Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

.asm file

```
.text:00401000                                 assume es:nothing, ss:nothing, ds:_data,    fs:nothing, gs:nothing
.text:00401000 56                            push    esi
.text:00401001 8D 44 24    08                       lea     eax, [esp+8]
.text:00401005 50                            push    eax
.text:00401006 8B F1                            mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                        call    ??0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * 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
```

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss

- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

3. Exploratory Data Analysis

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

Type *Markdown* and LaTeX: α^2

In [2]:
```
!pip install kaggle
from google.colab import files
from datetime import datetime
api_token = files.upload()
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: kaggle in /usr/local/lib/python3.8/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.8/dist-packages (from kaggle) (2022.9.24)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.8/dist-packages (from kaggle) (7.0.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from kaggle) (4.64.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.8/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (2.10)
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

In [5]:
```
!mkdir ~/.kagglek
!cp kaggle.json ~/.kaggle/
```

```
mkdir: cannot create directory ‘/root/.kagglek’: File exists
```

In [6]:
```
!kaggle competitions download -c 'malware-classification'
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading malware-classification.zip to /content
100% 35.3G/35.3G [03:30<00:00, 190MB/s]
100% 35.3G/35.3G [03:30<00:00, 180MB/s]
```

In [7]:
```
!unzip malware-classification
```

```
Archive:  malware-classification.zip
  inflating: dataSample.7z
  inflating: sampleSubmission.csv
  inflating: test.7z
  inflating: train.7z
  inflating: trainLabels.csv
```

In [8]:
```
# Keep only train.7z, trainLables files remove other files as shown below
!rm test.7z
!rm /content/sampleSubmission.csv.zip
!rm dataSample.7z
```

```
rm: cannot remove '/content/sampleSubmission.csv.zip': No such file or directory
```

In [9]:
```
#
data=!7z l train.7z
print(len(data))
print(type(data))
```

```
21759
<class 'IPython.utils.text.SList'>
```

In [10]:
```
#Delete all the byte files
!rm -r -f byte_file_list/
```

In [11]:
```
#
data=!7z l train.7z
print(len(data))
print(type(data))
```

```
21759
<class 'IPython.utils.text.SList'>
```

In [12]:
```
byte_file_list=[]
asm_file_list=[]
for i in data:
  if (i.endswith("bytes")):
    byte_file_list.append(i)
  elif (i.endswith("asm")):
    asm_file_list.append(i)
```

In [13]:
```
print('Number of Byte files',len(byte_file_list))
```

```
Number of Byte files 10868
```

In [14]:
```
print('Number of ASM files',len(asm_file_list))
```

```
Number of ASM files 10868
```

In [15]:
```
byte_file_list[0]
```

Out[15]:
```
'2015-01-29 05:00:00 ....A      5256192              train/01azqd4InC7m9JpocGv5.bytes'
```

In [16]:
```
# Removing extra character and to get only the file name
(byte_file_list[0].split()[-1]).replace('train/', '')
```

Out[16]:
```
'01azqd4InC7m9JpocGv5.bytes'
```

In [17]:
```
# Doing the same operation for all the files
byte_file_name=[]
for i in range(0,len(byte_file_list)):
  byte_file_name.append(byte_file_list[i].split()[-1].replace('train/', ''))
```

In [18]:
```
byte_file_name[:3]
```

Out[18]:
```
['01azqd4InC7m9JpocGv5.bytes',
 '01IsoiSMh5gxyDYTl4CB.bytes',
 '01jsnpXSAlgw6aPeDxrU.bytes']
```

In [19]:
```
# Finding only the file names of ASM files and storing the file names in an array
asm_file_name=[]
for i in range(0,len(asm_file_list)):
  asm_file_name.append(asm_file_list[i].split()[-1].replace('train/', ''))
```

In [20]:
```
asm_file_name[:3]
```

Out[20]:
```
['01azqd4InC7m9JpocGv5.asm',
 '01IsoiSMh5gxyDYTl4CB.asm',
 '01jsnpXSAlgw6aPeDxrU.asm']
```

In [21]:
```
import os
# Getting the current working directory
os.getcwd()
```

Out[21]:
```
'/content'
```

In [22]:
```
os.listdir()
```

Out[22]:
```
['.config',
 'sampleSubmission.csv',
 'result.csv',
 'kaggle.json',
 'asmoutputfile.csv',
 'trainLabels.csv',
 'malware-classification.zip',
 'train.7z',
 'trainLabels (1).csv',
 'sample_data']
```

In [23]:
```
byte_file_name[0]
```

Out[23]:
```
'01azqd4InC7m9JpocGv5.bytes'
```

In [24]:
```
# Downloading only a single byte file from train.7z
%%time
file_name=byte_file_name[0]
!7z e train.7z -o/content *$file_name -r
```

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R) Xeon(R) CPU @ 2.20GHz (406F0),ASM,AES-NI)

Scanning the drive for archives:
  0M Sca        1 file, 18810691091 bytes (18 GiB)

Extracting archive: train.7z
--
Path = train.7z
Type = 7z
Physical Size = 18810691091
Headers Size = 339764
Method = LZMA:24
Solid = +
Blocks = 94

      1% . train/kQEbWRHa04gOYDqM1NJ6.as                                        3% . train/kQEbWRHa04gOYDqM1NJ6.as                                        6% 1 . train/K
qEgONxfHdP5lLaBIGQk.as                                      8% 1 . train/KqEgONxfHdP5lLaBIGQk.as                                     11% 1 . train/KqEgONxfHdP5l
LaBIGQk.as                                 13% 1 . train/KqEgONxfHdP5lLaBIGQk.as                                 16% 1 . train/KqEgONxfHdP5lLaBIGQk.as
18% 4 . train/KqS2u8HUWPI6jGDEhvB4.as                                     20% 5 . train/kQsiVxDbAXt23wRWal57.as                                     21% 5 . trai
n/kQsiVxDbAXt23wRWal57.as                                  23% 6 . train/kqvJp5E0wbWgu9mnzSQB.as                                  25% 6 . train/kqvJp5E0wb
Wgu9mnzSQB.as                              27% 6 . train/kqvJp5E0wbWgu9mnzSQB.as                              29% 6 . train/kqvJp5E0wbWgu9mnzSQB.a
s                                  31% 7 . train/KQwj9O6dlPxNyf8zW0gp.as                                  33% 9 . train/KRBDxPLfj0JQV5heGX4S.as
35% 9 . train/KRBDxPLfj0JQV5heGX4S.as                                     38% 9 . train/KRBDxPLfj0JQV5heGX4S.as                                     40% 9 . trai
n/KRBDxPLfj0JQV5heGX4S.as                                  44% 9 . train/KRBDxPLfj0JQV5heGX4S.as                                  48% 9 . train/KRBDxPLfj0
JQV5heGX4S.as                              51% 10 . train/KRNHAm094TC7OJfEPp8h.a                              55% 10 . train/KRNHAm094TC7OJfEPp8h.
a                                  59% 13 . train/KsaoU1ZWTfqSFYbOxPg8.a                                  64% 14 . train/ksKnqcBVTC0a3zSGoveR.a
68% 15 . train/kSNnYl3ZLvB2WI7V4iEt.a                                     73% 15 . train/kSNnYl3ZLvB2WI7V4iEt.a                                     76% 22 . tra
in/l06yuHvrqdzTo8CQeAO5.a                                  80% 30 . train/L2QwdUyG0HISDVk8rcsE.a                                  83% 39 . train/LCrZhlaY3
MnDA1GKwtvT.a                              86% 42 . train/LejcaXxAEgC31WhlswQ2.a                              90% 48 . train/LH5pzdDSPOtgIaBC1jWo.
a                                  93% 53 . train/ljuryB4bfagHqV5FM9Ae.a                                  97% 58 . train/loIP1tiwELF9YNZQjSUO.a
99% 63 - train/01azqd4InC7m9JpocGv5.byt                                      Everything is Ok

Files: 64
Size:       810098483
Compressed: 18810691091
CPU times: user 218 ms, sys: 47 ms, total: 265 ms
Wall time: 7.81 s
```

