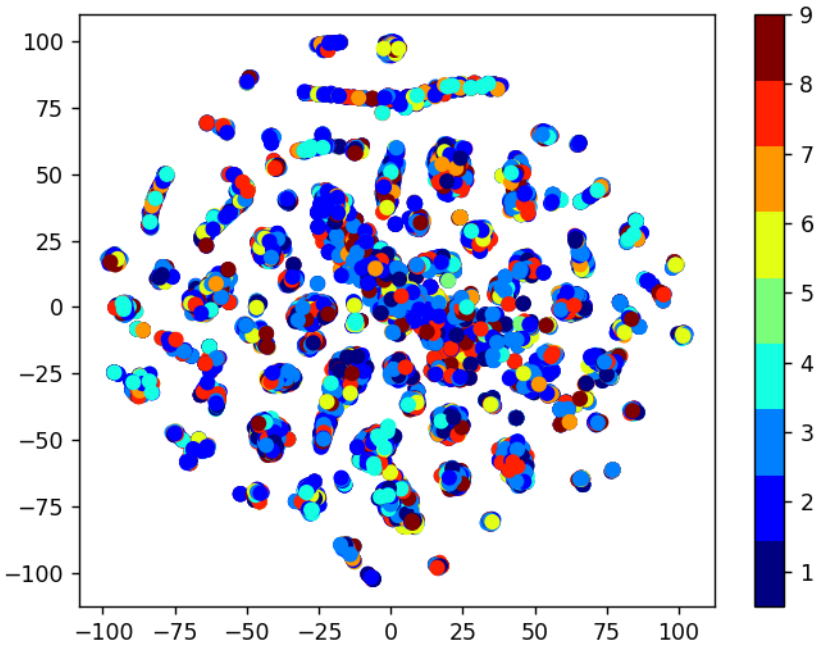
In [ ]:

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

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



In [ ]:

```
# by univariate analysis on the .asm file features we are getting very negligible information from

# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing tho
se features
# the plot looks very messy

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



TSNE for asm data with perplexity 50

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.

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

Merging both asm and byte file features

In [17]:

```
result.head()
```

Out[17]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.013560
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.001920
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	...	0.009804
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	...	0.002121
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	...	0.001530

5 rows × 261 columns

In [18]:

```
result_asm.head()
```

Out[18]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362

5 rows × 54 columns

In [10]:

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

```
(10868, 261)
(10868, 53)
```

In [9]:

```
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','Unnamed: 0'], axis=1)
#result_x = result_x.drop(['rtn','.BSS:','.CODE','Class','Unnamed: 0'], axis=1)
result_x.head()
```

Out[9]:

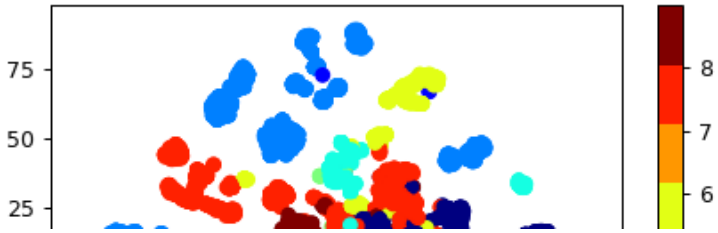
	ID	0	1	2	3	4	5	6	7	8	...	:dword	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.032784	0
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.010846	0
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.006773	0
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.001028	0
4	01SuzwMJElXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.009150	0

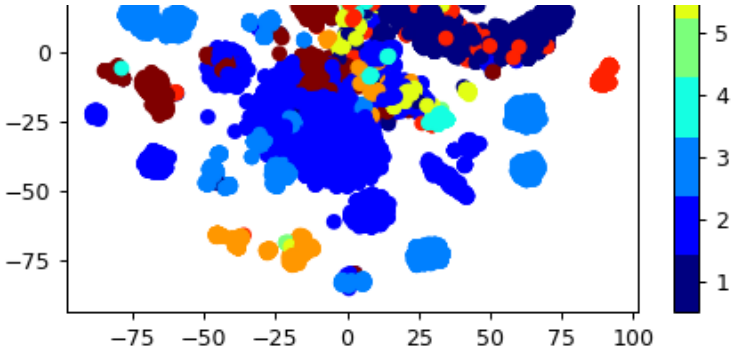
5 rows × 307 columns

Multivariate Analysis

In [20]:

```
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()
```





Splitting data into train, cv, test

In [56]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_
y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y
_train,test_size=0.20)
```

Applying Xgboost classifier usind randomsearch_cv

In [58]:

```
x_cfl=XGBClassifier()

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   30.6s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  5.6min remaining:  3.2min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  7.7min remaining:  2.3min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  9.7min remaining:  1.1min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 11.0min finished
```

Out[58]:

```
RandomizedSearchCV(cv=3,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None,
                                           n_estimators=100,...
                                           random_state=None, reg_alpha=None,
                                           reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
```

```
                    param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                         'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                           0.15, 0.2],
                                         'max_depth': [3, 5, 10],
                                         'n_estimators': [100, 200, 500, 1000,
                                                          2000],
                                         'subsample': [0.1, 0.3, 0.5, 1]},
                    verbose=10)
```

In [59]:

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

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.2, 'colsample_bytree': 0.
5}
```

In [60]:

```
x_cfl=XGBClassifier(n_estimators=200,max_depth=5,learning_rate=0.2,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 ("The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print("The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print("The test log loss is:",log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
The train log loss is: 0.011817622344896589
The cross validation log loss is: 0.0396769717768615
The test log loss is: 0.038946509342111836
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-60-b40a86e85137> in <module>
     10 predict_y = sig_clf.predict_proba(X_test_merge)
     11 print("The test log loss is:",log_loss(y_test_merge, predict_y))
---> 12 plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))

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

In [ ]:

```

```

Generating byte_bigrams

In [47]:

