Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

.asm file

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

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

Metric(s):

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

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRILGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [2]:
```
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
np.random.seed(1)
```

In [ ]:
```
# !wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.106 Safari/537.36" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9" --header="Accept-Language: en-US,en;q=0.9,hi;q=0.8" --header="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kagglesdsdata/competitions/4117/46665/train.7z?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1593262927&Signature=oBWxiwo0FRhCYxqUSwh1OW0oMLKtEGAxispLZCAXD2DZMbqEtboM%2BdrZkunX9eT3pJ2AGiKBp4jWDX0lD89Ydn01mtOgFSbwAnZAGK%2F%2FoMDrw%2BqqirSUQ1MTlyEmLNpF%2FcrbnzeXT9H%2FWU%2BhwwADyZxaG0DB5p7%2FHoL2yKUlXEr4MQ%2FQohCVHIvMWhQF9SSLjwgGt4HXd10zLcV8Nf6h1We6%2B2cdkBVhoVIoZVggzFm9FiAVhIfWxjV7T4E7ALz3PJNangLb5MwvU6ghejY6%2FW51vv7%2B1wNhV8vnh%2FtieCG5%2Ban7ZJb9F6Mh8gxcINu3MhKJ8%2FFPeHVogYp%2FfIpbnw%3D%3D&response-content-disposition=attachment%3B+filename%3Dtrain.7z" -c -O 'train.7z'
```

In [ ]:
```
# !py7zr x train.7z
```

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 creat
e a folder with the same name
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

# if we have folder called 'train' (train folder contains both .asm files and
 .bytes files) we will rename it 'asmFiles'
# for every file that we have in our 'asmFiles' directory we check if it is en
ding 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:
        if (file.endswith("bytes")):
            shutil.move(source+'//'+file,destination_1)
        if (file.endswith("asm")):
            shutil.move(source+'//'+file,destination_2)
```

In [ ]:
```
Y=pd.read_csv("MMD/trainLabels.csv")
```

3.1. Distribution of malware classes in whole data set

In [ ]:
```
Y=pd.read_csv("MMD/trainLabels.csv")
total = len(Y)*1.
plt.figure(figsize=(10, 6))
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()/tota
l))
plt.show()
```

3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [ ]:
```
#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=356157170
0, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519
638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_s
tat.htm
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file na
me
    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())
```

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

In [ ]:
```
%matplotlib inline
#boxplot of byte files
plt.figure(figsize=(10, 6))
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

3.2.3 feature extraction from byte files

In [ ]:

```
#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
from tqdm.notebook import tqdm
files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in tqdm(files):
    if(file.endswith("bytes")):
        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,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,
2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,
48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,
62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,
7c,7d,7e,7f,80,81,82,83,84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,
96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,
b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,
ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,
e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,
fe,ff,??")
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 [ ]:
```
byte_features=pd.read_csv("MMD/result.csv")
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(5)
```

In [ ]:
```
data_size_byte.head(2)
```

In [ ]:
```
byte_features_with_size = byte_features.merge(data_size_byte, on='ID')
byte_features_with_size.to_csv("MMD1/result_with_size.csv")
byte_features_with_size.head(2)
```

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

In [ ]:
```
# result[result['ID']=='k3Gv04b1HUEMmrPCJAct']
# result.drop_duplicates(subset='ID',keep='first',inplace=True)
```

In [ ]:
```
result.shape
```

In [ ]:
```
data_y =result['Class']
data_y.head(2)
```

3.2.4 Multivariate Analysis

In [ ]:
```
#multivariate analysis on byte files
xtsne = TSNE(perplexity=50)
results = xtsne.fit_transform(result.drop(['ID','Class'],axis=1))
vis_x = results[ :, 0]
vis_y = results[ :, 1]
plt.figure(figsize=(10, 6))
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 [ ]:
```
#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.figure(figsize=(10, 6))
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

Train Test split

In [ ]:
```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'
], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same dis
tribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_tr
ain,test_size=0.20)
```

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

In [ ]:

```
# it returns a dict, keys as class labels and values as the number of data poi
nts in that class
train_class_distribution = y_train.value_counts().sort_index()
test_class_distribution = y_test.value_counts().sort_index()
cv_class_distribution = y_cv.value_counts().sort_index()
plt.figure(figsize=(10, 6))
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

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


print('-'*80)
plt.figure(figsize=(10, 6))
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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

print('-'*80)
plt.figure(figsize=(10, 6))
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
```

```
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/y_cv.shape[0]*100), 3),
'%)')
```

In [ ]:
```
plt.figure(figsize=(10, 6))
data_y.value_counts().plot(kind='bar');
```

In [3]:

```
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 a
re predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at 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=(10,6))
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytick
labels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*50, "Precision matrix", "-"*50)
    plt.figure(figsize=(10,6))
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytick
labels=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=(10,6))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytick
labels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

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

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

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


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

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

4.1.2. K Nearest Neighbour Classification

In [ ]:

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



for i in range(len(cv_log_error_array)):
    print(" Log Loss for K = ",alpha[i],' is ',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

print("Best K  is : ",best_alpha)

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 K ")
plt.xlabel("K values ")
plt.ylabel("Error Measure")
plt.show()


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


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

In [ ]:

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR
.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.class
es_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.c
lasses_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

In [ ]:

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

In [ ]:

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

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

In [ ]:
```
random_cfl1.best_params_
```

In [ ]:
```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, max_depth=3,n_jobs=-1,)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In parallel we c an use all the cores that are present in our computer.

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

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

4.2.1 Feature extraction from asm files

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

In [ ]:

```
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='first'
folder_2 ='second'
folder_3 ='third'
folder_4 ='fourth'
folder_5 ='fifth'
folder_6 = 'output'
for i in [folder_1,folder_2,folder_3,folder_4,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:','.rdat
a:','.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', 'su
b', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ro
r', 'rol', 'jnb','jz','rtn','lea','movzx']
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodesfcount=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 fl
i:
            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' segm
ents
```

```
                        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:','.rdat
a:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
b', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ro
r', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as f
li:
            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:','.rdat
a:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
b', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ro
r', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('thrid/'+f,encoding='cp1252',errors ='replace') as fl
i:
            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:','.rdat
a:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
b', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ro
r', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as f
li:
            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:','.rdat
a:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
b', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ro
r', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
    files = os.listdir('fifth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fifth/'+f,encoding='cp1252',errors ='replace') as fl
i:
            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 [ ]:

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

In [ ]:
```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=356157170
0, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519
638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_s
tat.htm
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file na
me
    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())
```

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

In [ ]:
```
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

4.2.2 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()
```

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

In [ ]:
```
#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.figure(figsize=(10, 6))
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 [ ]:
```
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.C
ODE','size'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.figure(figsize=(10, 6))
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

- **We have taken only 52 features from asm files (after reading through many blogs and research papers)**
- **The univariate analysis was done only on few important features.**
- **Take-aways**
  - **1. Class 3 can be easily separated because of the frequency of segments,opcodes and keywords being less**
  - **2. Each feature has its unique importance in separating the Class labels.**

4.3 Train and test split

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

In [ ]:
```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_
y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y
_train_asm,stratify=y_train_asm,test_size=0.20)
```

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

4.4. Machine Learning models on features of .asm files

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

4.4.2 Logistic Regression

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

4.4.3 Random Forest Classifier

In [ ]:

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

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


best_alpha = np.argmin(cv_log_error_array)

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

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_
jobs=-1)
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_
clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.cl
asses_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_cl
f.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

In [ ]:

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

In [ ]:
```
x_cfl=XGBClassifier()

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

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

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

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

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

4.5.1. Merging both asm and byte file features

hh

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

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

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

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

4.5.2. Multivariate Analysis on final fearures

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

4.5.3. Train and Test split

In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, resu
lt_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_trai
n, y_train,stratify=y_train,test_size=0.20)
```

In [ ]:

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

In [ ]:
```
x_cfl=XGBClassifier()

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

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

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

In [ ]:
```
best_alpha=3
```

In [ ]:
```
alpha=[10,50,100,500,1000,2000]
predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

In [ ]:
```

```

Assignment

Bi-Gram of Byte File

In [ ]:
```
unigram_vocab =['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '0
a', '0b', '0c', '0d', '0e', '0f', '10', '11', '12', '13', '14', '15', '16', '1
7', '18', '19', '1a', '1b', '1c', '1d', '1e', '1f', '20', '21', '22', '23', '2
4', '25', '26', '27', '28', '29', '2a', '2b', '2c', '2d', '2e', '2f', '30', '3
1', '32', '33', '34', '35', '36', '37', '38', '39', '3a', '3b', '3c', '3d', '3
e', '3f', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '4a', '4
b', '4c', '4d', '4e', '4f', '50', '51', '52', '53', '54', '55', '56', '57', '5
8', '59', '5a', '5b', '5c', '5d', '5e', '5f', '60', '61', '62', '63', '64', '6
5', '66', '67', '68', '69', '6a', '6b', '6c', '6d', '6e', '6f', '70', '71', '7
2', '73', '74', '75', '76', '77', '78', '79', '7a', '7b', '7c', '7d', '7e', '7
f', '80', '81', '82', '83', '84', '85', '86', '87', '88', '89', '8a', '8b', '8
c', '8d', '8e', '8f', '90', '91', '92', '93', '94', '95', '96', '97', '98', '9
9', '9a', '9b', '9c', '9d', '9e', '9f', 'a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a
6', 'a7', 'a8', 'a9', 'aa', 'ab', 'ac', 'ad', 'ae', 'af', 'b0', 'b1', 'b2', 'b
3', 'b4', 'b5', 'b6', 'b7', 'b8', 'b9', 'ba', 'bb', 'bc', 'bd', 'be', 'bf', 'c
0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9', 'ca', 'cb', 'cc', 'c
d', 'ce', 'cf', 'd0', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'd
a', 'db', 'dc', 'dd', 'de', 'df', 'e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e
7', 'e8', 'e9', 'ea', 'eb', 'ec', 'ed', 'ee', 'ef', 'f0', 'f1', 'f2', 'f3', 'f
4', 'f5', 'f6', 'f7', 'f8', 'f9', 'fa', 'fb', 'fc', 'fd', 'fe', 'ff', '??']
print(len(unigram_vocab))
```