In [25]:
```
# Deleting the file
os.remove(file_name)
```

In [26]:
```
!mkdir asmfiles
```

In [27]:
```
#To download all the byte files from train.7z and below code will take almost 15mins.
start=datetime.now()

!7z e train.7z -o/content/bytefiles *.bytes -r

end=datetime.now()
difference=end-start
print('Time taken to extract all the byte files',difference)
```

```
4% 306 - train/0qV43bnIcBDPF1WlG6tJ.byte                4% 309 - train/0raU6iGvjZREQJYB1HX5.byte
4% 310 - train/0RCcpJ1DuKNMBsaQUgmA.byte                4% 315 - train/0rlkeuBLpWm15TyFRHUJ.byte
4% 321 - train/0ScbCK2aI4xrwOALDmNd.byte                4% 325 - train/0SHJ2aOnMCUGdLZrTQDV.byte
4% 328 - train/0SkbpFZGvraL6fQXTzol.byte                4% 331 - train/0sM6DQlxcP3oAfd9ZVBT.byte
4% 333 - train/0SUAyQWoTeZb8qmIPMK3.byte                4% 335 - train/0TBj7glMRJScN9GmOzyi.byte
4% 337 - train/0TpWNF7ULhd4Yk5bXQyP.byte                4% 340 - train/0tZy7jgOKCIJslPRvrop.byte
4% 344 - train/0UcDbq2yojiThfdLRNS4.byte                4% 347 - train/0uLJEcpVlIWkTBeHZU4w.byte
4% 350 - train/0UTxRyirCXl6tsSE5GAv.byte                4% 351 - train/0UwHZzuXy7GFStRKEgdx.byte
4% 355 - train/0VfuK267xTdeBcmLaCp1.byte                4% 357 - train/0VLen4rGI1wNjOqzyZRb.byte
4% 363 - train/0wmZBjQenOTsu5bpSkf6.byte                4% 369 - train/0X8fojtR6QIblypvHLsM.byte
4% 372 - train/0xfjbVBPYX1pSt4rdWOe.byte                4% 379 - train/0XSHfwZAgYm7eFVInuDT.byte
4% 381 - train/0xUc2vRzyYtqFa4VhLnG.byte                4% 385 - train/0YIPlyX3fbNs2ntJRGgH.byte
4% 386 - train/0yjkG4BHeCbTimIslgr7.byte                4% 390 - train/0yvBxsufKXzM6CwpcSYU.byte
4% 392 - train/0Z3nQDPiU7LMEesmG4TB.byte                4% 396 - train/0ZHV6acpJ9KkAWPjEI71.byte
4% 400 - train/0zr6vIw74DgVSKXiRUY1.byte                4% 405 - train/125y4VsArzkCNOGZfu6o.byte
4% 407 - train/12Jd4qpOzTtQC3E6PXDb.byte                4% 409 - train/12TEseaJ4jnbyORqKxhf.byte
4% 411 - train/13FZ9DOhywk78sJiVULu.byte                4% 414 - train/13YpdP5vTLOazSQFRgJn.byte
4% 416 - train/14DUiwIb5xrfkF30cThJ.byte                5% 421 - train/15mfGKRS6aVevjAoYL8w.byte
5% 423 - train/15tQNEDpb0mi63kfOWM8.byte                5% 426 - train/16eajI7F2vQKhwi9dM4O.byte
```

In [28]:
```
print('Number of byte files extracted is',len(os.listdir('bytefiles/')))
```

```
Number of byte files extracted is 10868
```

In [28]:

In [28]:

In [28]:

In [28]:

In [28]:

In [28]:

3.1. Distribution of malware classes in whole data set

In [29]:
```
Y=pd.read_csv("trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

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

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

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

3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [30]:
```
#file sizes of byte files

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

```
                     ID      size  Class
0  frLD0F7HvGAwkdWKsNYU  8.099609      3
1  A41Ov2nFcx8flziRwrap  1.005371      1
2  gA94Fpfcq5lRMvmkKXYJ  8.113770      3
3  K1MACq57pdQfEsgjclS9  0.396484      1
4  9toXzFrSARZ0TJGKMbV6  8.113770      3
```