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

In [10]:

```
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,2(
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,
4,45,46,47,48,49,4a,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,(
,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,80,81,82,83,84,85,86,8
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,
b,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,(
,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
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [11]:

```
def byte_bigram():
    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])
    return byte_bigram_vocab
```

In [12]:

```
byte_bigram_vocab = byte_bigram()
```

In [15]:

```
len(byte_bigram_vocab)
```

Out[15]:

```
66049
```

In [20]:

```
byte_bigram_vocab[:5]
```

Out[20]:

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

In [13]:

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

In [ ]:

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

In [29]:

```
F = open('byte_bi.pckl','wb')
pickle.dump(bytebigram_vect,F)
F.close
```

Out[29]:

```
<function BufferedWriter.close>
```

In [28]:

```
F = open('byte_bi.pckl','rb')
bytebigram_vect = pickle.load(F)
F.close()
```

In [29]:

```
bytebigram_vect
```

Out[29]:

```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'
```

```
 with 497280662 stored elements in Compressed Sparse Row format>
```

In [34]:

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

In [14]:

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

In [29]:

```
byte_bigram_vect
```

Out[29]:

```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'
 with 497280662 stored elements in Compressed Sparse Column format>
```

ASM Image

In [19]:

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

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

In [8]:

```
def entropy(p,n):
    '''
    Calculate entropy of the file
    '''
    p_ratio = float(p)/(p+n)
    n_ratio = float(n)/(p+n)
    return -p_ratio*math.log(p_ratio) - n_ratio * math.log(n_ratio)
```

In [9]:

```
def info_gain(p0,n0,p1,n1,p,n):
    '''
    Calculate information gain
    '''
    return entropy(p,n) - float(p0+n0)/(p+n)*entropy(p0,n0) - float(p1+n1)/(p+n)*entropy(p1,n1)
```

In [10]:

```
def read_image(filename):
    '''
    Read image data
```

```
    '''
    f = open(filename,'rb')
    ln = os.path.getsize(filename) # length of file in bytes
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
    a.fromfile(f,ln-rem)
    f.close()
    g = np.reshape(a,(int(len(a)/width), width))
    g = np.uint8(g)
    g = np.resize(g, (1000,))
    return list(g)
```

In [11]:

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

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

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in range(1000)]
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image('asmFiles/'+ fname)
            outrows.append([file_id] + image_data)

            # Print progress
            if (idx+1) % 100 == 0:
                print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

        # Write remaining files
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []
```

In [12]:

```
# Now divide the train files into four groups for multiprocessing
start_time = tm.time()
tfiles = os.listdir('asmFiles')
quart = int(len(tfiles)/4)
# print(quart)
train1 = tfiles[:quart]
train2 = tfiles[quart:(2*quart)]
train3 = tfiles[(2*quart):(3*quart)]
train4 = tfiles[(3*quart):]
print(len(tfiles), quart, (len(train1)+len(train2)+len(train3)+len(train4)))
trains = [train1, train2, train3, train4]
p = Pool(4)
p.map(extract_asm_image_features, trains)
print("Elapsed time: {:.2f} hours.".format((tm.time() - start_time)/3600.0))
```

```
10868 2717 10868
3029 100 of 2717 files processed.
3030 100 of 2717 files processed.
3032 100 of 2717 files processed.
3031 100 of 2717 files processed.
3030 200 of 2717 files processed.
3029 200 of 2717 files processed.
3031 200 of 2717 files processed.
3032 200 of 2717 files processed.
3030 300 of 2717 files processed.
3031 300 of 2717 files processed.
```

3031 300 of 2717 files processed.
3032 300 of 2717 files processed.
3029 300 of 2717 files processed.
3030 400 of 2717 files processed.
3031 400 of 2717 files processed.
3029 400 of 2717 files processed.
3032 400 of 2717 files processed.
3030 500 of 2717 files processed.
3029 500 of 2717 files processed.
3031 500 of 2717 files processed.
3030 600 of 2717 files processed.
3032 500 of 2717 files processed.
3029 600 of 2717 files processed.
3030 700 of 2717 files processed.
3031 600 of 2717 files processed.
3032 600 of 2717 files processed.
3029 700 of 2717 files processed.
3031 700 of 2717 files processed.
3030 800 of 2717 files processed.
3032 700 of 2717 files processed.
3029 800 of 2717 files processed.
3031 800 of 2717 files processed.
3030 900 of 2717 files processed.
3032 800 of 2717 files processed.
3029 900 of 2717 files processed.
3031 900 of 2717 files processed.
3030 1000 of 2717 files processed.
3032 900 of 2717 files processed.
3031 1000 of 2717 files processed.
3029 1000 of 2717 files processed.
3032 1000 of 2717 files processed.
3029 1100 of 2717 files processed.
3030 1100 of 2717 files processed.
3031 1100 of 2717 files processed.
3032 1100 of 2717 files processed.
3029 1200 of 2717 files processed.
3030 1200 of 2717 files processed.
3031 1200 of 2717 files processed.
3032 1200 of 2717 files processed.
3031 1300 of 2717 files processed.
3029 1300 of 2717 files processed.
3030 1300 of 2717 files processed.
3032 1300 of 2717 files processed.
3031 1400 of 2717 files processed.
3030 1400 of 2717 files processed.
3029 1400 of 2717 files processed.
3032 1400 of 2717 files processed.
3031 1500 of 2717 files processed.
3029 1500 of 2717 files processed.
3030 1500 of 2717 files processed.
3032 1500 of 2717 files processed.
3031 1600 of 2717 files processed.
3030 1600 of 2717 files processed.
3029 1600 of 2717 files processed.
3032 1600 of 2717 files processed.
3030 1700 of 2717 files processed.
3031 1700 of 2717 files processed.
3029 1700 of 2717 files processed.
3030 1800 of 2717 files processed.
3032 1700 of 2717 files processed.
3029 1800 of 2717 files processed.
3031 1800 of 2717 files processed.
3029 1900 of 2717 files processed.
3032 1800 of 2717 files processed.
3031 1900 of 2717 files processed.
3030 1900 of 2717 files processed.
3032 1900 of 2717 files processed.
3029 2000 of 2717 files processed.
3031 2000 of 2717 files processed.
3030 2000 of 2717 files processed.
3032 2000 of 2717 files processed.
3029 2100 of 2717 files processed.
3031 2100 of 2717 files processed.
3030 2100 of 2717 files processed.
3032 2100 of 2717 files processed.
3029 2200 of 2717 files processed.
3031 2200 of 2717 files processed.
3030 2200 of 2717 files processed.

```
3030 2200 of 2717 files processed.
3031 2300 of 2717 files processed.
3032 2200 of 2717 files processed.
3029 2300 of 2717 files processed.
3030 2300 of 2717 files processed.
3032 2300 of 2717 files processed.
3031 2400 of 2717 files processed.
3030 2400 of 2717 files processed.
3029 2400 of 2717 files processed.
3032 2400 of 2717 files processed.
3031 2500 of 2717 files processed.
3029 2500 of 2717 files processed.
3030 2500 of 2717 files processed.
3032 2500 of 2717 files processed.
3029 2600 of 2717 files processed.
3032 2600 of 2717 files processed.
3030 2600 of 2717 files processed.
3031 2600 of 2717 files processed.
3031 2700 of 2717 files processed.
3029 2700 of 2717 files processed.
3030 2700 of 2717 files processed.
3032 2700 of 2717 files processed.
Elapsed time: 0.33 hours.
```