There is 257 unique words in byte file so in bi-gram there will be 257*257=66049 combinations

In [ ]:
```
bi_gram_vocab=[]
for i in range(len(unigram_vocab)):
    for j in range(len(unigram_vocab)):
        bi_gram_vocab.append(unigram_vocab[i]+' '+unigram_vocab[j])

print(len(bi_gram_vocab))
print(bi_gram_vocab[:5])
```

In [ ]:
```
import multiprocessing

multiprocessing.cpu_count()
```

In [ ]:
```
os.mkdir('')
```

In [ ]:
```
os.listdir('byteFiles')
```

In [ ]:

```
%%time
# importing the multiprocessing module
import multiprocessing
from tqdm.notebook import tqdm
def part1(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[0:1000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_1_1000.npz', bigram_vect)

def part2(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[1000:2000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_2_1000.npz', bigram_vect)



def part3(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[2000:3000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_3_1000.npz', bigram_vect)

def part4(bi_gram_vocab):
    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
```

```
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[3000:4000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_4_1000.npz', bigram_vect)

def part5(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[4000:5000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_5_1000.npz', bigram_vect)

def part6(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[5000:6000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_6_1000.npz', bigram_vect)



def part7(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[6000:7000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_7_1000.npz', bigram_vect)

def part8(bi_gram_vocab):
```

```
    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[7000:8000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_8_1000.npz', bigram_vect)
if __name__ == "__main__":
    # creating processes
    p1 = multiprocessing.Process(target=part1, args=(bi_gram_vocab, ))
    p2 = multiprocessing.Process(target=part2, args=(bi_gram_vocab, ))
    p3 = multiprocessing.Process(target=part3, args=(bi_gram_vocab, ))
    p4 = multiprocessing.Process(target=part4, args=(bi_gram_vocab, ))
    p5 = multiprocessing.Process(target=part5, args=(bi_gram_vocab, ))
    p6 = multiprocessing.Process(target=part6, args=(bi_gram_vocab, ))
    p7 = multiprocessing.Process(target=part7, args=(bi_gram_vocab, ))
    p8 = multiprocessing.Process(target=part8, args=(bi_gram_vocab, ))

    # starting process 1
    p1.start()
    # starting process 2
    p2.start()

    # starting process 3
    p3.start()
    # starting process 4
    p4.start()

    # starting process 5
    p5.start()
    # starting process 6
    p6.start()

    # starting process 7
    p7.start()
    # starting process 8
    p8.start()



    # wait until process 1 is finished
    p1.join()
    # wait until process 2 is finished
    p2.join()
      # wait until process 3 is finished
    p3.join()
    # wait until process 4 is finished
    p4.join()

    # wait until process 5 is finished
    p5.join()
    # wait until process 6 is finished
```

```
    p6.join()
      # wait until process 7 is finished
    p7.join()
    # wait until process 8 is finished
    p8.join()

    # both processes finished
    print("Done!")
```

In [ ]:

```
%%time
# importing the multiprocessing module
import multiprocessing
from tqdm.notebook import tqdm
def part9(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[8000:9000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_9_1000.npz', bigram_vect)

def part10(bi_gram_vocab):

    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((1000, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[9000:10000])):
        f = open('byteFiles/'+file, "r")
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_10_1000.npz', bigram_vect)



def part11(bi_gram_vocab):
    from sklearn.feature_extraction.text import CountVectorizer
    from scipy.sparse import csr_matrix
    import scipy.sparse
    file_name_list=[]
    bigram_vect = csr_matrix((868, 66049))
    bi_gram_vector = CountVectorizer(lowercase=False,ngram_range=(2,2),vocabul
ary=bi_gram_vocab)
    for i,file in tqdm(enumerate(os.listdir('byteFiles')[10000:10868])):
        f = open('byteFiles/'+file, "r")
#         file_name=file.split('.')[0] #taking file name
#         file_name_list.append(file_name)
        bigram_vect[i]= bi_gram_vector.fit_transform([f.read().replace("\n","
 ").lower()]) #after getting vectorize
    scipy.sparse.save_npz('bigram/bytebigram_11_868.npz', bigram_vect)
if __name__ == "__main__":
    # creating processes
    p9 = multiprocessing.Process(target=part9, args=(bi_gram_vocab, ))
    p10 = multiprocessing.Process(target=part10, args=(bi_gram_vocab, ))
    p11 = multiprocessing.Process(target=part11, args=(bi_gram_vocab, ))
```

```
    # starting process 9
    p9.start()
    # starting process 10
    p10.start()

    # starting process 11
    p11.start()


    # wait until process 9 is finished
    p9.join()
    # wait until process 10 is finished
    p10.join()
      # wait until process 11 is finished
    p11.join()

    # both processes finished
    print("Done!")
```

In [ ]:
```
import scipy.sparse
from scipy.sparse import coo_matrix, vstack
byte_bigram_vect1       = scipy.sparse.load_npz('bigram/bytebigram_1_1000.npz'
)
byte_bigram_vect2       = scipy.sparse.load_npz('bigram/bytebigram_2_1000.npz'
)
byte_bigram_vect3       = scipy.sparse.load_npz('bigram/bytebigram_3_1000.npz'
)
byte_bigram_vect4       = scipy.sparse.load_npz('bigram/bytebigram_4_1000.npz'
)
byte_bigram_vect5       = scipy.sparse.load_npz('bigram/bytebigram_5_1000.npz'
)
byte_bigram_vect6       = scipy.sparse.load_npz('bigram/bytebigram_6_1000.npz'
)
byte_bigram_vect7       = scipy.sparse.load_npz('bigram/bytebigram_7_1000.npz'
)
byte_bigram_vect8       = scipy.sparse.load_npz('bigram/bytebigram_8_1000.npz'
)
byte_bigram_vect9       = scipy.sparse.load_npz('bigram/bytebigram_9_1000.npz'
)
byte_bigram_vect10      = scipy.sparse.load_npz('bigram/bytebigram_10_1000.np
z')
byte_bigram_vect11      = scipy.sparse.load_npz('bigram/bytebigram_11_868.npz'
)

byte_vector_all=vstack([byte_bigram_vect1,byte_bigram_vect2,byte_bigram_vect3,
byte_bigram_vect4,byte_bigram_vect5,byte_bigram_vect6,byte_bigram_vect7,byte_b
igram_vect8,byte_bigram_vect9,byte_bigram_vect10,byte_bigram_vect11])
print(byte_vector_all.shape)
```

In [ ]:
```
import scipy.sparse
scipy.sparse.save_npz('bigram/all_bytebigram.npz', byte_vector_all) ## saving
 all bi-gram variable
```

In [ ]:
```
os.listdir('byteFiles')[:5]
```

In [ ]:
```
##based on File sequence getting class labels
class_value=pd.read_csv("MMD/trainLabels.csv")
file_list =[ i.split('.')[0] for i in os.listdir('byteFiles')] # getting text
 file as ID
file_list = pd.DataFrame(file_list, columns=['Id']) #converting list in to Pan
das Dataframe
y_value = file_list.merge(class_value, on='Id') #Based on Id getting Class val
ue
y_value.to_csv('MMD1/y_value.csv', index=False)
y_value.head()
```

In [ ]:
```
%%time
#normalizing bi-gram
from sklearn.preprocessing import normalize
from scipy.sparse import csr_matrix
import scipy.sparse
byte_vector_normalize = normalize(scipy.sparse.load_npz('bigram/all_bytebigra
m.npz'), axis = 0)
```

In [ ]:
```
#saving normalized bi-gram
import scipy.sparse
scipy.sparse.save_npz('bigram/byte_vector_normalize.npz', byte_vector_normaliz
e)
```

In [ ]:
```
%%time
import scipy.sparse
byte_vector_normalize = scipy.sparse.load_npz('bigram/byte_vector_normalize.np
z')
```

In [ ]:
```
byte_vector_normalize.shape, y_value.shape
```

In [ ]:
```
%%time
data_y = y_value['Class']
# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(byte_vector_normalize, dat
a_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same dis
tribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_tr
ain,test_size=0.20)
```

In [ ]:
```
#After spliting into x and y removing bi-gram variable about 6 GB Ram will cle
ar
import gc
import resource
del byte_vector_normalize
gc.collect()
```

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_
jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)
filename = '002_Random_forest_bi_gram_model_r_cfl.sav'
pickle.dump(sig_clf, open('Models'+filename, 'wb'))

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

In [ ]:
```
filename = '002_Random_forest_bi_gram_model_r_cfl.sav'
r_cfl = pickle.load(open("Models/"+filename, 'rb'))
```

I have tried XGboost with 66049 feature in any condition it does not work even less then 100 estimator/for loop as suggested Team.Every time it is getting out of memory issue.After lot's of search I found to that we can work on important feature which os less dimentions for train model

In [ ]:
```
type(byte_vector_normalize)
```

In [ ]:
```
feat_importances = pd.Series(r_cfl.feature_importances_,)
feat_importances.nlargest(5000).plot(kind='barh')
plt.title("Best 5000 features")
plt.xlabel("Features percentage")
plt.ylabel("feature name")
plt.show()
```

In [ ]:
```
feat_importances = pd.Series(r_cfl.feature_importances_,)
feat_importances.nlargest(1000).plot(kind='barh')
plt.title("Best 1000 features")
plt.xlabel("Features percentage")
plt.ylabel("feature name")
plt.show()
```

In [ ]:
```
imp_feature_index =np.argsort(r_cfl.feature_importances_)[::-1]
top_1000_bigram_feature =byte_vector_normalize[:,imp_feature_index[0:1000]]
```

In [ ]:
```
byte_bi_df = pd.DataFrame.sparse.from_spmatrix(top_1000_bigram_feature, column
s = column_bi_gram)
byte_bi_df['Id']=y_value['Id']
byte_bi_df.head()
```

In [ ]:
```
import scipy.sparse
scipy.sparse.save_npz('bigram/top_1000_bigram_feature.npz', top_1000_bigram_fe
ature)
```