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

In [31]:
```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

3.2.3 feature extraction from byte files

In [32]:
```
byte_features=pd.read_csv("result.csv")
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)
```

Out[32]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8	f9	fa	fb	fc	fd	fe	ff	??
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687	3101	3211	3097	2758	3099	2759	5753	1824
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536	439	281	302	7639	518	17001	54902	8588

2 rows × 258 columns

In [33]:
```
data_size_byte.head(2)
```

Out[33]:

	ID	size	Class
0	frLD0F7HvGAwkdWKsNYU	8.099609	3
1	A41Ov2nFcx8flziRwrap	1.005371	1

In [34]:
```
byte_features_with_size = byte_features.merge(data_size_byte, on='ID')
byte_features_with_size.to_csv("result_with_size.csv")
byte_features_with_size.head(2)
```

Out[34]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	size	Class
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759	5753	1824	5.012695	9
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	17001	54902	8588	6.556152	2

2 rows × 260 columns

In [35]:
```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
result = normalize(byte_features_with_size)
```

In [36]:
```
result.head(2)
```

Out[36]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.01356	0.013107	0.013634	0.031724	0.014549	0.014348	0.007843	0.000129	0.092
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.00192	0.001147	0.001329	0.087867	0.002432	0.088411	0.074851	0.000606	0.121

2 rows × 260 columns

In [37]:
```
data_y = result['Class']
result.head()
```

Out[37]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.013560	0.013107	0.013634	0.031724	0.014549	0.014348	0.007843	0.000129	0.(
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.001920	0.001147	0.001329	0.087867	0.002432	0.088411	0.074851	0.000606	0.
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.009804	0.011777	0.012604	0.028423	0.013080	0.013937	0.067001	0.000033	0.(
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.002121	0.001886	0.002272	0.013032	0.002211	0.003957	0.010904	0.000984	0.(
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.001530	0.000853	0.001052	0.007511	0.001038	0.001258	0.002998	0.000636	0.(

5 rows × 260 columns

3.2.4 Multivariate Analysis

In [38]:
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

In [39]:
```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

Train Test split

In [40]:
```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output 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 distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

In [41]:
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

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

In [42]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

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

```
print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

In [43]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.4786746951662932
Log loss on Test Data using Random Model 2.5085172748196287
Number of misclassified points  89.28242870285189
---------------------------------------------------- Confusion matrix -----------------------------------------------------

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>

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

4.1.2. K Nearest Neighbour Classification

In [44]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

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

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

```
log_loss for k =  1 is 0.24100686534884144
log_loss for k =  3 is 0.23445538821324652
log_loss for k =  5 is 0.25914061944983285
log_loss for k =  7 is 0.2748048487571393
log_loss for k =  9 is 0.2872390437253671
log_loss for k =  11 is 0.3014856770509452
log_loss for k =  13 is 0.31380668375657916

<IPython.core.display.Javascript object>


For values of best alpha =  3 The train log loss is: 0.11344239473068664
For values of best alpha =  3 The cross validation log loss is: 0.23445538821324652
For values of best alpha =  3 The test log loss is: 0.23521830497651106
Number of misclassified points  5.47378104875805
---------------------------------------------------- Confusion matrix ----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>


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

4.1.3. Logistic Regression

In [45]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.1939683701471913
log_loss for c =  0.0001 is 1.193940121542623
log_loss for c =  0.001 is 1.1906667368124408
log_loss for c =  0.01 is 1.1635431971183319
log_loss for c =  0.1 is 1.0519645404517977
log_loss for c =  1 is 1.0037735001121642
log_loss for c =  10 is 0.934191828902247
log_loss for c =  100 is 0.8936183005546295
log_loss for c =  1000 is 0.8832889143528865
```

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

```
log loss for train data 0.8478862698437228
log loss for cv data 0.8832889143528865
log loss for test data 0.8704767818122213
Number of misclassified points  28.28886844526219
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```

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

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

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

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

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

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

4.1.4. Random Forest Classifier

In [46]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

predict_y = sig_clf.predict_proba(X_train)
```

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

```
log_loss for c =  10 is 0.11038203351728029
log_loss for c =  50 is 0.10067793036198432
log_loss for c =  100 is 0.10028007915568642
log_loss for c =  500 is 0.09828660757683265
log_loss for c =  1000 is 0.09783198481012378
log_loss for c =  2000 is 0.09788072008671848
log_loss for c =  3000 is 0.09789772098079232

<IPython.core.display.Javascript object>


For values of best alpha =  1000 The train log loss is: 0.026326113266325016
For values of best alpha =  1000 The cross validation log loss is: 0.09783198481012378
For values of best alpha =  1000 The test log loss is: 0.08198128336391075
Number of misclassified points  1.7479300827966882
---------------------------------------------------- Confusion matrix -----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>


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

4.2 Modeling with .asm files

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


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

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

4.2.1 Feature extraction from asm files

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

In [47]:
```
#To download all the byte files from train.7z and below code will take almost 15mins.
start=datetime.now()

!7z e train.7z -o/content/asmfiles *.bytes -r

end=datetime.now()
difference=end-start
print('Time taken to extract all the asm files',difference)
```

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R) Xeon(R) CPU @ 2.20GHz (406F0),ASM,AES-NI)

Scanning the drive for archives:
  0M Sca        1 file, 18810691091 bytes (18 GiB)

Extracting archive: train.7z
--
Path = train.7z
Type = 7z
Physical Size = 18810691091
Headers Size = 339764
Method = LZMA:24
Solid = +
Blocks = 94

      0% . train/kQEbWRHa04gOYDqM1NJ6.as                                    0% 1 . train/KqEgONxfHdP5lLaBIGQk.as                                    0% 2 .
train/kqiOdVbRQlB2s907GLMv.as                                    0% 5 . train/kQsiVxDbAXt23wRWal57.as                                    0% 6 . train/kqv
```

In [192]:
```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the extracted features from .asm files
# this file will be uploaded in the drive, you can directly use this
dfasm=pd.read_csv("asmoutputfile.csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='left')
result_asm.head()
```

Out[192]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83	0	17	48	29	1
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12	0	14	0	20	1
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14	0	11	0	9	1
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	0	8	0	6	1
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	0	11	0	11	1

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [49]:
```
#file sizes of byte files

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

```
                     ID      size  Class
0  frLD0F7HvGAwkdWKsNYU  8.099609      3
1  A41Ov2nFcx8flziRwrap  1.005371      1
2  gA94Fpfcq5lRMvmkKXYJ  8.113770      3
3  K1MACq57pdQfEsgjclS9  0.396484      1
4  9toXzFrSARZ0TJGKMbV6  8.113770      3
```

4.2.1.2 Distribution of .asm file sizes

In [50]:
```
#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 [51]:
```
# add the file size feature to previous extracted features
print(result_asm.shape)
print(asm_size_byte.shape)
result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='ID', how='left')
result_asm.head()
```

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

Out[51]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	size
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	48	29	1	0.679688
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	0	20	1	0.396484
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	0	9	1	0.410645
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	0	6	1	0.396484
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	0	11	1	0.396484

5 rows × 54 columns

In [52]:
```
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

Out[52]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	size
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.001057	0.0	0.030797	0.001468	0.003173	1	0.010759
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.000153	0.0	0.025362	0.000000	0.002188	1	0.005435
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.000178	0.0	0.019928	0.000000	0.000985	1	0.005701
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.000025	0.0	0.014493	0.000000	0.000657	1	0.005435
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.000064	0.0	0.019928	0.000000	0.001204	1	0.005435

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [53]:
```
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 [54]:
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```