In [13]:

```
#merging all generated csv files

labels = pd.read_csv('trainLabels.csv')
d1 = pd.read_csv('3029-image-features-asm.csv')
d2 = pd.read_csv('3030-image-features-asm.csv')
d3 = pd.read_csv('3031-image-features-asm.csv')
d4 = pd.read_csv('3032-image-features-asm.csv')
d4.shape
```

Out[13]:

```
(2717, 1001)
```

In [14]:

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

Out[14]:

```
(10868, 1001)
```

In [15]:

```
data.reset_index(drop=True, inplace=True)
```

In [16]:

```
labels.head()
```

Out[16]:

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

In [17]:

```
sorted_train_data = data.sort_values(by='filename', axis=0, ascending=True, inplace=False)
```

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

In [18]:

```
X.shape, y.shape
```

Out[18]:

```
((10868, 1000), (10868,))
```

Selecting top 50% variance asm_image features

In [19]:

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

Out[19]:

```
(10868, 500)
```

In [20]:

```
selected_names = fsp.get_support(indices=True)
selected_names = selected_names + 1
selected_names
```

Out[20]:

```
array([  2,   4,   5,  15,  21,  22,  24,  25,  26,  27,  29,  30,  32,
        33,  34,  35,  41,  42,  43,  44,  48,  50, 125, 126, 135, 136,
       138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 151, 152,
       154, 155, 156, 157, 158, 160, 161, 162, 163, 164, 165, 167, 169,
       173, 174, 179, 186, 188, 190, 198, 201, 202, 205, 215, 216, 217,
       219, 220, 221, 222, 223, 224, 226, 227, 229, 236, 240, 241, 242,
       243, 244, 245, 246, 247, 248, 249, 252, 253, 260, 261, 262, 263,
       264, 265, 266, 267, 268, 269, 271, 272, 273, 282, 287, 291, 292,
       293, 294, 295, 296, 297, 307, 308, 310, 311, 312, 313, 314, 315,
       316, 317, 318, 319, 321, 323, 326, 327, 328, 330, 334, 337, 338,
       339, 340, 341, 343, 344, 345, 346, 349, 350, 351, 352, 353, 354,
       356, 357, 358, 359, 366, 367, 368, 370, 371, 372, 373, 374, 375,
       376, 378, 379, 380, 381, 384, 385, 386, 387, 388, 390, 391, 392,
       399, 400, 401, 402, 403, 404, 405, 408, 409, 410, 412, 413, 414,
       415, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431,
       436, 437, 439, 440, 441, 442, 443, 445, 446, 447, 448, 449, 450,
       451, 452, 453, 457, 458, 459, 460, 461, 464, 465, 466, 467, 477,
       478, 479, 480, 481, 482, 538, 539, 555, 556, 557, 558, 559, 560,
       561, 563, 564, 567, 568, 571, 572, 573, 580, 581, 582, 583, 584,
       585, 586, 587, 588, 589, 590, 597, 598, 600, 601, 602, 603, 606,
       607, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624,
       627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 640, 641,
       642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654,
       655, 656, 657, 658, 659, 662, 664, 670, 671, 672, 673, 674, 675,
       676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688,
       689, 691, 692, 693, 694, 695, 696, 701, 702, 703, 704, 708, 709,
       711, 712, 713, 714, 715, 717, 718, 719, 720, 721, 722, 723, 724,
       725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 738,
       739, 740, 743, 744, 751, 752, 753, 754, 755, 756, 757, 758, 759,
       760, 761, 762, 763, 765, 774, 775, 776, 777, 778, 779, 780, 781,
       782, 784, 785, 786, 787, 788, 789, 793, 798, 801, 802, 813, 814,
       818, 819, 820, 830, 831, 835, 836, 837, 838, 840, 841, 847, 848,
       849, 850, 851, 852, 853, 855, 856, 857, 866, 867, 868, 869, 870,
       873, 874, 875, 876, 877, 878, 879, 882, 898, 899, 904, 907, 908,
       919, 920, 923, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939,
       940, 941, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957,
       958, 959, 960, 961, 962, 963, 965, 966, 967, 968, 973, 974, 975,
       976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 989, 990, 991,
       992, 995, 996, 997, 998, 9991)
```

In [21]:

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

Out[21]:

	filename	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23	ASM_24	ASM_25	...	ASM_984	ASM_
5061	01IsoiSMh5gxyDYTI4CB	116	120	116	9	32	32	32	32	32	...	54	
5350	01SuzwMJEIXsK7A8dQbI	69	68	69	48	9	9	13	10	72	...	32	
2765	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13	10	72	...	32	
3222	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13	10	72	...	32	
3562	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13	10	72	...	83	