In [ ]:
```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','train log loss','Test log loss']
ptable.add_row(["random","Byte files","2.49","2.49"])
ptable.add_row(["KNN ","unigram Byte files","0.114","0.21"])
ptable.add_row(["Logistic Regression","Unigram Byte files"," 0.875","0.88"])
ptable.add_row(["RandomForest","Unigram Byte files"," 0.0261","0.095"])
ptable.add_row(["Xgboost "," Unigram Byte files"," 0.021","0.071"])
ptable.add_row(["\n","\n","\n",""])

ptable.add_row(["KNN "," Unigram ASM files"," 0.023","0.085"])
ptable.add_row(["Logistic Regression","Unigram ASM files"," 1.00","0.992"])
ptable.add_row(["RandomForest","Unigram ASM files"," 0.0141","0.0424"])
ptable.add_row(["Xgboost "," Unigram ASM files"," 0.013","0.033"])

ptable.add_row(["\n","\n","\n",""])

ptable.add_row(["RandomForest","Unigram ASM+Byte files"," 0.0167","0.0424"])
ptable.add_row(["Xgboost "," Unigram ASM+Byte files"," 0.0128","0.027"])

ptable.add_row(["\n","\n","\n",""])
ptable.add_row(["RandomForest","bi-gram Byte files 66049 features"," 0.0210",
"0.066"])
ptable.add_row(["\n","\n","\n",""])
ptable.add_row(["Xgboost "," bi-gram Byte files 1000 features"," 0.0149","0.04
4"])
ptable.add_row(["Xgboost "," unigram ASM +bi-gram Byte files 1000 features","
 0.0150","0.058"])
print(ptable)
```

In [ ]:
```
def imp_features(data,result_y, 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[:keep])
    imp_feature_name = np.take(features, imp_feature_indx[:keep])
    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')
    plt.show()
    return imp_feature_indx[:keep]
```

Image Feature

In [ ]:
```
def read_image(filename):
    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,(len(a)//width,width))
    g = np.uint8(g)
    return list(g.flatten()[:1000])
```

In [ ]:
```
from IPython.display import Image
Image(filename="asm_image/"+'0cfIE39ihRNo2rkZOw5H.png')
```

In [ ]:

```
import array
import imageio
def extract_asm_image_features1():
    tfiles=train1
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/1-image-features-asm.csv'
    print('feature file:', feature_file)

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in ra
nge(1000)]
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image(ext_drive + fname)
            outrows.append([file_id] + image_data)
            # Print progress
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(outrows)
              outrows = []

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


def extract_asm_image_features2():
    tfiles=train2
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/2-image-features-asm.csv'
    print('feature file:', feature_file)

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

            # Print progress
```

```
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

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


def extract_asm_image_features3():
    tfiles=train3
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/3-image-features-asm.csv'
    print('feature file:', feature_file)

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

            # Print progress
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

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


def extract_asm_image_features4():
    tfiles=train4
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/4-image-features-asm.csv'
    print('feature file:', feature_file)

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
```

```
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in ra
nge(1000)]
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image(ext_drive + fname)
            outrows.append([file_id] + image_data)

            # Print progress
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(outrows)
              outrows = []

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


def extract_asm_image_features5():
    tfiles=train5
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/5-image-features-asm.csv'
    print('feature file:', feature_file)

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

            # Print progress
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(outrows)
              outrows = []

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


def extract_asm_image_features6():
    tfiles=train6
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)
```

```
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/6-image-features-asm.csv'
    print('feature file:', feature_file)

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

            # Print progress
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(outrows)
              outrows = []

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


def extract_asm_image_features7():
    tfiles=train7
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/7-image-features-asm.csv'
    print('feature file:', feature_file)

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

            # Print progress
            if (idx+1) % 10 == 0:
#               print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(outrows)
              outrows = []

        # Write remaining files
```

```
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []


def extract_asm_image_features8():
    tfiles=train8
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/8-image-features-asm.csv'
    print('feature file:', feature_file)

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

            # Print progress
            if (idx+1) % 10 == 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 [ ]:

```
ext_drive = 'asmFiles/'
tfiles = os.listdir(ext_drive)
part = len(tfiles)/8
part1=round(part)
train1 = tfiles[:part1]
train2 = tfiles[part1:(2*part1)]
train3 = tfiles[(2*part1):(3*part1)]
train4 = tfiles[(3*part1):(4*part1)]
train5 = tfiles[(4*part1):(5*part1)]
train6 = tfiles[(5*part1):(6*part1)]
train7 = tfiles[(6*part1):(7*part1)]
train8 = tfiles[(7*part1):]

if __name__ == "__main__":
    # creating processes
    p1 = multiprocessing.Process(target=extract_asm_image_features1, args=())
    p2 = multiprocessing.Process(target=extract_asm_image_features2, args=())
    p3 = multiprocessing.Process(target=extract_asm_image_features3, args=())
    p4 = multiprocessing.Process(target=extract_asm_image_features4, args=())
    p5 = multiprocessing.Process(target=extract_asm_image_features5, args=())
    p6 = multiprocessing.Process(target=extract_asm_image_features6, args=())
    p7 = multiprocessing.Process(target=extract_asm_image_features7, args=())
    p8 = multiprocessing.Process(target=extract_asm_image_features8, args=())

    # starting process 1
    p1.start()
    # starting process 2
    p2.start()

    # starting process 3
    p3.start()
    # starting process 4
    p4.start()

    # starting process 5
    p5.start()
    # starting process 6
    p6.start()

    # starting process 7
    p7.start()
    # starting process 8
    p8.start()



    # wait until process 1 is finished
    p1.join()
    # wait until process 2 is finished
    p2.join()
      # wait until process 3 is finished
    p3.join()
    # wait until process 4 is finished
    p4.join()

    # wait until process 5 is finished
```

```
    p5.join()
    # wait until process 6 is finished
    p6.join()
      # wait until process 7 is finished
    p7.join()
    # wait until process 8 is finished
    p8.join()

    # both processes finished
    print("Done!")
```

In [4]:
```
df1=pd.read_csv("data/1-image-features-asm.csv")
df2=pd.read_csv("data/2-image-features-asm.csv")
df3=pd.read_csv("data/3-image-features-asm.csv")
df4=pd.read_csv("data/4-image-features-asm.csv")
df5=pd.read_csv("data/5-image-features-asm.csv")
df6=pd.read_csv("data/6-image-features-asm.csv")
df7=pd.read_csv("data/7-image-features-asm.csv")
df8=pd.read_csv("data/8-image-features-asm.csv")

df_image= pd.concat([df1,df2,df3,df4,df5,df6,df7,df8], axis=0)
df_image.to_csv("data/final-image-features-asm.csv")
df_image =pd.read_csv("data/final-image-features-asm.csv")


df_image.shape
```

Out[4]:
```
(10868, 1002)
```

In [ ]:
```
df_image.head()
```

Get ASM Feature using API

In [ ]:

```
keywords = ['Virtual','Offset','loc','Import','Imports','var','Forwarder','UIN
T','LONG','BOOL','WORD','BYTES','large','short','dd','db','dw','XREF','ptr','D
ATA','FUNCTION','extrn','byte','word','dword','char','DWORD','stdcall','arg',
'locret','asc','align','WinMain','unk','cookie','off','nullsub','DllEntryPoin
t','System32','dll','CHUNK','BASS','HMENU','DLL','LPWSTR','void','HRESULT','HD
C','LRESULT','HANDLE','HWND','LPSTR','int','HLOCAL','FARPROC','ATOM','HMODULE'
,'WPARAM','HGLOBAL','entry','rva','COLLAPSED','config','exe','Software','Curre
ntVersion','__imp_','INT_PTR','UINT_PTR','---Seperator','PCCTL_CONTEXT','__IMP
ORT_','INTERNET_STATUS_CALLBACK','.rdata:','.data:','.text:','case','installdi
r','market','microsoft','policies','proc','scrollwindow','search','trap','visu
alc','___security_cookie','assume','callvirtualalloc','exportedentry','hardwar
e','hkey_current_user','hkey_local_machine','sp-analysisfailed','unableto']
known_sections = ['.text', '.data', '.bss', '.rdata', '.edata', '.idata', '.rs
rc', '.tls', '.reloc']
registers = ['edx','esi','es','fs','ds','ss','gs','cs','ah','al',
                 'ax','bh','bl','bx','ch','cl','cx','dh','dl','dx',
                 'eax','ebp','ebx','ecx','edi','esp']

opcodes = ['add','al','bt','call','cdq','cld','cli','cmc','cmp','const','cwd',
'daa','db'
                ,'dd','dec','dw','endp','ends','faddp','fchs','fdiv','fdivp',
'fdivr','fild'
                ,'fistp','fld','fstcw','fstcwimul','fstp','fword','fxch','imu
l','in','inc'
                ,'ins','int','jb','je','jg','jge','jl','jmp','jnb','jno','jnz'
,'jo','jz'
                ,'lea','loope','mov','movzx','mul','near','neg','not','or','ou
t','outs'
                ,'pop','popf','proc','push','pushf','rcl','rcr','rdtsc','rep',
'ret','retn'
                ,'rol','ror','sal','sar','sbb','scas','setb','setle','setnle',
'setnz'
                ,'setz','shl','shld','shr','sidt','stc','std','sti','stos','su
b','test'
                ,'wait','xchg','xor']

#https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list
```

In [ ]:

```
def count_asm_symbols(asm_code):
    symbols = [0]*7
    for row in asm_code:
        if '*' in row:
            symbols[0] += 1
        if '-' in row:
            symbols[1] += 1
        if '+' in row:
            symbols[2] += 1
        if '[' in row:
            symbols[3] += 1
        if ']' in row:
            symbols[4] += 1
        if '@' in row:
            symbols[5] += 1
        if '?' in row:
            symbols[6] += 1

    return symbols


def count_asm_registers(asm_code):
    registers_values = [0]*len(registers)
    for row in asm_code:
        row.decode
        parts = str(row).replace(',',' ').replace('+',' ').replace('*',' ').re
place('[',' ').replace(']',' ') \
                    .replace('-',' ').split()
        for register in registers:
            registers_values[registers.index(register)] += parts.count(registe
r)
    return registers_values


def count_asm_opcodes(asm_code):
    opcodes_values = [0]*len(opcodes)
    for row in asm_code:
        parts = row.split()

        for opcode in opcodes:
            if opcode in parts:
                opcodes_values[opcodes.index(opcode)] += 1
                break
    return opcodes_values


def count_asm_APIs(asm_code, apis):
    apis_values = [0]*len(apis)
    for row in asm_code:
        for i in range(len(apis)):
            if apis[i] in str(row):
                apis_values[i] += 1
                break
    return apis_values
```

```
def count_asm_misc(asm_code):
    keywords_values = [0]*len(keywords)
    for row in asm_code:
        for i in range(len(keywords)):
            if keywords[i] in str(row):
                keywords_values[i] += 1
                break
    return keywords_values
```

In [ ]:

```
def extract_asm_feature1():
    tfiles=train1
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/'+'1-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []

def extract_asm_feature2():
    tfiles=train2
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' +'2-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')
```

```
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []

def extract_asm_feature3():
    tfiles=train3
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' +'3-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)
```

```
        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []
###
def extract_asm_feature4():
    tfiles=train4
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/'+'4-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
```

```
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []

def extract_asm_feature5():
    tfiles=train5
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' +'5-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []

def extract_asm_feature6():
    tfiles=train6
```

```
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' +'6-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []
###
def extract_asm_feature7():
    tfiles=train7
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' +'7-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)
```

```
    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

        # Writing remaining files
        if len(feature_counts) > 0:
            fw.writerows(feature_counts)
            feature_counts = []

def extract_asm_feature8():
    tfiles=train8
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' +'8-malware-features-asm.csv' # Windows API, symbol
s, registers, opcodes, etc...
    print('feature file:', feature_file)

    fapi = open("data/APIs.txt")
    defined_apis = fapi.readlines()
    defined_apis = defined_apis[0].split(',')

    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_counts = []
    with open(feature_file, 'w') as f:
        # write the csv header
        fw = writer(f)
        colnames = ['filename'] + registers + opcodes + defined_apis + keyword
s
        fw.writerow(colnames)

        for idx, fname in enumerate(asm_files):
            fasm = open(ext_drive + fname, 'rb')
            content = fasm.readlines()
```

```
            reg_vals = count_asm_registers(content)
            opc_vals = count_asm_opcodes(content)
            api_vals = count_asm_APIs(content, defined_apis)
#             sec_vals = count_asm_sections(content)
            mis_vals = count_asm_misc(content)
            count_vals = reg_vals + opc_vals + api_vals + mis_vals
            feature_counts.append([fname[:fname.find('.asm')]] + count_vals)

            # Writing rows after every 10 files processed
            if (idx+1) % 10 == 0:
              print(pid, idx + 1, 'of', ftot, 'files processed.')
              fw.writerows(feature_counts)
              feature_counts = []

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

In [ ]:

```
ext_drive = 'asmFiles/'
tfiles = os.listdir(ext_drive)
part = len(tfiles)/8
part1=round(part)
train1 = tfiles[:part1]
train2 = tfiles[part1:(2*part1)]
train3 = tfiles[(2*part1):(3*part1)]
train4 = tfiles[(3*part1):(4*part1)]
train5 = tfiles[(4*part1):(5*part1)]
train6 = tfiles[(5*part1):(6*part1)]
train7 = tfiles[(6*part1):(7*part1)]
train8 = tfiles[(7*part1):]

if __name__ == "__main__":
    # creating processes
    p1 = multiprocessing.Process(target=extract_asm_image_features1, args=())
    p2 = multiprocessing.Process(target=extract_asm_image_features2, args=())
    p3 = multiprocessing.Process(target=extract_asm_image_features3, args=())
    p4 = multiprocessing.Process(target=extract_asm_image_features4, args=())
    p5 = multiprocessing.Process(target=extract_asm_image_features5, args=())
    p6 = multiprocessing.Process(target=extract_asm_image_features6, args=())
    p7 = multiprocessing.Process(target=extract_asm_image_features7, args=())
    p8 = multiprocessing.Process(target=extract_asm_image_features8, args=())

    # starting process 1
    p1.start()
    # starting process 2
    p2.start()

    # starting process 3
    p3.start()
    # starting process 4
    p4.start()

    # starting process 5
    p5.start()
    # starting process 6
    p6.start()

    # starting process 7
    p7.start()
    # starting process 8
    p8.start()



    # wait until process 1 is finished
    p1.join()
    # wait until process 2 is finished
    p2.join()
      # wait until process 3 is finished
    p3.join()
    # wait until process 4 is finished
    p4.join()

    # wait until process 5 is finished
```

```
    p5.join()
    # wait until process 6 is finished
    p6.join()
      # wait until process 7 is finished
    p7.join()
    # wait until process 8 is finished
    p8.join()

    # both processes finished
    print("Done!")
```

In [5]:
```
df1=pd.read_csv("data/1-malware-features-asm.csv")
df2=pd.read_csv("data/2-malware-features-asm.csv")
df3=pd.read_csv("data/3-malware-features-asm.csv")
df4=pd.read_csv("data/4-malware-features-asm.csv")
df5=pd.read_csv("data/5-malware-features-asm.csv")
df6=pd.read_csv("data/6-malware-features-asm.csv")
df7=pd.read_csv("data/7-malware-features-asm.csv")
df8=pd.read_csv("data/8-malware-features-asm.csv")

df_asm_feature= pd.concat([df1,df2,df3,df4,df5,df6,df7,df8], axis=0)
df_asm_feature.to_csv("data/final-malware-features-asm.csv",index=False)
df_asm_feature =pd.read_csv("data/final-malware-features-asm.csv")
df_asm_feature.shape
```

Out[5]:
```
(10868, 1007)
```

In [ ]:
```
df_asm_feature.head()
```

In [ ]:
```
import scipy.sparse
byte_vector= scipy.sparse.load_npz('bigram/all_bytebigram.npz')
byte_vector.shape
```

In [ ]:
```
class_value=pd.read_csv("MMD/trainLabels.csv")
file_list =[ i.split('.')[0] for i in os.listdir('byteFiles')] # getting text
 file as ID
file_list = pd.DataFrame(file_list, columns=['Id']) #converting list in to Pan
das Dataframe
y_value = file_list.merge(class_value, on='Id') #Based on Id getting Class val
ue
y_value.head()
```

In [ ]:
```
imp_feature_index=imp_features(byte_vector,y_value['Class'], bi_gram_vocab,500
)
```

In [ ]:
```
top_500_bigram_feature =byte_vector[:,imp_feature_index]
bi_500_gram_vocab = [bi_gram_vocab[i] for i in imp_feature_index]
top_500_bigram_feature.shape
```

In [ ]:
```
byte_bi_df = pd.DataFrame.sparse.from_spmatrix(top_500_bigram_feature, columns
= bi_500_gram_vocab)
byte_bi_df['Id']=y_value['Id']
byte_bi_df.to_csv("data/500-bi-gram_feature.csv",index=False)
byte_bi_df.head()
```

In [51]:
```
byte_bi_df= pd.read_csv("data/500-bi-gram_feature.csv")
byte_bi_df.head()
```

Out[51]:

	fd a4	8f 99	f7 99	91 26	2b 97	44 49	5e 5f	fc 20	98 87	92 2f	...	00 52	3c 2f	00 d1	00 96	8c 00	00 88	6 7
0	0.0	0.0	0.0	0.0	0.0	12.0	3.0	0.0	0.0	1.0	...	106.0	0.0	3.0	4.0	3.0	10.0	10.
1	0.0	0.0	3.0	0.0	1.0	471.0	37.0	16.0	1.0	1.0	...	113.0	3.0	18.0	10.0	51.0	105.0	4.
2	5.0	2.0	2.0	3.0	2.0	5.0	1.0	1.0	2.0	1.0	...	8.0	2.0	2.0	1.0	4.0	7.0	1.
3	2.0	3.0	1.0	4.0	3.0	11.0	6.0	3.0	3.0	0.0	...	58.0	5.0	2.0	9.0	18.0	33.0	3.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	5.0	0.0	4.0	4.0	2.0	5.0	0.

5 rows × 501 columns

Opcode Bigram

In [ ]:
```
##startb
from tqdm.notebook import tqdm
def opcode_collect():
    op_file = open("opcode_file.txt", "w+")
    for asmfile in tqdm(os.listdir('asmFiles')):
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='re
place') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
opcode_collect()
```

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

In [ ]:
```
from tqdm.notebook import tqdm
from sklearn.feature_extraction.text import CountVectorizer
from scipy.sparse import csr_matrix
vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asmopcodebigram)
opcodebivect = csr_matrix((10868, len(asmopcodebigram)))
raw_opcode = open('opcode_file.txt').read().split('\n')

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

In [ ]:
```
import scipy.sparse
scipy.sparse.save_npz('bigram/opcodebigram.npz', opcodebivect)
```

In [ ]:
```
opcodebi_df = pd.DataFrame.sparse.from_spmatrix(opcodebivect, columns = asmopc
odebigram)
asm_list =pd.DataFrame([i.split('.')[0] for i in os.listdir('asmFiles')],colum
ns=['Id'])
opcodebi_df['Id'] = asm_list['Id']
opcodebi_df.to_csv("opcode_bi-gram.csv",index=False)
opcodebi_df.head()
```

In [52]:
```
opcodebi_df=pd.read_csv("opcode_bi-gram.csv")
opcodebi_df.head()
```

Out[52]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	jmp inc	...	movzx call	movzx shl	movzx ror	movzx rol
0	0.0	409.0	0.0	158.0	5.0	48.0	63.0	0.0	2.0	1.0	...	0.0	2.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	29.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
4	5.0	59.0	0.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

5 rows × 677 columns

In [54]:
```
byte_bi_df.shape,opcodebi_df.shape
```

Out[54]:
```
((10868, 501), (10868, 677))
```

Combining (API Feature+opcode+keywords) as ASM +img+ bi-gram Byte

In [67]:
```
print("Image Feature shape",df_image.shape)
print("Asm feature Feature shape",df_asm_feature.shape)
print("BiGram byte Feature shape",byte_bi_df.shape)
```