In [55]:
```
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 [56]:
```
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 [57]:
```
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 [58]:
```
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 [59]:
```
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 [60]:
```
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 [61]:
```
ax = sns.boxplot(x="Class", y="push", data=result_asm)
plt.title("boxplot of .asm push opcode")
plt.show()
```

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

4.2.2 Multivariate Analysis on .asm file features

In [62]:
```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic-neighbourhood-embeddingt-sne-part-1/

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

In [63]:
```
# by univariate analysis on the .asm file features we are getting very negligible information from
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing those features
# the plot looks very messy

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

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

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 [64]:
```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

In [65]:
```
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 [66]:
```
print( X_cv_asm.isnull().all())
```

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

4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

In [67]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)  Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

```
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


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

```
log_loss for k =  1 is 0.09547429326799622
log_loss for k =  3 is 0.1008910288331824
log_loss for k =  5 is 0.11274173337534614
log_loss for k =  7 is 0.11942280875629824
log_loss for k =  9 is 0.12506252600127538
log_loss for k =  11 is 0.13213210467589515
log_loss for k =  13 is 0.13851440108957136
log_loss for k =  15 is 0.14325481907157506
log_loss for k =  17 is 0.14621387108081335
log_loss for k =  19 is 0.15046980183145287
```

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


log loss for train data 0.025976901009384254
log loss for cv data 0.09547429326799622
log loss for test data 0.10419666383599757
Number of misclassified points  1.8399264029438822
---------------------------------------------------- Confusion matrix ----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>


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

4.4.2 Logistic Regression

In [68]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  1e-05 is 0.9507784586297774
log_loss for c =  0.0001 is 0.9533405510203488
log_loss for c =  0.001 is 0.9486118182019984
log_loss for c =  0.01 is 0.9231831977514223
log_loss for c =  0.1 is 0.8206984493468864
log_loss for c =  1 is 0.7583909731715867
log_loss for c =  10 is 0.8766418307282302
log_loss for c =  100 is 0.8615517043813025
log_loss for c =  1000 is 0.8566737007018138
```

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

```
log loss for train data 0.7568486363411877
log loss for cv data 0.7583909731715867
log loss for test data 0.7653138931521885
Number of misclassified points  24.42502299908004
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```

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

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

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

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

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

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

4.4.3 Random Forest Classifier

In [69]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

```
log_loss for c =  10 is 0.04139244741412242
log_loss for c =  50 is 0.03508364516447039
log_loss for c =  100 is 0.03490694673353031
log_loss for c =  500 is 0.0347821076489442
log_loss for c =  1000 is 0.035047439803334246
log_loss for c =  2000 is 0.03502537745848366
log_loss for c =  3000 is 0.03503639264979104
```

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

```
log loss for train data 0.014400280682281769
log loss for cv data 0.0347821076489442
log loss for test data 0.03660729229738254
Number of misclassified points  0.5979760809567618
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```

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

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

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

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

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

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

4.4.4 XgBoost Classifier

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

4.5.1. Merging both asm and byte file features

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

Out[70]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.013560	0.013107	0.013634	0.031724	0.014549	0.014348	0.007843	0.000129	0.(
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.001920	0.001147	0.001329	0.087867	0.002432	0.088411	0.074851	0.000606	0.
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.009804	0.011777	0.012604	0.028423	0.013080	0.013937	0.067001	0.000033	0.(
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.002121	0.001886	0.002272	0.013032	0.002211	0.003957	0.010904	0.000984	0.(
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.001530	0.000853	0.001052	0.007511	0.001038	0.001258	0.002998	0.000636	0.(

5 rows × 260 columns

In [71]:
```
result_asm.head()
```

Out[71]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	size
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.001057	0.0	0.030797	0.001468	0.003173	1	0.010759
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.000153	0.0	0.025362	0.000000	0.002188	1	0.005435
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.000178	0.0	0.019928	0.000000	0.000985	1	0.005701
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.000025	0.0	0.014493	0.000000	0.000657	1	0.005435
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.000064	0.0	0.019928	0.000000	0.001204	1	0.005435

5 rows × 54 columns

In [72]:
```
print(result.shape)
print(result_asm.shape)
```

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

In [73]:
```
result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[73]:

	0	1	2	3	4	5	6	7	8	9	...	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip	size_y
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.015418	0.025875	0.025744	0.004910	0.008930	0.0	0.027174	0.000428	0.049896	0.092219
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.004961	0.012316	0.007858	0.007570	0.005350	0.0	0.043478	0.000673	0.024839	0.121237
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.000095	0.006181	0.000100	0.003773	0.000713	0.0	0.048913	0.000000	0.012802	0.084499
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.000343	0.000746	0.000301	0.000360	0.001057	0.0	0.030797	0.001468	0.003173	0.010759
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.000343	0.013875	0.000482	0.012932	0.001363	0.0	0.027174	0.000000	0.008316	0.006233

5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

In [74]:
```
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 [75]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

4.5.4. Random Forest Classifier on final features

In [76]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
log_loss for c =  10 is 0.043154349598171084
log_loss for c =  50 is 0.041570891987482185
log_loss for c =  100 is 0.041330215019059
log_loss for c =  500 is 0.040407869191034236
log_loss for c =  1000 is 0.04000620062266539
log_loss for c =  2000 is 0.03988714750085982
log_loss for c =  3000 is 0.03978820911611415

<IPython.core.display.Javascript object>


For values of best alpha =  3000 The train log loss is: 0.016079391879821086
For values of best alpha =  3000 The cross validation log loss is: 0.03978820911611415
For values of best alpha =  3000 The test log loss is: 0.04122011867312097
```

4.5.5. XgBoost Classifier on final features

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video (https://www.youtube.com/watch?v=VLQTRlLGz5Y#t=13m11s)) and include pixel intensity features to improve the logloss

```
1. you need to donwload the train from kaggle, which is of size ~17GB, after extracting it will occupy ~128GB data your dirve

2. if you are having computation power limitations, you can try using google colab, with GPU option enabled (you can search for how to enable GPU in colab)
or you can work with the Google Cloud, check this tutorials by one of our student: https://www.youtube.com/channel/UCRH_z-oM0LROvHPe_KYR4Wg (we suggest you
to use GCP over Colab)

3. To Extract the .7z file in google cloud, once after you upload the file into server, in your ipython notebook create a new cell and write thess commands
  a. !sudo apt-get install p7zip
  b. !7z x file_name.7z -o path/where/you/want/to/extract

https://askubuntu.com/a/341637
```

In [77]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
# matplotlib.use('nbAgg')
%matplotlib notebook
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
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize
from nltk.util import ngrams
import h5py
import copy
```