5 rows × 501 columns

In [22]:

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

In [27]:

```
asm_img_df = pd.read_csv('sorted-features-asm-50percent.csv')
asm_img_df.shape
```

Out[27]:

```
(10868, 501)
```

In [28]:

```
asm_img_df.rename(columns={'filename': 'ID'}, inplace=True)
```

In [29]:

```
asm_img_df.shape
```

Out[29]:

```
(10868, 501)
```

In [30]:

```
asm_img_df.head()
```

Out[30]:

	ID	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23	ASM_24	ASM_25	...	ASM_984	ASM_98
0	01IsoiSMh5gxyDYTI4CB	116	120	116	9	32	32	32	32	32	...	54	4
1	01SuzwMJEIXsK7A8dQbI	69	68	69	48	9	9	13	10	72	...	32	1
2	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13	10	72	...	32	1
3	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13	10	72	...	32	1
4	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13	10	72	...	83	8

5 rows × 501 columns

Important features selection

Important features selection

In [15]:

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

In [ ]:

```

```

Top 300 bigram_byte_features

In [16]:

```
byte_bi_indxes = imp_features(byte_bigram_vect, byte_bigram_vocab, 300)
```

In [33]:

```
np.save('byte_bi_indx', byte_bi_indxes)
```

In [41]:

```
byte_bi_indxes = np.load('byte_bi_indx.npy')
```

In [17]:

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

In [18]:

```
byte_bi_df = pd.DataFrame(top_byte_bi, columns = np.take(byte_bigram_vocab, byte_bi_indxes))
```

In [36]:

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

In [40]:

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

In [20]:

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

In [21]:

```
byte_bi_df.head()
```

Out[21]:

	00 00	ff ff	00 57	00 52	fd ff	02 00	b9 00	00 5a	00 82	05 00	...	6f 73	0c 00	00 0
0	0.001081	0.000084	0.001180	0.000310	0.000599	0.000497	0.000720	0.000480	0.000569	0.001080	...	0.000200	0.000411	0.00028
1	0.002139	0.001754	0.000240	0.000301	0.000274	0.000607	0.000111	0.000568	0.000348	0.000857	...	0.000114	0.000444	0.00007
2	0.005542	0.003399	0.001859	0.000365	0.003317	0.005805	0.000480	0.000114	0.000411	0.002023	...	0.000200	0.000756	0.00022
3	0.002863	0.002340	0.003259	0.000256	0.002469	0.002351	0.000111	0.000202	0.001390	0.002298	...	0.000228	0.001874	0.00000
4	0.051411	0.000277	0.025732	0.010971	0.000399	0.013785	0.022637	0.015364	0.035888	0.020372	...	0.000171	0.019856	0.01047

5 rows × 301 columns

In [22]:

```
byte_bi_df.shape
```

Out[22]:

```
(10868, 301)
```

Merging byte_features, byte_bigrams, asm_features and asm_image_features

In [31]:

```
data_x = pd.merge(result_x, byte_bi_df, on='ID', how='left')
data_x = pd.merge(data_x, asm_img_df, on='ID', how='left')
data_y = result_y
data_x.head()
```

Out[31]:

	ID	0	1	2	3	4	5	6	7	8	...	ASM_984
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	32
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	54
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	32
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	83
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	32

5 rows × 1107 columns

In [50]:

```
data_x = data_x.drop(['ID'], axis=1)
```

In [51]:

```
data_x.shape
```

Out[51]:

```
(10868, 2807)
```

In [52]:

```
data_y.shape
```

Out[52]:

```
(10868,)
```

In [53]:

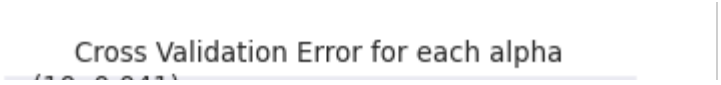
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(data_x, data_y,stratify=data_y,test
_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y
_train,test_size=0.20)
```

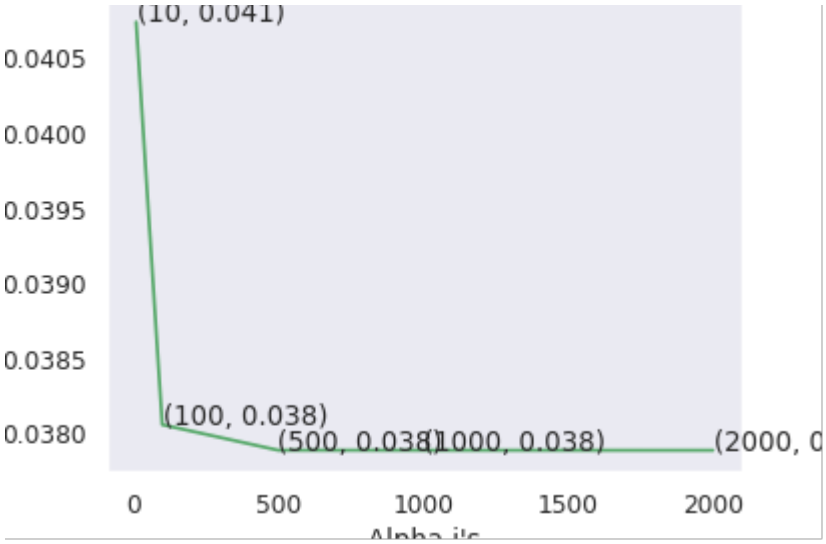
Xgboost classifier

In [54]:

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

# find more about XGBClassifier function here
http://xgboost.readthedocs.io/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, verbo
se=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 no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 0.040745030503098176
log_loss for c =  100 is 0.03805794664523516
log_loss for c =  500 is 0.0378876437712408
log_loss for c =  1000 is 0.0378872745742807
log_loss for c =  2000 is 0.03788708484661639
```






```
CPU times: user 14h 44min 7s, sys: 2min 15s, total: 14h 46min 22s
Wall time: 1h 54min 9s
```