```
Image Feature shape (10868, 1002)
Asm feature Feature shape (10868, 1007)
BiGram byte Feature shape (10868, 501)
```

In [68]:
```
final_train = pd.merge(df_image, df_asm_feature,on='Id', how='left')
final_train.shape
```

Out[68]: (10868, 2008)

In [71]:
```
final_train = pd.merge(final_train, byte_bi_df,on='Id', how='left')
final_train.shape
```

Out[71]: (10868, 2507)

In [72]:
```
final_train.head()
```

Out[72]:

	Id	ASM_0	ASM_1	ASM_2	ASM_3	ASM_4	ASM_5	ASM_6	ASM_7
0	01IsoiSMh5gxyDYTl4CB	46	116	101	120	116	58	48	48
1	01SuzwMJEIXsK7A8dQbl	72	69	65	68	69	82	58	48
2	01azqd4InC7m9JpocGv5	72	69	65	68	69	82	58	48
3	01jsnpXSAlgw6aPeDxrU	72	69	65	68	69	82	58	48
4	01kcPWA9K2BOxQeS5Rju	72	69	65	68	69	82	58	49

5 rows × 2507 columns

In [73]:
```
Y=pd.read_csv("MMD/trainLabels.csv")
final_train = pd.merge(final_train,Y,on='Id', how='left')
```

In [74]:
```
final_train.isnull().values.any()
```

Out[74]: False

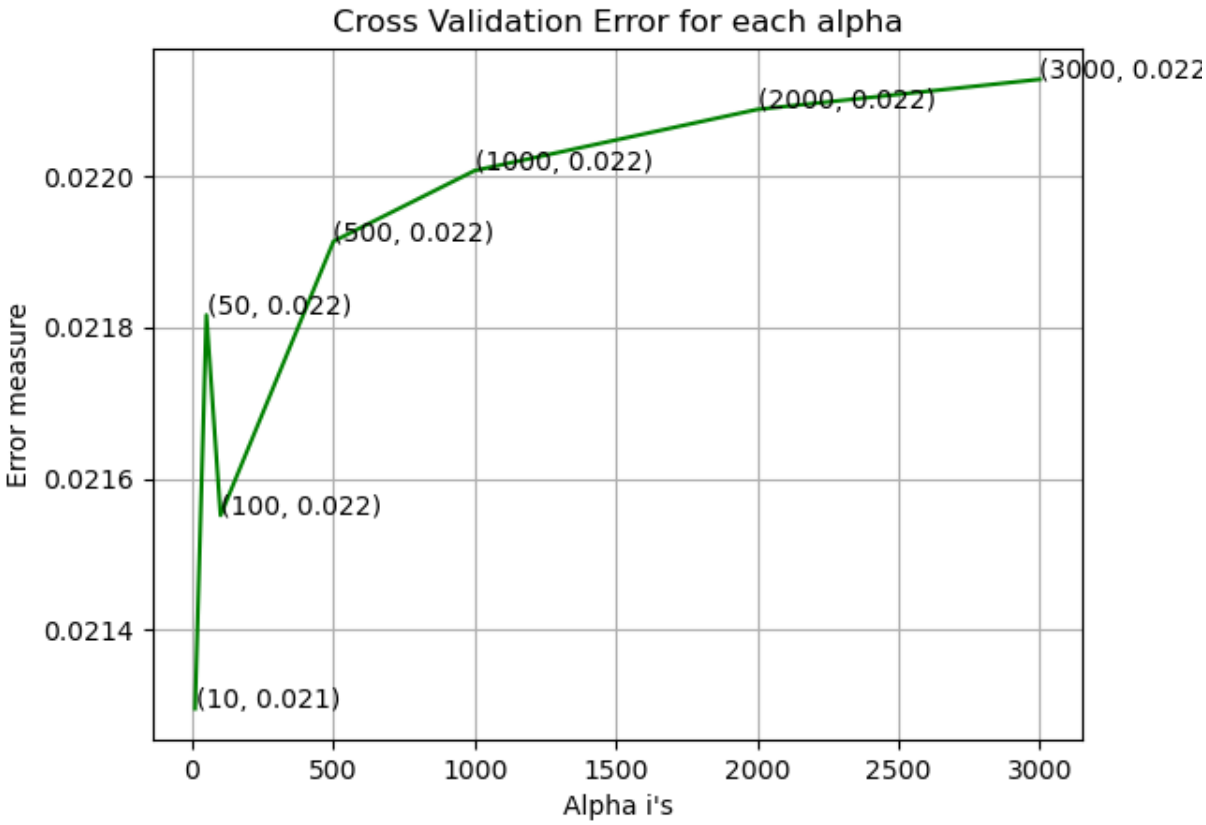
In [75]:
```
data_y = final_train['Class']
# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(final_train.drop(['Id','Cl
ass'], axis=1), data_y,stratify=data_y,test_size=0.20,random_state=42)
# split the train data into train and cross validation by maintaining same dis
tribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_tr
ain,test_size=0.20,random_state=43)
print("X_train : ",X_train.shape)
print("X_cv : ",X_cv.shape)
print("X_test : ",X_test.shape)
```

```
X_train :  (6955, 2506)
X_cv :  (1739, 2506)
X_test :  (2174, 2506)
```

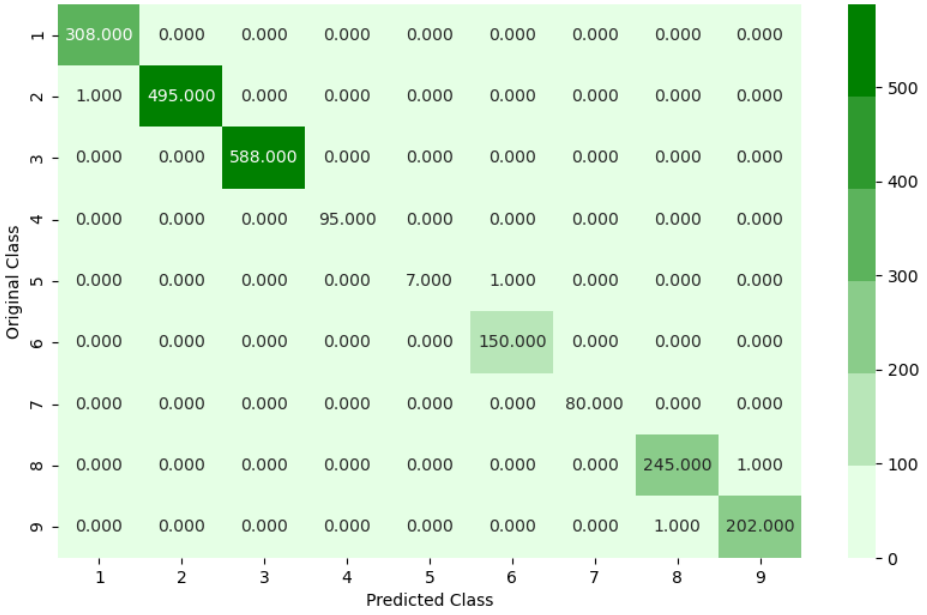
In [76]:

```
%%time
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1,clas
s_weight="balanced")
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_,
eps=1e-15))
# filename = '009_Random_forest_asm_bye_img_4gram_asm_cv_log_error_array.sav'
# pickle.dump(sig_clf, open('Models/'+filename, 'wb'))

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_
jobs=-1,class_weight="balanced")
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)
filename = '010_Random_img_asm_forest_asm_model_r_cfl.sav'
pickle.dump(r_cfl, open('Models/'+filename, 'wb'))

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

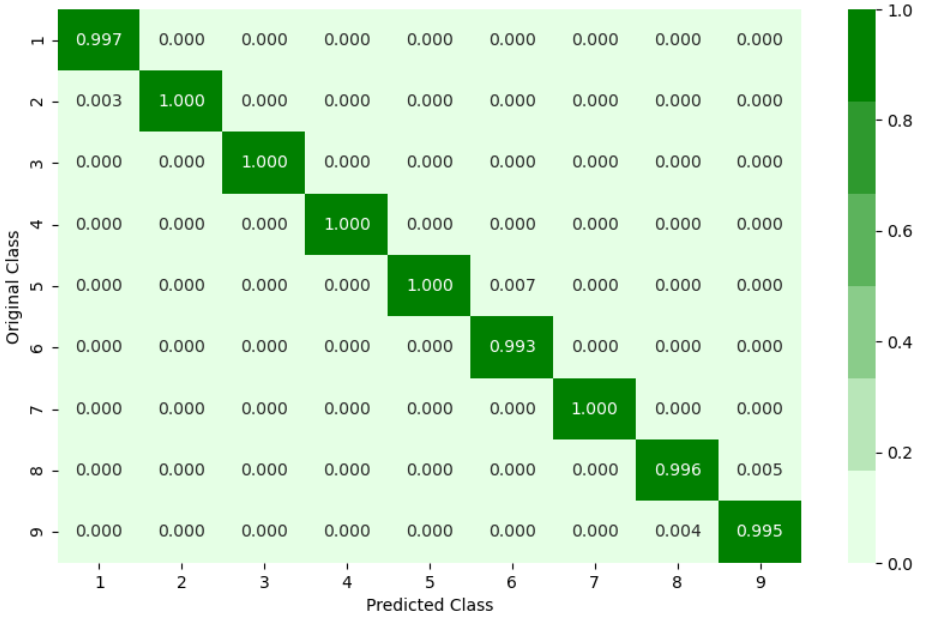
```
log_loss for c =  10 is 0.021295764573551614
log_loss for c =  50 is 0.02181669318496596
log_loss for c =  100 is 0.021551441240414342
log_loss for c =  500 is 0.021914201185111035
log_loss for c =  1000 is 0.02200786976537463
log_loss for c =  2000 is 0.022088760454197404
log_loss for c =  3000 is 0.022128564750154376
```




```
For values of best alpha =  10 The train log loss is: 0.012505711275416127
For values of best alpha =  10 The cross validation log loss is: 0.0212957645
73551614
For values of best alpha =  10 The test log loss is: 0.01991644845429843
Number of misclassified points  0.18399264029438822
---------------------------------------------------- Confusion matrix ---------
----------------------------------------
```