In [78]:
```
%%time

uni_gram_byte_features = pd.read_csv( "result.csv")

uni_gram_byte_features['ID']  = uni_gram_byte_features['ID'].str.split('.').str[0]

print('Unigram byte_featues shape ', uni_gram_byte_features.shape)

uni_gram_byte_features.head(2)
```

```
Unigram byte_featues shape  (10868, 258)
CPU times: user 312 ms, sys: 25 ms, total: 337 ms
Wall time: 424 ms
```

Out[78]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8	f9	fa	fb	fc	fd	fe	ff	??
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687	3101	3211	3097	2758	3099	2759	5753	1824
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536	439	281	302	7639	518	17001	54902	8588

2 rows × 258 columns

#34. File sizes of Byte files - Feature Extraction -For FINAL Model Train

In [79]:
```
%%time
from tqdm import tqdm
# here we are ectracting byte file feature from bytefiles
Y=pd.read_csv("trainLabels.csv")

files=os.listdir( 'bytefiles')

filenames=Y['Id'].tolist()

class_y=Y['Class'].tolist()

class_bytes=[]

sizebytes=[]

fnames=[]

for file in tqdm(files):

    statinfo=os.stat('bytefiles/'+file)
    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)

byte_feature_size=pd.DataFrame({'ID':fnames, 'size':sizebytes,'Class':class_bytes})

print (byte_feature_size.head())
```

```
100%|██████████| 10868/10868 [00:05<00:00, 1878.24it/s]

                     ID      size  Class
0  frLD0F7HvGAwkdWKsNYU  8.099609      3
1  A41Ov2nFcx8flziRwrap  1.005371      1
2  gA94Fpfcq5lRMvmkKXYJ  8.113770      3
3  K1MACq57pdQfEsgjclS9  0.396484      1
4  9toXzFrSARZ0TJGKMbV6  8.113770      3
CPU times: user 5.6 s, sys: 272 ms, total: 5.87 s
Wall time: 5.83 s
```

#35. Creating some important Files and Folders, which I shall use later for saving Featuarized versions of .csv files

In [80]:
```
import pickle as pkl
```

In [81]:
```
if not os.path.isdir("featurization"):
    os.makedirs( "featurization")


if not os.path.isdir( "featurization/featurization_final"):
    os.mkdir( "featurization/featurization_final")
# here we are writinh the feature class_labe to get the labels of dataset from dt
class_labels=byte_feature_size["Class"]

with open('featurization/class_labels.pkl', 'wb') as file:
    pkl.dump(class_labels, file)

# Load the class class_labels for training with random forest feature selector

with open('featurization/class_labels.pkl', 'rb') as file:
    class_labels=pkl.load(file)
```

#36. Merging Unigram of Byte Files + Size of Byte Files to create uni_gram_byte_features__with_size

In [83]:
```
%%time

uni_gram_byte_features__with_size_1 = uni_gram_byte_features.merge(byte_feature_size, on="ID")
# HERE WE CONVERTING THE EXTRACTED FILE INTO CSV FILE
uni_gram_byte_features__with_size_1.to_csv("featurization/uni_gram_byte_features__with_size.csv" , index=False)
```

```
CPU times: user 945 ms, sys: 58.5 ms, total: 1 s
Wall time: 993 ms
```

In [84]:
```
uni_gram_byte_features__with_size_1.head(1)
```

Out[84]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	size	Class
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759	5753	1824	5.012695	9

1 rows × 260 columns

In [85]:
```
#HERE WE ARE DROPPING THE ID COLUMN BECAUSE I WANT TO NORMALIXE THE DATA
uni_gram_byte_features__with_size_2=uni_gram_byte_features__with_size_1.drop(['ID'], axis=1)
```

In [86]:
```
#HERE WE ARE NORMALIZING THE DATASET
uni_gram_byte_features__with_size = normalize(uni_gram_byte_features__with_size_2)
```

In [87]:
```
# reference : https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
%%time

from sklearn.feature_extraction.text import CountVectorizer


# THESE ARE THE  ALL THE TOKENS THAT WE HAVE FOR BYTE FEATURE EXTRACTION
bigram_tokens="00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,\
2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,\
59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,\
87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,\
b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,\
e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"

bigram_tokens=bigram_tokens.split(",")

def calculate_bigram(bigram_tokens):
    sentence=""
    vocabulary_list_for_byte_bigrams=[]
    for i in tqdm(range(len(bigram_tokens))):
        for j in range(len(bigram_tokens)):
            bigram=bigram_tokens[i]+" "+bigram_tokens[j]
            sentence=sentence+bigram+","
            vocabulary_list_for_byte_bigrams.append(bigram)
    return vocabulary_list_for_byte_bigrams

vocabulary_list_for_byte_bigrams = calculate_bigram(bigram_tokens)
```

```
100%|██████████| 257/257 [00:01<00:00, 225.23it/s]

CPU times: user 1.05 s, sys: 100 ms, total: 1.15 s
Wall time: 1.15 s
```

#37. Bi-Gram Byte Feature extraction from byte files

In [88]:
```
%%time
# reference: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

import scipy
vectorizer = CountVectorizer(tokenizer=lambda x: x.split(),lowercase=False, ngram_range=(2,2),vocabulary=vocabulary_list_for_byte_bigrams)

file_list_byte_files=os.listdir( 'bytefiles')

features=["ID"]+vectorizer.get_feature_names()

byte_file_bigram_df=pd.DataFrame(columns=features)

# Creating "featurization/byte_files_bigram_df.csv" and writng to it the full bi-gram data frame
with open( "featurization/byte_files_bigram_df.csv", mode='w') as byte_file_bigram_df:
    byte_file_bigram_df.write(','.join(map(str, features)))
    byte_file_bigram_df.write('\n')
    for _, file in tqdm(enumerate(file_list_byte_files)):
        file_id=file.split(".")[0] #ID of each file
        file = open( 'bytefiles/' + file)
        corpus_byte_codes=[file.read().replace('\n', ' ').lower()]
        bigrams_counts = vectorizer.transform(corpus_byte_codes)

        row = scipy.sparse.csr_matrix(bigrams_counts).toarray() \

        byte_file_bigram_df.write('\n')

        file.close()
```

```
10868it [3:35:28,  1.19s/it]

CPU times: user 3h 15min 2s, sys: 16min 21s, total: 3h 31min 24s
Wall time: 3h 35min 29s
```