In [25]:

```
x_cfl=XGBClassifier(n_estimators=1000)
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 ("The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print("The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print("The test log loss is:",log_loss(y_test_merge, predict_y))
```

```
The train log loss is: 0.012823403314118013
The cross validation log loss is: 0.041225070212975355
The test log loss is: 0.04371007724943907
```

Xgboost classifier using Random_search_cv

In [23]:

```
x_cfl=XGBClassifier()

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed: 10.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 58.4min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed: 71.1min remaining: 41.2min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 78.4min remaining: 23.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 83.6min remaining:  9.3min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 89.4min finished
```

Out[23]:

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

In [24]:

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

```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.2, 'colsample_bytree': 0
.5}
```

In [57]:

```
x_cfl=XGBClassifier(n_estimators=500,max_depth=10,learning_rate=0.2,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 ("The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print("The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print("The test log loss is:",log_loss(y_test_merge, predict_y))
```

```
The train log loss is: 0.015394929702521432
The cross validation log loss is: 0.060668493001928156
The test log loss is: 0.058978369974757985
```

In [ ]:

```

```

Opcode bigram and trigram

In [55]:

```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','i
mul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
```

In [56]:

```
def asmopcodebigram():
    asmopcodebigram = []
```

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

In [57]:

```
asmopcodebigram = asmopcodebigram()
```

In [58]:

```
len(asmopcodebigram)
```

Out[58]:

```
676
```

In [79]:

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

In [80]:

```
asmopcodetrigram = asmopcodetrigram()
```

In [81]:

```
len(asmopcodetrigram)
```

Out[81]:

```
17576
```

Collecting opcodes from each asm file

In [4]:

```
# dividing asmfiles into chunks for multiprocessing
tfiles = os.listdir("asmFiles")
quart = int(len(tfiles)/5)

train1 = tfiles[:quart]
train2 = tfiles[quart:(2*quart)]
train3 = tfiles[(2*quart):(3*quart)]
train4 = tfiles[(3*quart):(4*quart)]
train5 = tfiles[(4*quart):]

print(len(tfiles), quart, (len(train1)+len(train2)+len(train3)+len(train4)+len(train5)))
```

```
10868 2173 10868
```

In [9]:

```
from tqdm import tqdm
```

In [5]:

```
def firstprocess():
    op_file = open("first.txt", "w+")
    for asmfile in tqdm(train1):    #for each asmfile
        opcode_str = ""
```

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

def secondprocess():
    op_file = open("second.txt", "w+")
    for asmfile in tqdm(train2):   #for each asmfile
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()

def thirdprocess():
    op_file = open("third.txt", "w+")
    for asmfile in tqdm(train3):   #for each asmfile
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()

def fourthprocess():
    op_file = open("fourth.txt", "w+")
    for asmfile in tqdm(train4):   #for each asmfile
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()

def fifthprocess():
    op_file = open("fifth.txt", "w+")
    for asmfile in tqdm(train5):   #for each asmfile
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
```

In [6]:

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

```
    p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

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

```
100%|██████████| 2176/2176 [1:21:26<00:00,  2.25s/it]
100%|██████████| 2173/2173 [1:21:34<00:00,  2.25s/it]
100%|██████████| 2173/2173 [1:22:46<00:00,  2.29s/it]
100%|██████████| 2173/2173 [1:23:04<00:00,  2.29s/it]
100%|██████████| 2173/2173 [1:24:49<00:00,  2.34s/it]
```

In [4]:

```
import os
```

In [7]:

```
# combining all opcode file =s into one file
new_file = open("opcode_file.txt", "w+")
for i in ['first.txt','second.txt','third.txt','fourth.txt','fifth.txt']:
    with open(i, "r") as f:
        new_file.write(f.read())

new_file.close()
```

In [ ]:

```

```

Calculating opcode_bigrams

In [18]:

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

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

```
100%|██████████| 10868/10868 [06:35<00:00, 27.49it/s]
```

In [19]:

```
opcodebivect
```

Out[19]:

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

In [20]:

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

Calculating opcode_trigrams

In [24]:

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

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

```
100%|██████████| 10868/10868 [24:58<00:00,  7.25it/s]
```

In [25]:

```
opcodetrivect
```

Out[25]:

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

In [26]:

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

In [82]:

```
opcodetrivect=scipy.sparse.load_npz('opcodetrigram.npz')
```

In [83]:

```
opcodetrivect=normalize(opcodetrivect, axis = 0)
```

In [60]:

```
opcodebivect=scipy.sparse.load_npz('opcodebigram.npz')
```

In [61]:

```
opcodebivect=normalize(opcodebivect, axis = 0)
```

opcode_bigram important features

In [63]:

```
op_bi_indxes = imp_features(opcodebivect, asmopcodebigram, 300)
```

In [70]:

```
opcodebivect
```

Out[70]:

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

In [72]:

```
top_op_bi = np.zeros((10868, 0))
for i in op_bi_indxes:
    sliced = opcodebivect[:, i].todense()
    top_op_bi = np.hstack([top_op_bi, sliced])
```

In [73]:

```
op_bi_df = pd.DataFrame(top_op_bi, columns = np.take(asmopcodebigram, op_bi_indxes))
for col in op_bi_df.columns:
    if col not in np.take(asmopcodebigram, op_bi_indxes):
        op_bi_df.drop(col, axis = 1, inplace = True)
```

In [75]:

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

In [23]:

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

In [24]:

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

Out[24]:

	mov mov	push push	mov push	push mov	call call	push call	lea push	call mov	sub mov	call push	...	retf mov	inc call	jz dec	imul nop
0	0.000009	0.000810	0.000761	0.000542	0.000199	0.001121	0.000174	0.000066	0.000997	0.000541	...	0.0	0.000000	0.0	0.0
1	0.000963	0.000462	0.000932	0.001064	0.000000	0.000634	0.000443	0.000950	0.003252	0.000947	...	0.0	0.001263	0.0	0.0
2	0.000003	0.000729	0.000439	0.000165	0.000000	0.000815	0.000095	0.000009	0.000000	0.000000	...	0.0	0.000000	0.0	0.0
3	0.000000	0.000683	0.000014	0.000192	0.000796	0.001149	0.002214	0.000009	0.000000	0.001002	...	0.0	0.000000	0.0	0.0
4	0.000011	0.000755	0.000802	0.000604	0.000199	0.001100	0.000316	0.000066	0.000824	0.000487	...	0.0	0.000000	0.0	0.0

5 rows × 301 columns

In [ ]:

```
```

opcode_bigram important features

In [85]:

```
op_tri_indxes = imp_features(opcodetrivect, asmopcodetrigram, 500)
```

In [86]:

```
top_op_tri = np.zeros((10868, 0))
for i in op_tri_indxes:
    sliced = opcodetrivect[:, i].todense()
    top_op_tri = np.hstack([top_op_tri, sliced])
```

In [87]:

```
op_tri_df = pd.DataFrame(top_op_tri, columns = np.take(asmopcodetrigram, op_tri_indxes))
for col in op_tri_df.columns:
    if col not in np.take(asmopcodetrigram, op_tri_indxes):
        op_tri_df.drop(col, axis = 1, inplace = True)
```

In [88]:

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

In [25]:

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

In [26]:

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

Out[26]:

	push push push	mov mov mov	push mov push	mov push push	push push call	mov push mov	mov push call	mov sub mov	call call call	mov mov push	...	cmp mov jmp	inc xor pop	inc cmp mov	lea pus po
0	0.001005	0.000000	0.000476	0.000195	0.001086	0.001022	0.001374	0.001413	0.0	0.000123	...	0.000000	0.0	0.000000	0.0000
1	0.000279	0.000653	0.000641	0.000545	0.000268	0.001115	0.000937	0.003231	0.0	0.000932	...	0.003177	0.0	0.000389	0.0000
2	0.000987	0.000000	0.000434	0.000760	0.000844	0.000056	0.000656	0.000000	0.0	0.000025	...	0.000000	0.0	0.000000	0.0019
3	0.000652	0.000000	0.000000	0.000039	0.000871	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.0530
4	0.000950	0.000000	0.000662	0.000214	0.000992	0.001078	0.001467	0.001211	0.0	0.000123	...	0.000000	0.0	0.000000	0.0000

5 rows × 501 columns

Merging byte_features, byte_bigrams, asm_features, asm_image_features, opcode_bigrams and oopcode_trigrams

In [92]:

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

In [32]:

```
X = pd.concat([data_x,op_tri_df,op_bi_df], axis = 1, join = 'inner')
```

In [33]:

```
X = X.drop('ID', axis = 1)
X.head()
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...	shl cmp	retf mov	inc call	jz dec
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.0	0.0	0.000000	0.0
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.0	0.0	0.001263	0.0
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.0	0.0	0.000000	0.0
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.0	0.0	0.000000	0.0
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.0	0.0	0.000000	0.0

5 rows × 1906 columns

In [96]:

```
X.to_csv('X.csv')
```

In [34]:

```
Y = data_y
```

Splitting data

In [35]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(X, Y,stratify=Y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y
```