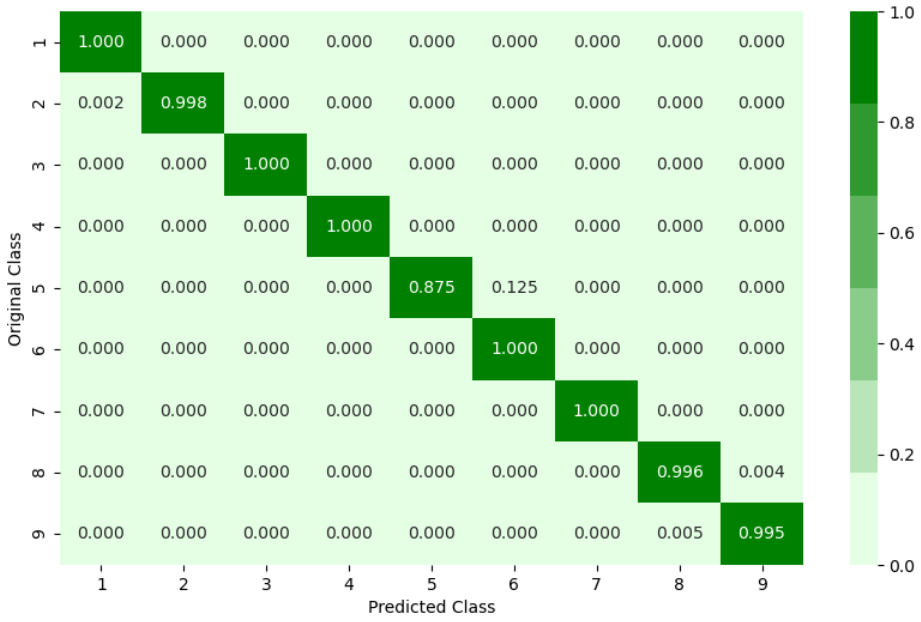


---------- Precision matrix ----------



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.]
CPU times: user 29min 16s, sys: 34.8 s, total: 29min 51s
Wall time: 5min 11s
```

In [ ]:
```
del final_train
```

only Image and ASM feature is used

In [11]:
```
final_train = pd.merge(df_image, df_asm_feature,on='Id', how='left')
final_train.shape
Y=pd.read_csv("MMD/trainLabels.csv")
final_train = pd.merge(final_train,Y,on='Id', how='left')
```

Out[11]:
```
(10868, 2007)
```

In [12]:
```
Y=pd.read_csv("MMD/trainLabels.csv")
final_train = pd.merge(final_train,Y,on='Id', how='left')
```

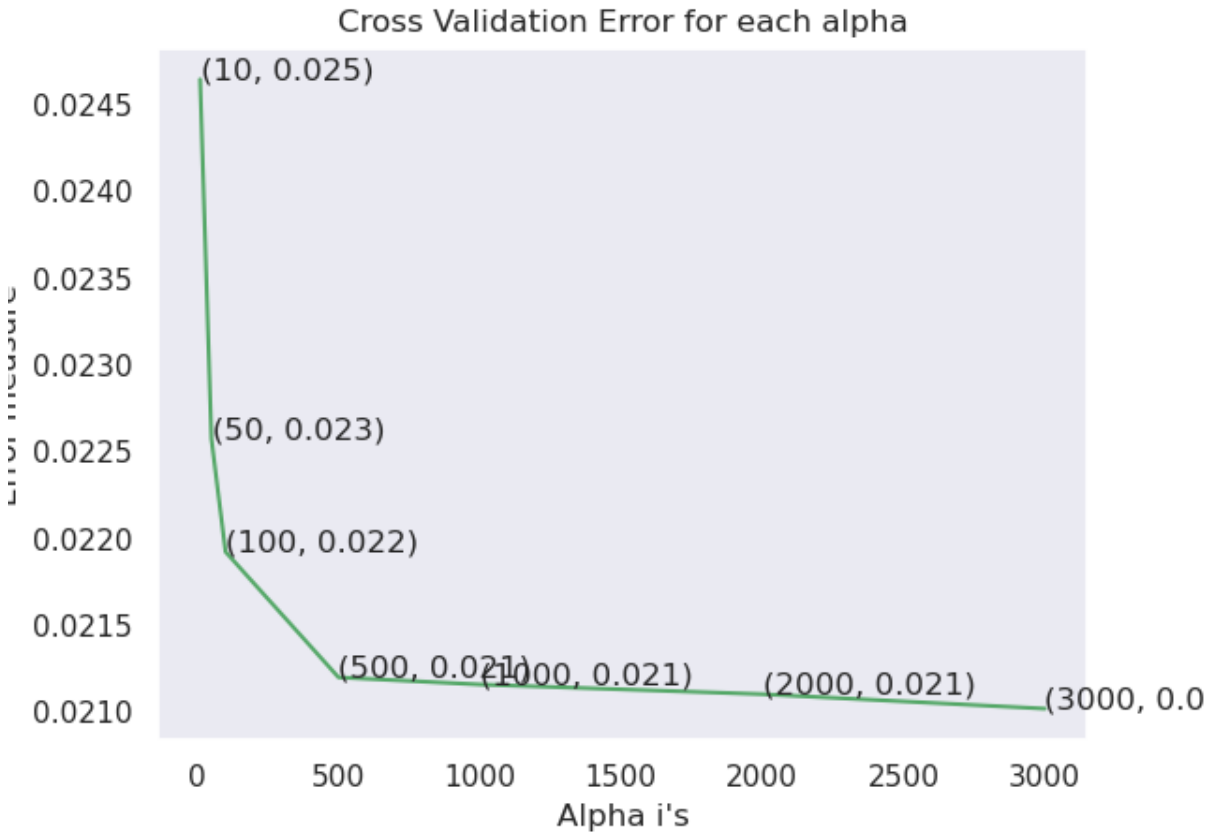
In [13]:
```
data_y = final_train['Class']
# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(final_train.drop(['Id','Cl
ass'], axis=1), data_y,stratify=data_y,test_size=0.20,random_state=42)
# split the train data into train and cross validation by maintaining same dis
tribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_tr
ain,test_size=0.20,random_state=43)
print("X_train : ",X_train.shape)
print("X_cv : ",X_cv.shape)
print("X_test : ",X_test.shape)
```

```
X_train :  (6955, 2006)
X_cv :  (1739, 2006)
X_test :  (2174, 2006)
```

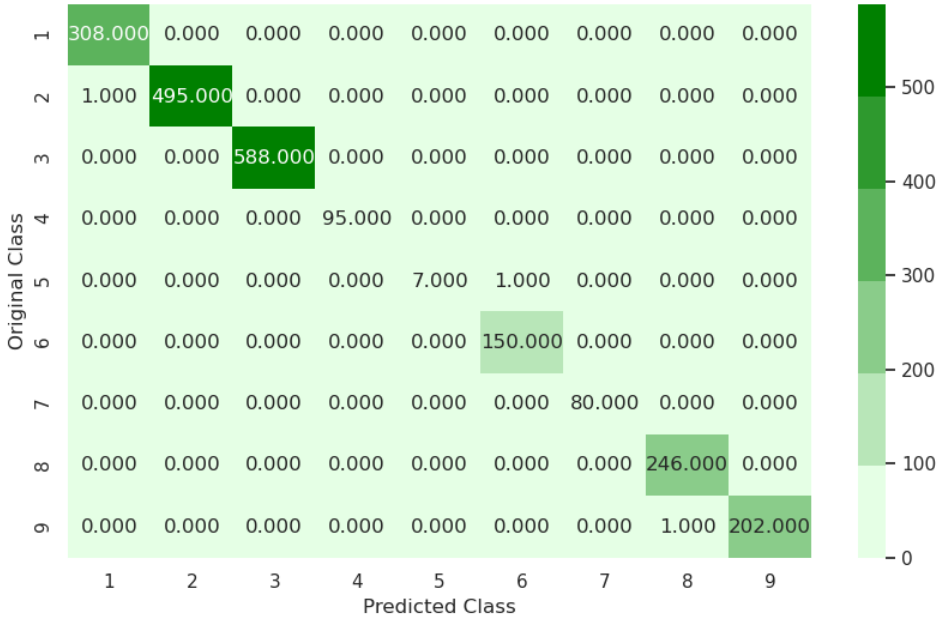
In [150]:

```
%%time
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1,clas
s_weight="balanced")
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_,
eps=1e-15))
filename = '0011_Random_forest_asm and_img_cv_log_error_array.sav'
pickle.dump(sig_clf, open('data/'+filename, 'wb'))

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_
jobs=-1,class_weight="balanced")
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)
filename = '011_Random__forest_asm_img__model_r_cfl.sav'
pickle.dump(r_cfl, open('data/'+filename, 'wb'))

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

```
log_loss for c =  10 is 0.024648286660537908
log_loss for c =  50 is 0.022576971552161738
log_loss for c =  100 is 0.02192494207317405
log_loss for c =  500 is 0.02120093006434583
log_loss for c =  1000 is 0.021159967381247428
log_loss for c =  2000 is 0.02110407117240446
log_loss for c =  3000 is 0.0210207593304968
```




```
For values of best alpha =  3000 The train log loss is: 0.011090688406979457
For values of best alpha =  3000 The cross validation log loss is: 0.02102075
93304968
For values of best alpha =  3000 The test log loss is: 0.01574697304527489
Number of misclassified points  0.13799448022079117
---------------------------------------------------- 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.]
CPU times: user 26min 59s, sys: 1min 16s, total: 28min 15s
Wall time: 6min 4s
```