In [89]:
```
byte_file_bigram_df
```

Out[89]:
```
<_io.TextIOWrapper name='featurization/byte_files_bigram_df.csv' mode='w' encoding='UTF-8'>
```

In [92]:
```
%%time

# HERE I  SELECTING THE ONLY 7000 ROWS BECAUSE OF LACK OF RAM
X_byte_bigram_all_df = pd.read_csv("featurization/byte_files_bigram_df.csv" ,nrows =7000)
X_byte_bigram_all_df.head(2)
```

```
CPU times: user 9min 25s, sys: 53.8 s, total: 10min 18s
Wall time: 9min 56s
```

Out[92]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	...	?? f7	?? f8	?? f9	?? fa	?? fb	?? fc	?? fd	?? fe	?? ff	?? ??
0	frLD0F7HvGAwkdWKsNYU	2228	10	7	12	5	10	12	16	13	...	0	0	0	0	0	0	0	0	0	1453584
1	A41Ov2nFcx8flziRwrap	115514	314	107	161	170	33	21	61	66	...	0	0	0	0	0	0	0	0	0	10314

2 rows × 66050 columns

In [117]:
```
#HERE I M TAKING THE ONLY 10K COLUMNS FROM 66K COLUMNS BECAUSE OF LACK OF COMPUTATIONAL POWER
X_byte_bigram_all_df_2 =X_byte_bigram_all_df.iloc[:, 0:10000]
X_byte_bigram_all_df_1 = X_byte_bigram_all_df_2.drop(columns=['ID'])
X_byte_bigram_all_df_1.head()
```

Out[117]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	...	26 df	26 e0	26 e1	26 e2	26 e3	26 e4	26 e5	26 e6	26 e7	26 e8
0	2228	10	7	12	5	10	12	16	13	5	...	10	6	14	10	17	8	9	12	8	13
1	115514	314	107	161	170	33	21	61	66	100	...	0	0	0	0	1	0	1	0	0	4
2	5708	38	21	77	18	57	19	36	24	46	...	11	9	8	18	13	4	16	8	15	14
3	13715	315	59	95	88	33	12	45	52	50	...	1	2	2	3	1	1	1	0	2	2
4	6517	38	24	72	16	61	15	33	20	48	...	10	6	18	8	11	15	6	13	9	9

5 rows × 9999 columns

HERE WE ARE SELECTING THE MOST IMP FEATURE FROM 10K FEATURES THAT WE HAVE SELECTED

In [259]:
```
X = X_byte_bigram_all_df_1.iloc[:,0:20]  #independent columns
y = X_byte_bigram_all_df_1.iloc[:,-1]    #target column i.e price range
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.05887941 0.05660353 0.04895126 0.04746991 0.04895614 0.0480042
 0.04755586 0.04322714 0.06096193 0.05235797 0.04406624 0.04453162
 0.04662593 0.04887868 0.04594739 0.05086521 0.05538383 0.04742639
 0.0568931  0.04641428]
```

In [260]:
```
X = X_byte_bigram_all_df_1.iloc[:,0:20]  #independent columns
y = X_byte_bigram_all_df_1.iloc[:,-1]    #target column i.e price range
#get correlations of each features in dataset
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

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

In [263]:
```
result_y.shape
# here we have taken only 7k features because of lack of ram
A =result_y.loc[1:7000]
A.shape
```

Out[263]:
```
(7000,)
```

#1.2 Prepare, Merge and Split data

In [111]:
```
#  HERE WE ARE SPILLITING THE DATASET

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

In [112]:
```
X_train.shape, y_train.shape
```

Out[112]:
```
((5600, 299), (5600,))
```

In [113]:
```
result_x.shape
```

Out[113]:
```
(7000, 299)
```

Random Forest Classifier on Bigram of ByteFiles

HERE WE ARE IMPLEMENTING THE RANDOM FOREST CLASSIFIER

In [114]:
```
%%time
plt.close()

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

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 1.883143428122486
log_loss for c =  50 is 1.8844841941137938
log_loss for c =  100 is 1.884404505826144
log_loss for c =  500 is 1.8846693945692607
log_loss for c =  1000 is 1.8840082712911408
log_loss for c =  2000 is 1.883696771088103
log_loss for c =  3000 is 1.88371524231195

<IPython.core.display.Javascript object>


CPU times: user 11min 41s, sys: 27.1 s, total: 12min 8s
Wall time: 13min 27s
```

RANDOM FOREST CLASSIFIER TUNNING FOR BEST HYPERPARAMETER

In [115]:
```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  10 The train log loss is: 1.9309605188891892
For values of best alpha =  10 The cross validation log loss is: 1.883143428122486
For values of best alpha =  10 The test log loss is: 1.8842643312963245
CPU times: user 1.88 s, sys: 608 ms, total: 2.49 s
Wall time: 3.47 s
```

In [116]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
Number of misclassified points  71.64285714285714
---------------------------------------------------- Confusion matrix ----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>

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

In [264]:
```
X_byte_bigram_all_df.columns
```

Out[264]:
```
Index(['ID', '00 00', '00 01', '00 02', '00 03', '00 04', '00 05', '00 06',
       '00 07', '00 08',
       ...
       '?? f7', '?? f8', '?? f9', '?? fa', '?? fb', '?? fc', '?? fd', '?? fe',
       '?? ff', '?? ??'],
      dtype='object', length=66050)
```

In [174]:
```
A =class_labels.loc[1:7000]
A.shape
```

Out[174]:
```
(7000,)
```

In [175]:
```
%time


from sklearn.feature_selection import SelectKBest, chi2, f_regression
select_kbest_object = SelectKBest(score_func=chi2, k=2000)
# HERE WE ARE SELECTNG THE MOST IMP ONLY 2000 FEATURES
most_imp_features_byte_bigram = select_kbest_object.fit(X_byte_bigram_all_df.drop("ID", axis=1), A)
most_imp_byte_bigram_feature_score_df = pd.DataFrame(most_imp_features_byte_bigram.scores_)
# Creating a df from all the column names from the original full X_byte_bigram_all_df df
most_imp_byte_bigram_columns_df = pd.DataFrame(X_byte_bigram_all_df.columns)
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.2 µs
```

In [176]:
```
most_imp_byte_bigram_columns_df
```

Out[176]:

	0
0	ID
1	00 00
2	00 01
3	00 02
4	00 03
...	...
66045	?? fc
66046	?? fd
66047	?? fe
66048	?? ff
66049	?? ??