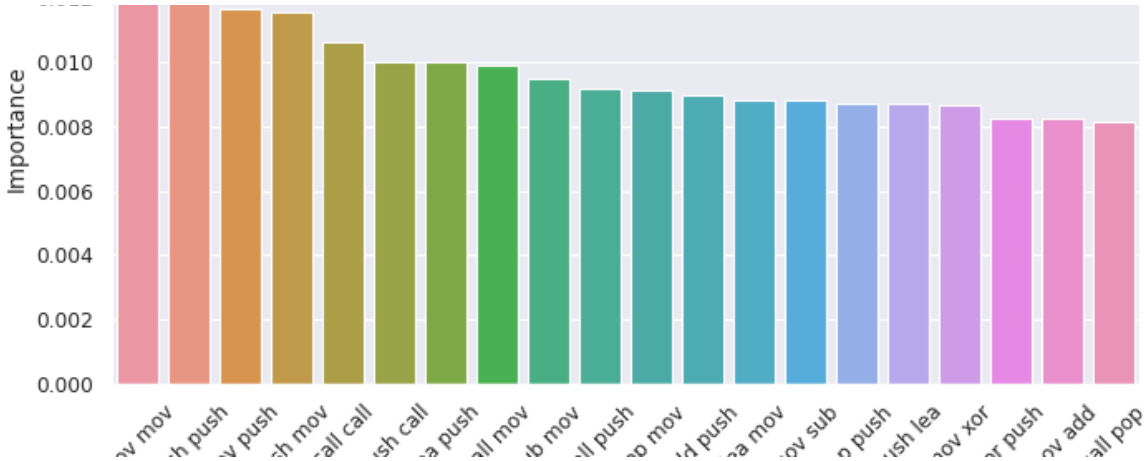
```
_train,test_size=0.20)
```

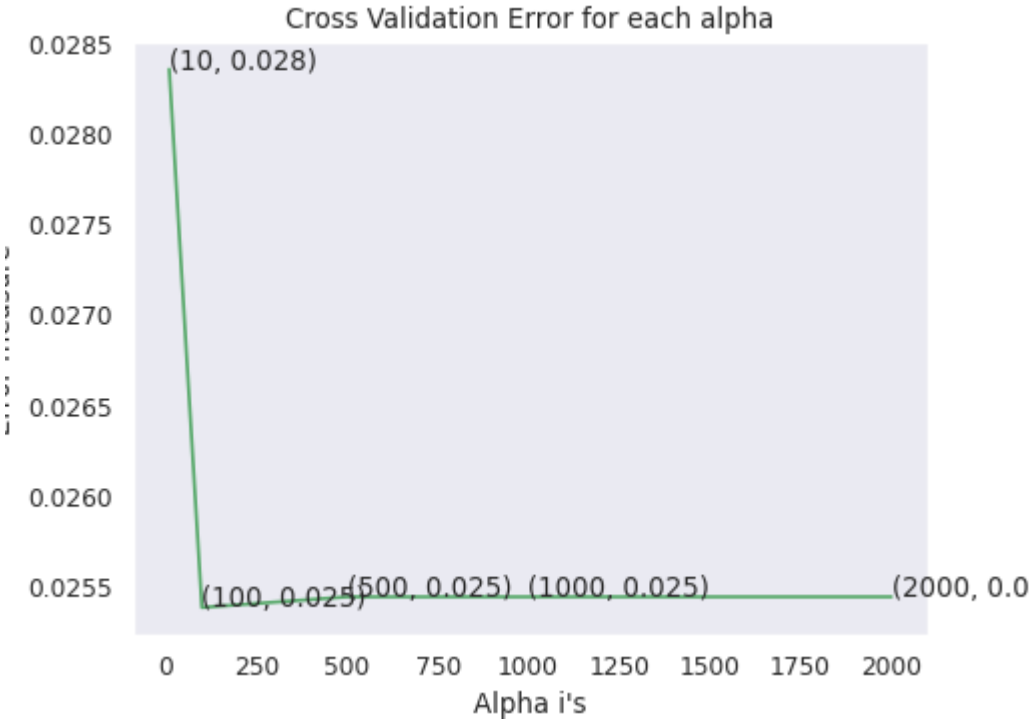
Xbgoost Classifier

In [101]:

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

# find more about XGBClassifier function here
http://xgboost.readthedocs.io/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, verbo
se=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 no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 0.028363323865031945
log_loss for c =  100 is 0.025382244755187144
log_loss for c =  500 is 0.025441122041622466
log_loss for c =  1000 is 0.025441177917746395
log_loss for c =  2000 is 0.025441264222881295
```






```
CPU times: user 18h 39min 29s, sys: 2min 21s, total: 18h 41min 51s
Wall time: 2h 24min 36s
```

In [102]:

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],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_lo
ss(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))
```

```
For values of best alpha =  100 The train log loss is: 0.01055921234891739
For values of best alpha =  100 The cross validation log loss is: 0.025382244755187144
For values of best alpha =  100 The test log loss is: 0.03151029819427085
CPU times: user 32min 34s, sys: 3.13 s, total: 32min 37s
Wall time: 32min 38s
```

Xgboost using random searchcv

In [36]:

```
x_cfl=XGBClassifier()

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 10.5min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed: 23.5min remaining: 13.6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 28.0min remaining:  8.5min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 30.2min remaining:  3.4min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 32.5min finished
```

Out[36]:

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

In [37]:

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

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 10, 'learning_rate': 0.05, 'colsample_bytree':
0.3}
```

In [38]:

```
%%time

x_cfl=XGBClassifier(n_estimators=200,max_depth=10,learning_rate=0.05,colsample_bytree=0.3,subsample
=1,nthread=-1)
```

```
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)
```

```
CPU times: user 23min 55s, sys: 928 ms, total: 23min 56s
Wall time: 23min 57s
```

Out[38]:

```
CalibratedClassifierCV(base_estimator=XGBClassifier(base_score=0.5,
                                                    booster='gbtree',
                                                    colsample_bylevel=1,
                                                    colsample_bynode=1,
                                                    colsample_bytree=0.3,
                                                    gamma=0, gpu_id=-1,
                                                    importance_type='gain',
                                                    interaction_constraints='',
                                                    learning_rate=0.05,
                                                    max_delta_step=0,
                                                    max_depth=10,
                                                    min_child_weight=1,
                                                    missing=nan,
                                                    monotone_constraints='()',
                                                    n_estimators=200, n_jobs=-1,
                                                    nthread=-1,
                                                    num_parallel_tree=1,
                                                    objective='multi:softprob',
                                                    random_state=0, reg_alpha=0,
                                                    reg_lambda=1,
                                                    scale_pos_weight=None,
                                                    subsample=1,
                                                    tree_method='exact',
                                                    validate_parameters=1,
                                                    verbosity=None))
```

In [39]:

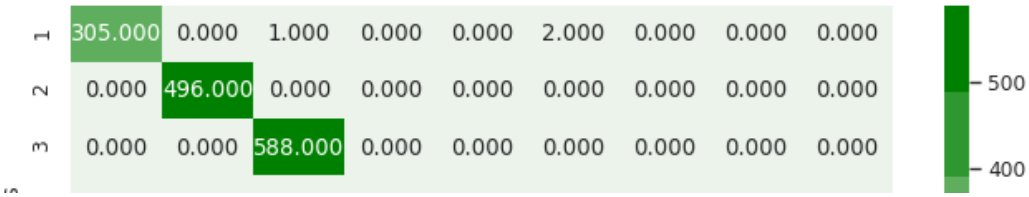
```
alpha=[100,200,500,1000,2000]
best_alpha = 1
predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(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))
```

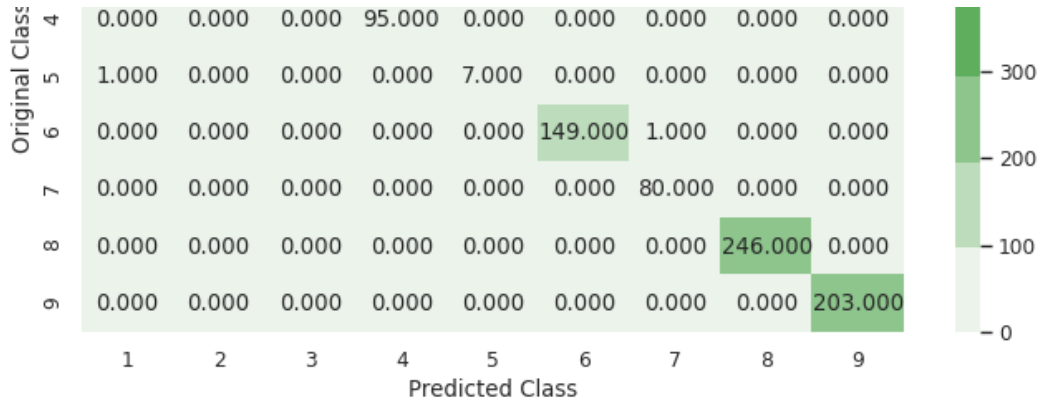
```
For values of best alpha =  200 The train log loss is: 0.009995845363399695
For values of best alpha =  200 The cross validation log loss is: 0.027039163498227315
For values of best alpha =  200 The test log loss is: 0.02146737308773401
```

In [40]:

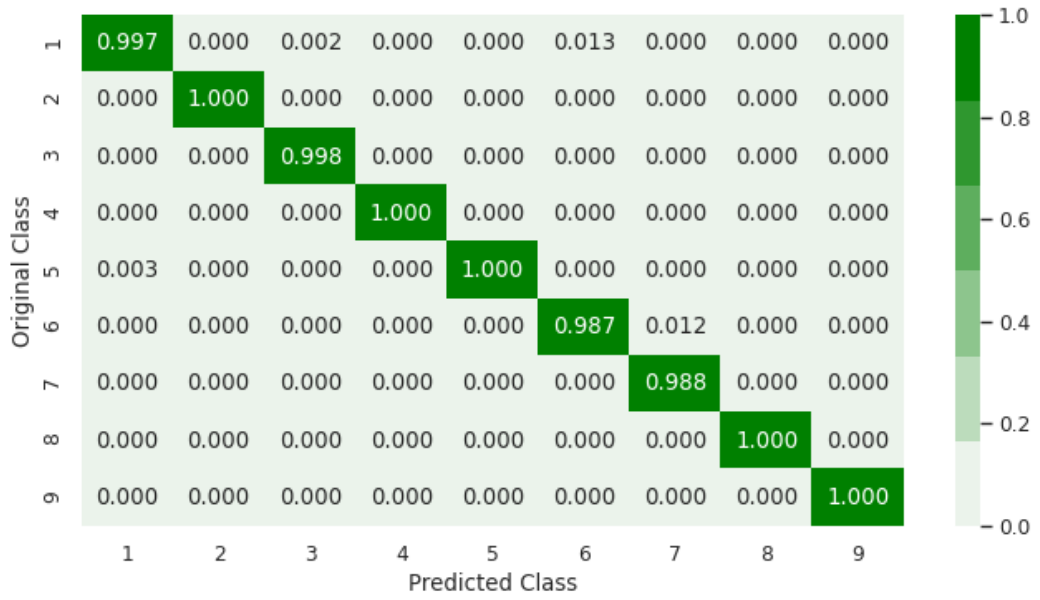
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

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



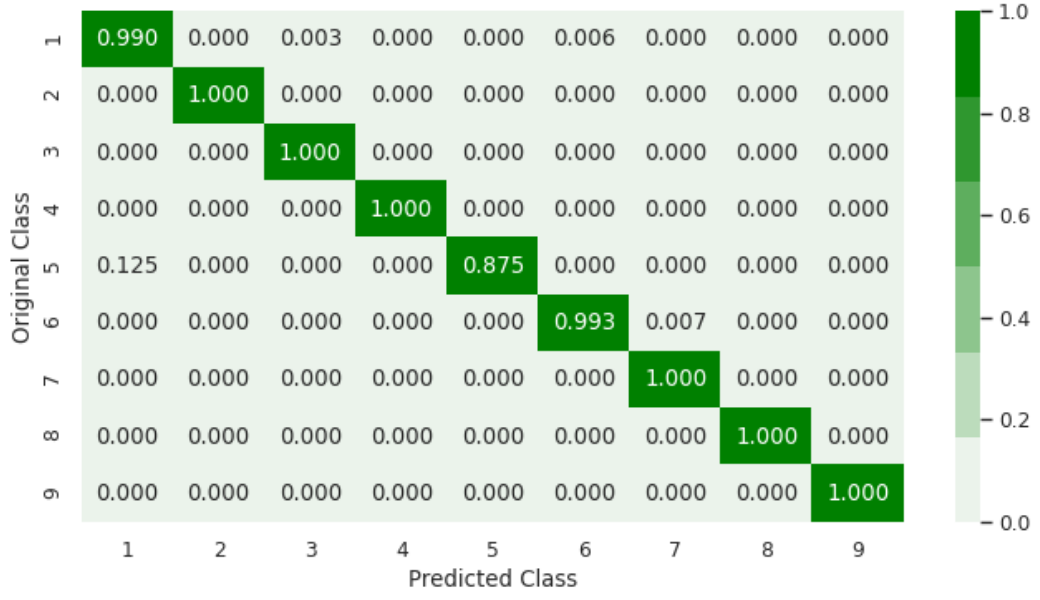


---------- Precision matrix ----------



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

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



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

Procedure

- **Firstly separated byte files and asm files.**
- **Then used unigram count features and performed EDA on both byte and asm files**
- **Combined both asm and byte unigram features and applied xgboost classifier**
- **Then obtained byte_bigrams and asm_image features**
- **Applied models on all the features.**
- **Then calculated opcode_bigrams, opcode_trigrams and combined all the features.**
- **Finally applied xgboost classifier on the final combined feature**

Summary

Features	Classifier	Train log-loss	Test log-loss
byte_feat + asm_feat	Xgboost	0.01	0.038
byte_feat + byte_bi + asm_feat + asm_img	Xgboost	0.01	0.04
byte_feat + byte_bi + asm_feat + asm_img + opcode_bi + opcode_tri	Xgboost	0.009	0.02