In [16]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15],
     'n_estimators':[200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl1.fit(X_train,y_train)
filename = '11_xgboost_asm_img_feature.sav'
pickle.dump(random_cfl1, open("data/"+filename, 'wb'))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 38.4min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 53.9min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 85.3min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 113.0min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 121.8min remaining: 26.7min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 129.0min remaining:  8.2min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 129.9min finished
```

In [17]:
```
random_cfl1.best_params_
```

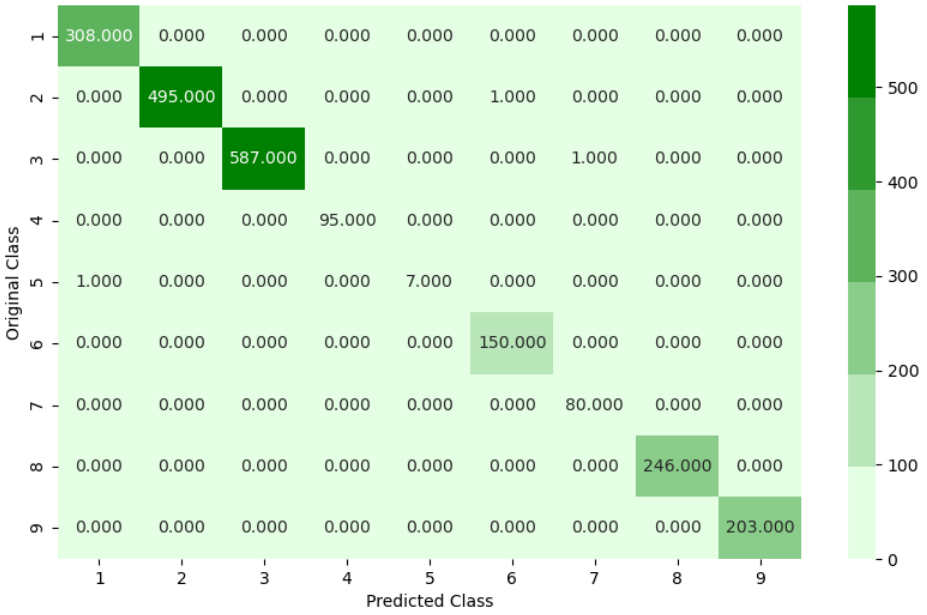
Out[17]:
```
{'subsample': 0.1,
 'n_estimators': 1000,
 'max_depth': 10,
 'learning_rate': 0.15,
 'colsample_bytree': 0.3}
```

In [18]:
```
subsample = random_cfl1.best_params_['subsample']
n_estimators=  random_cfl1.best_params_['n_estimators']
max_depth =  random_cfl1.best_params_['max_depth']
learning_rate =  random_cfl1.best_params_['learning_rate']
colsample_bytree =  random_cfl1.best_params_['colsample_bytree']
```

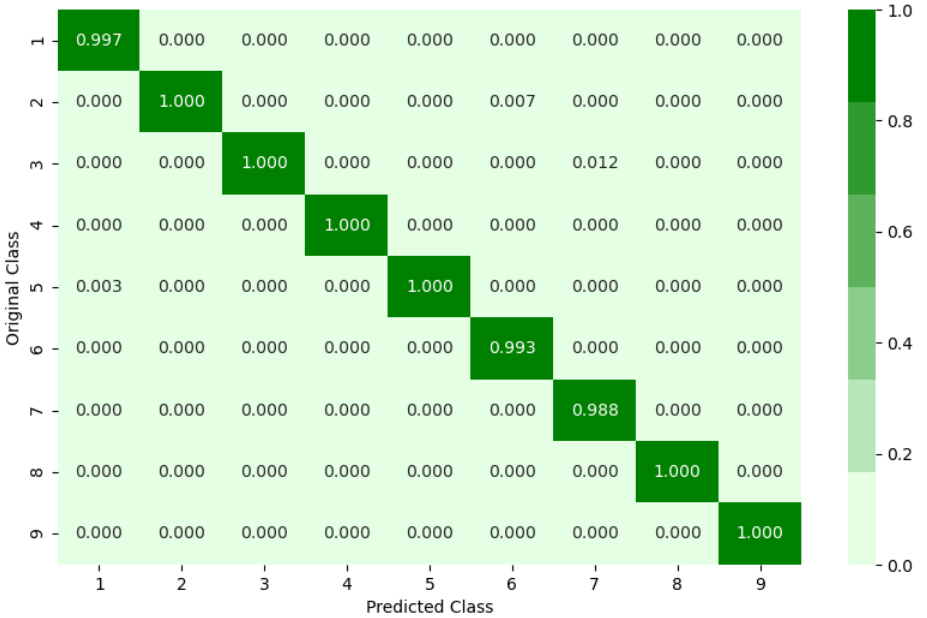
In [20]:

```
x_cfl=XGBClassifier(n_estimators=n_estimators, learning_rate=learning_rate, co
lsample_bytree= colsample_bytree, max_depth=max_depth,subsample=subsample,n_jo
bs=-1,)
x_cfl.fit(X_train,y_train)
filename1 = '0012_xgboost_x_cfl1_asm_img_feature.sav'
pickle.dump(x_cfl, open("data/"+filename1, 'wb'))

c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

filename2 = '0012_xgboost_asm_img_feature.sav'
pickle.dump(c_cfl, open("data/"+filename2, 'wb'))
# c_cfl = pickle.load(open("Models/"+filename2, 'rb'))

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

```
train loss 0.013965069811241897
cv loss 0.03130530573603067
test loss 0.017909803607904805
Number of misclassified points  0.13799448022079117
---------------------------------------------------- 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.]
```

Feature ASM +img +byte +byte gram+asm bigram

In [44]:
```
byte_features_with_size=pd.read_csv("MMD1/result_with_size.csv").drop(columns=
['Unnamed: 0','Class'],axis=1).rename(columns={'ID':'Id'})
byte_features_with_size.shape
```

Out[44]: (10869, 259)

In [45]:
```
final_train = pd.merge(final_train, byte_features_with_size,on='Id', how='lef
t')
final_train.shape
```

Out[45]: (10869, 2266)

In [55]:
```
# byte_bi_df.shape,opcodebi_df.shape
final_train = pd.merge(final_train, byte_bi_df,on='Id', how='left')
final_train = pd.merge(final_train, opcodebi_df,on='Id', how='left')
final_train.shape
```

Out[55]: (10869, 3442)

In [56]:
```
data_y = final_train['Class']
# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(final_train.drop(['Id','Cl
ass'], axis=1), data_y,stratify=data_y,test_size=0.20,random_state=42)
# split the train data into train and cross validation by maintaining same dis
tribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_tr
ain,test_size=0.20,random_state=43)
print("X_train : ",X_train.shape)
print("X_cv : ",X_cv.shape)
print("X_test : ",X_test.shape)
```

```
X_train :  (6956, 3440)
X_cv :  (1739, 3440)
X_test :  (2174, 3440)
```

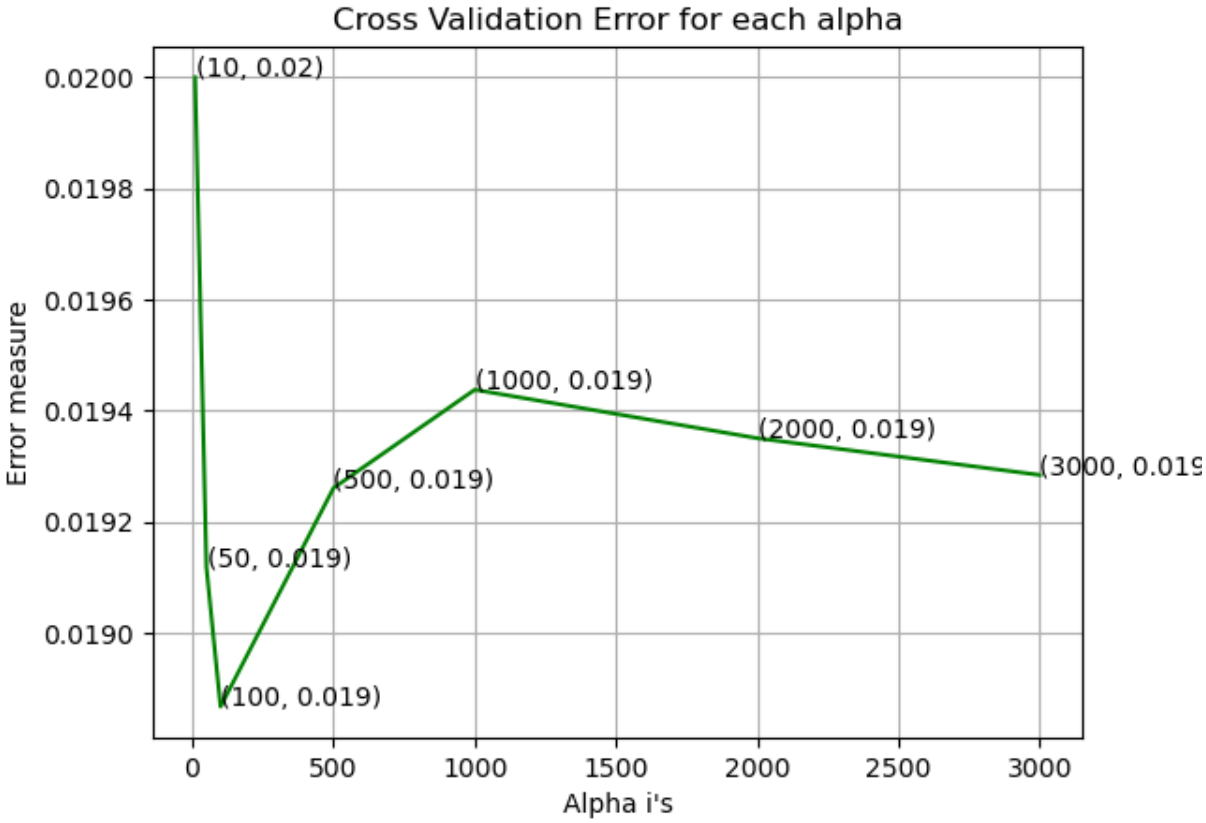
In [64]:
```
X_train.isnull().values.any()
X_test .isnull().values.any()
```

Out[64]:
```
False
```

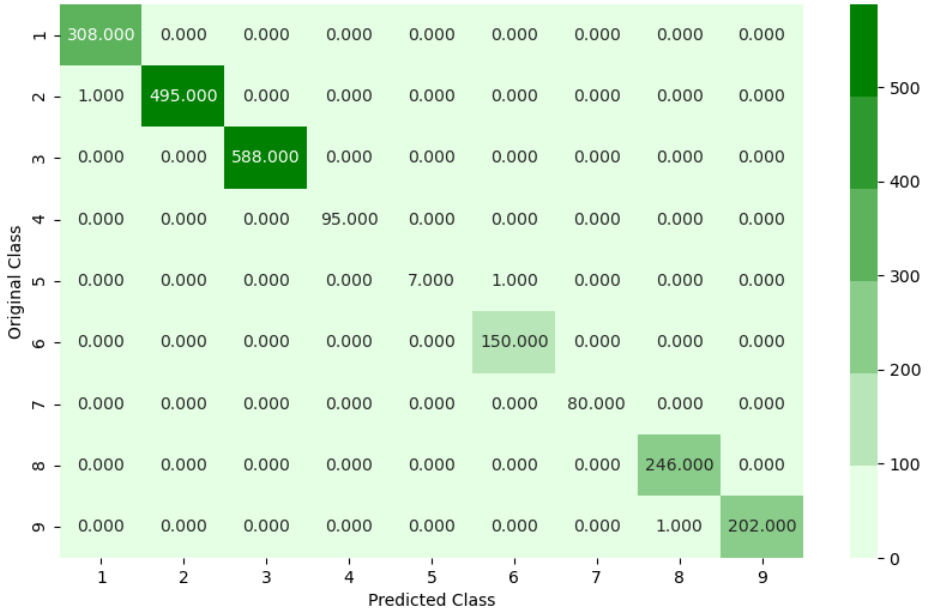
In [65]:

```
%%time
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1,clas
s_weight="balanced")
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_,
eps=1e-15))
# filename = '0011_Random_forest_asm and_img_cv_log_error_array.sav'
# pickle.dump(sig_clf, open('data/'+filename, 'wb'))

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_
jobs=-1,class_weight="balanced")
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)
# filename = '011_Random__forest_asm_img__model_r_cfl.sav'
# pickle.dump(r_cfl, open('data/'+filename, 'wb'))

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

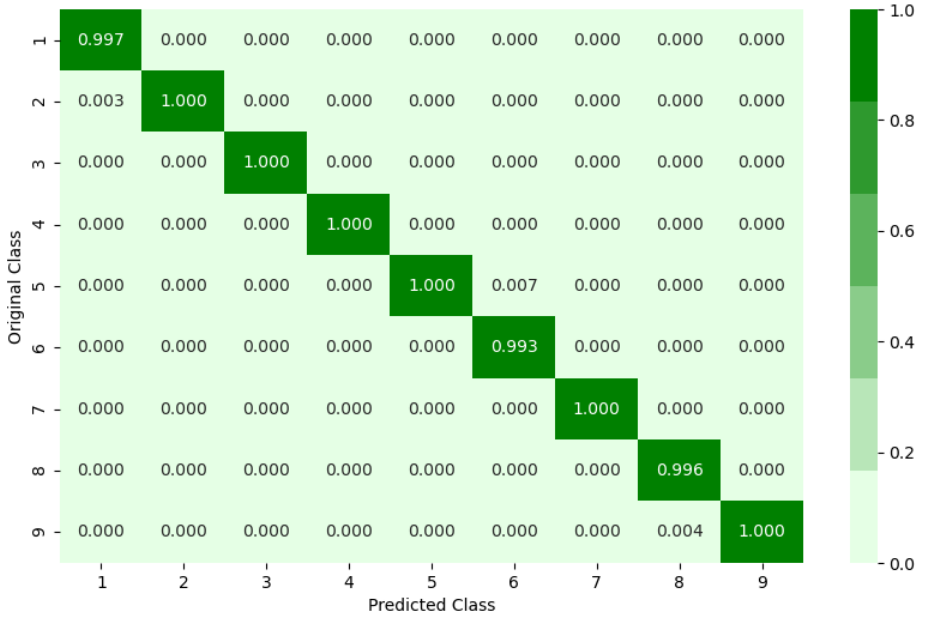
```
log_loss for c =  10 is 0.019999884152768925
log_loss for c =  50 is 0.01911879871142681
log_loss for c =  100 is 0.018867733909471197
log_loss for c =  500 is 0.019261107274040606
log_loss for c =  1000 is 0.019437961941572747
log_loss for c =  2000 is 0.019350664731982544
log_loss for c =  3000 is 0.0192845089146984
```




```
For values of best alpha =  100 The train log loss is: 0.010829050375701347
For values of best alpha =  100 The cross validation log loss is: 0.018867733
909471197
For values of best alpha =  100 The test log loss is: 0.016645127080542887
Number of misclassified points  0.13799448022079117
-------------------------------------------------- Confusion matrix ---------
----------------------------------------
```

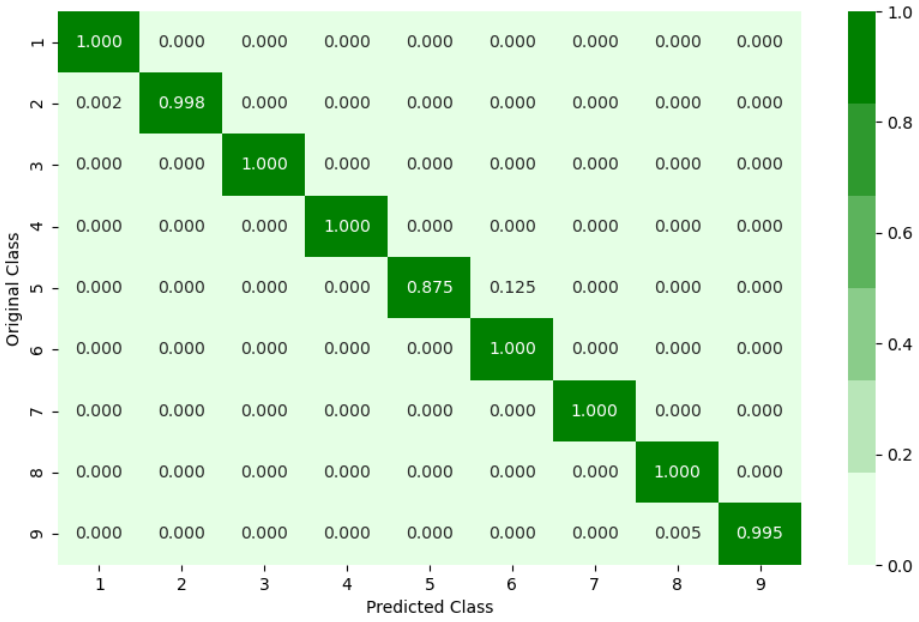


---------- Precision matrix ----------



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.]
CPU times: user 41min 34s, sys: 31 s, total: 42min 5s
Wall time: 6min 40s

In [77]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','train log loss','Test log loss']
ptable.add_row(["random","Byte files","2.49","2.49"])
ptable.add_row(["KNN ","unigram Byte files","0.114","0.21"])
ptable.add_row(["Logistic Regression","Unigram Byte files"," 0.875","0.88"])
ptable.add_row(["RandomForest","Unigram Byte files"," 0.0261","0.095"])
ptable.add_row(["Xgboost "," Unigram Byte files"," 0.021","0.071"])
ptable.add_row(["\n","\n","\n",""])

ptable.add_row(["KNN "," Unigram ASM files"," 0.023","0.085"])
ptable.add_row(["Logistic Regression","Unigram ASM files"," 1.00","0.992"])
ptable.add_row(["RandomForest","Unigram ASM files"," 0.0141","0.0424"])
ptable.add_row(["Xgboost "," Unigram ASM files"," 0.013","0.033"])

ptable.add_row(["\n","\n","\n",""])

ptable.add_row(["RandomForest","Unigram ASM+Byte files"," 0.0167","0.0424"])
ptable.add_row(["Xgboost "," Unigram ASM+Byte files"," 0.0128","0.027"])

ptable.add_row(["\n","\n","\n",""])
ptable.add_row(["RandomForest","bi-gram Byte files 66049 features"," 0.0210",
"0.066"])
ptable.add_row(["\n","\n","\n",""])
ptable.add_row(["Randomforest "," ASM API +ASM Img + bi-gram Byte files 500 fe
atures","  0.0125","0.0199"])
ptable.add_row(["\n","\n","\n",""])
ptable.add_row(["RandomForest "," ASM API +ASM img  +byte bi-gram + asm bigra
m"," 0.0108","0.019"])
ptable.add_row(["\n","\n","\n",""])
ptable.add_row(["RandomForest "," ASM API +ASM img"," 0.0110","0.0157"])
ptable.add_row(["XGBoost "," ASM API +ASM img"," 0.0139","0.0179"])
print(ptable)
```

Model	Features	train log loss	Test log loss
random	Byte files	2.49	2.49
KNN	unigram Byte files	0.114	0.21
Logistic Regression	Unigram Byte files	0.875	0.88
RandomForest	Unigram Byte files	0.0261	0.095
Xgboost	Unigram Byte files	0.021	0.071
KNN	Unigram ASM files	0.023	0.085
Logistic Regression	Unigram ASM files	1.00	0.992
RandomForest	Unigram ASM files	0.0141	0.0424
Xgboost	Unigram ASM files	0.013	0.033
RandomForest	Unigram ASM+Byte files	0.0167	0.0424
Xgboost	Unigram ASM+Byte files	0.0128	0.027
RandomForest	bi-gram Byte files 66049 features	0.0210	0.066
Randomforest	ASM API +ASM Img + bi-gram Byte files 500 features	0.0125	0.0199
RandomForest	ASM API +ASM img +byte bi-gram + asm bigram	0.0108	0.019

RandomForest	ASM API +ASM img	0.0110	0.0157	
XGBoost	ASM API +ASM img	0.0139	0.0179	