66050 rows × 1 columns

In [177]:
```
byte_bigram_df_important_feature_score = pd.concat([most_imp_byte_bigram_columns_df, most_imp_byte_bigram_feature_score_df],axis=1)

byte_bigram_df_important_feature_score.columns = ["Byte Bigram Top 2000 Feature Names","Byte Bigram Top 2000 Feature Score"]

#HERE WE WILL CONCATE THE BOTH THE FEATURES
byte_bigram_df_important_feature_score = byte_bigram_df_important_feature_score.nlargest(10, "Byte Bigram Top 2000 Feature Score")

byte_bigram_df_important_feature_score.head(2)
byte_bigram_df_important_feature_score.shape
```

Out[177]: (10, 2)

In [178]:
```
top_2000_most_imp_byte_bigram_feature_names = list(byte_bigram_df_important_feature_score["Byte Bigram Top 2000 Feature Names"])
top_2000_byte_bigram_features = pd.concat([X_byte_bigram_all_df["ID"], X_byte_bigram_all_df[top_2000_most_imp_byte_bigram_feature_names]], axis=1)
top_2000_byte_bigram_features.to_csv("featurization/featurization_final/top_2000_imp_byte_bigram_df.csv",index=None)
print(top_2000_byte_bigram_features.shape)
top_2000_byte_bigram_features.head()
```

```
(7000, 11)
```

Out[178]:

	ID	?? ff	21 bc	bd 20	21 d3	e7 0a	ba 0a	21 90	d4 20	52 0a	0b b9
0	frLD0F7HvGAwkdWKsNYU	0	10	16	18	11	17	12	20	8	11
1	A41Ov2nFcx8flziRwrap	0	0	0	0	1	1	0	2	1	2
2	gA94Fpfcq5lRMvmkKXYJ	0	13	8	10	10	15	15	9	14	9
3	K1MACq57pdQfEsgjclS9	0	0	1	3	0	2	1	0	0	3
4	9toXzFrSARZ0TJGKMbV6	0	10	11	8	12	13	19	10	16	11

In [179]:
```
top_2000_byte_bigram_features.shape
```

Out[179]: (7000, 11)

#39. ASM Unigram - Top 52 Unigram Features from ASM Files - Final Model Training

In [196]:
```
dfasm=pd.read_csv("/content/asmoutputfile.csv")
Y.columns = ['ID', 'Class']
unigram_asm = pd.merge(dfasm, Y, on='ID', how='left')
```

In [197]:
```
B =unigram_asm.drop(['ID','Class'], axis=1)
B.head(2)
```

Out[197]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	:dword	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip
0	19	744	0	127	57	0	323	0	3	0	...	137	18	66	15	43	83	0	17	48	29
1	17	838	0	103	49	0	0	0	3	0	...	130	18	29	48	82	12	0	14	0	20

2 rows × 51 columns

In [182]:
```
unigram_asm_1 = normalize(B)
```

In [183]:
```
unigram_asm_2 = pd.DataFrame(unigram_asm_1)
```

In [184]:
```
# First read the file that was generated above code
# Meaning, the code that ran for around 48 hours as mentioned above.
dfasm=pd.read_csv( "/content/asmoutputfile.csv")

Y.columns = ['ID', 'Class']
# Note, Y is all the Train Labels of 0 to 9 which has been defined earlier as below
# Y = pd.read_csv(root_path + "trainLabels.csv")

unigram_asm = pd.merge(dfasm, Y, on='ID', how='left')
```

In [185]:
```
unigram_asm_2.head()
```

Out[185]:

	0	1	2	3	4	5	6	7	8	9	...	41	42	43	44	45	46	47	48	49	50
0	0.022154	0.867499	0.0	0.148081	0.066462	0.0	0.376616	0.0	0.003498	0.0	...	0.159741	0.020988	0.076956	0.017490	0.050138	0.096777	0.0	0.019822	0.055968	0.033814
1	0.019442	0.958394	0.0	0.117798	0.056040	0.0	0.000000	0.0	0.003431	0.0	...	0.148677	0.020586	0.033166	0.054896	0.093781	0.013724	0.0	0.016011	0.000000	0.022873
2	0.035228	0.884840	0.0	0.103611	0.089106	0.0	0.300472	0.0	0.006217	0.0	...	0.174067	0.026939	0.087033	0.020722	0.138839	0.029011	0.0	0.022794	0.000000	0.018650
3	0.072072	0.962374	0.0	0.182300	0.080551	0.0	0.000000	0.0	0.012719	0.0	...	0.105988	0.025437	0.033916	0.059353	0.029677	0.008479	0.0	0.033916	0.000000	0.025437
4	0.038200	0.903318	0.0	0.132577	0.382000	0.0	0.000000	0.0	0.006741	0.0	...	0.040447	0.026965	0.020224	0.040447	0.065165	0.011235	0.0	0.024718	0.000000	0.024718

5 rows × 51 columns

File Size of ASM Files - Feature Extraction - Final Model Training

HERE WE ARE EXTRACTING THE ASM FEATURES FROM ASM FILE

In [152]:

```
files=os.listdir( 'asmfiles')

filenames=Y['ID'].tolist()

class_y=Y['Class'].tolist()

class_bytes=[]

sizebytes=[]

fnames=[]

for file in tqdm(files):

    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_file_size=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})


asm_file_size.head()
```

```
100%|██████████| 10868/10868 [00:05<00:00, 1983.93it/s]
```

Out[152]:

	ID	size	Class
0	frLD0F7HvGAwkdWKsNYU	4.347168	3
1	A41Ov2nFcx8flziRwrap	4.347168	1
2	gA94Fpfcq5lRMvmkKXYJ	4.347168	3
3	K1MACq57pdQfEsgjclS9	4.347168	1
4	9toXzFrSARZ0TJGKMbV6	4.347168	3

#. Merging ASM Unigram + ASM File Size

In [153]:
```
#HERE WE WILL MERGE BOTH ASM UNIGRAM ANDD ASM FILE SIZE
unigram_asm_feature__with_size=pd.merge(asm_file_size, unigram_asm.drop(columns=["Class"]),on='ID', how='left')
unigram_asm_feature__with_size.to_csv( "featurization/unigram_asm_feature__with_size")
unigram_asm_feature__with_size.head()
```

Out[153]:

	ID	size	Class	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	...	:dword	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip
0	frLD0F7HvGAwkdWKsNYU	4.347168	3	17	2508	0	121	783373	0	658	...	122	16	43	40	34	46	0	12	0	39
1	A41Ov2nFcx8flziRwrap	4.347168	1	19	26477	0	270	4448	0	7450	...	4825	1071	2646	1826	3384	642	0	19	0	71
2	gA94Fpfcq5lRMvmkKXYJ	4.347168	3	17	972	0	183	1029	0	219	...	64	18	146	175	51	92	0	12	0	39
3	K1MACq57pdQfEsgjclS9	4.347168	1	19	1772	0	250	560	0	0	...	316	67	72	92	241	29	0	18	0	58
4	9toXzFrSARZ0TJGKMbV6	4.347168	3	17	1364	0	186	1036	0	218	...	79	47	178	218	81	152	0	12	0	39

5 rows × 54 columns

In [154]:
```
unigram_asm_feature__with_size_2 =unigram_asm_feature__with_size.drop(['ID'], axis=1)
unigram_asm_feature__with_size_2.head()
```

Out[154]:

	size	Class	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	:dword	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip
0	4.347168	3	17	2508	0	121	783373	0	658	0	...	122	16	43	40	34	46	0	12	0	39
1	4.347168	1	19	26477	0	270	4448	0	7450	0	...	4825	1071	2646	1826	3384	642	0	19	0	71
2	4.347168	3	17	972	0	183	1029	0	219	0	...	64	18	146	175	51	92	0	12	0	39
3	4.347168	1	19	1772	0	250	560	0	0	0	...	316	67	72	92	241	29	0	18	0	58
4	4.347168	3	17	1364	0	186	1036	0	218	0	...	79	47	178	218	81	152	0	12	0	39

5 rows × 53 columns

In [155]:
```
#HERE WE ARE NORMALIZING THE DATASET
unigram_asm_feature__with_size_3 = normalize(unigram_asm_feature__with_size_2)
```

In [156]:
```
%%time
import time
import numpy as np
import os
import codecs
import imageio
import array
from datetime import datetime as dt

if not os.path.isdir( "image_file_asm"):
    os.mkdir( "image_file_asm")

asmfile_list=os.listdir("asmfiles/")
```

```
CPU times: user 33.8 ms, sys: 7.95 ms, total: 41.8 ms
Wall time: 62.7 ms
```

#. ASM Files - Convert the ASM files to images.

EXTRACTING ASM IMAGE FEATURES

In [157]:
```
def extract_images_from_text(arr_of_filenames, folder_to_save_generated_images):
    for file_name in tqdm(arr_of_filenames):

      this_file = codecs.open( "asmfiles/" + file_name, 'rb')
      size_of_current_asm_file = os.path.getsize( "asmfiles/"+file_name)

       # WITH SIZE
      width_of_file = 256

      remainder = size_of_current_asm_file % width_of_file

      array_of_image = array.array('B')

      array_of_image.fromfile(this_file, size_of_current_asm_file-remainder)

      this_file.close()

      arr_of_generated_image = np.reshape(array_of_image[:width_of_file * width_of_file], (width_of_file, width_of_file))

      arr_of_generated_image = np.uint8(arr_of_generated_image)

      imageio.imwrite(folder_to_save_generated_images+'/' + file_name.split(".")[0] + '.png', arr_of_generated_image)
directory_to_save_generated_image =  'image_file_asm'

extract_images_from_text(asmfile_list, directory_to_save_generated_image)
```

```
100%|██████████| 10868/10868 [07:06<00:00, 25.48it/s]
```

here we will extract top 800 image features

In [268]:
```
file_list_asm_files=os.listdir( 'image_file_asm/')

with open( "featurization/top_800_image_asm_df.csv", mode='w') as top_800_image_asm_df: #file_list_asm_files = 10868, top_800_image_asm_df=800
    top_800_image_asm_df.write(','.join(map(str, ["ID"]+["pixel_asm{}".format(i) for i in range(10)])))
    top_800_image_asm_df.write('\n')

    for image in tqdm(file_list_asm_files):
        file_id_asm_files=image.split(".")[0]

        asm_image_array=imageio.imread( "image_file_asm/"+image)

        asm_image_array=asm_image_array.flatten()[:10]
        top_800_image_asm_df.write(','.join(map(str, [file_id_asm_files]+list(asm_image_array))))
        top_800_image_asm_df.write('\n')
```

```
100%|██████████| 10868/10868 [00:26<00:00, 409.32it/s]
```

In [207]:
```
%%time

data=pd.read_csv("featurization/top_800_image_asm_df.csv")
data.shape
data.head(2)
```

```
CPU times: user 24.7 ms, sys: 985 µs, total: 25.7 ms
Wall time: 33.1 ms
```

Out[207]:

	ID	pixel_asm0	pixel_asm1	pixel_asm2	pixel_asm3	pixel_asm4	pixel_asm5	pixel_asm6	pixel_asm7	pixel_asm8	pixel_asm9
0	FSMegdBp4jGDmah6vCsr	48	48	52	48	49	48	48	48	32	67
1	fsID2jBe0w5tbhr7QF8G	48	48	52	48	49	48	48	48	32	56

In [213]:
```
data.shape
```

Out[213]:
```
(10868, 11)
```

In [187]:
```
labels = pd.read_csv('/content/trainLabels (1).csv')
```

In [214]:
```
data.reset_index(drop=True, inplace=True)
```

In [188]:
```
labels.head()
```

Out[188]:

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

HERE WE ARE SORTING THE DATASET

In [208]:
```
sorted_train_data = data.sort_values(by='ID', axis=0, ascending=True, inplace=False)
sorted_train_labels = labels.sort_values(by='Id', axis=0, ascending=True, inplace=False)
X = sorted_train_data.iloc[:,1:]
y = np.array(sorted_train_labels.iloc[:,1])
```

Type *Markdown* and LaTeX: α^2

In [166]:
```
X.shape, y.shape
```

Out[166]:
```
((7000, 300), (7000,))
```

Selecting top 50% features who has most variance

In [167]:
```
fsp = SelectPercentile(chi2, percentile=50)
X_new_50 = fsp.fit_transform(X, y)
X_new_50.shape
```

Out[167]:
```
(7000, 150)
```

In [168]:
```
selected_names = fsp.get_support(indices=True)
selected_names = selected_names + 1
selected_names
```

Out[168]:
```
array([  1,   2,   3,   4,   5,   6,   7,   9,  12,  13,  14,  15,  16,
        17,  19,  21,  23,  25,  26,  33,  37,  39,  41,  45,  46,  47,
        49,  51,  52,  57,  58,  60,  62,  65,  66,  67,  68,  69,  70,
        71,  73,  74,  76,  78,  81,  82,  84,  85,  86,  87,  88,  90,
        91,  93,  94,  95, 105, 107, 108, 109, 114, 117, 118, 120, 123,
       125, 126, 127, 129, 131, 132, 133, 134, 137, 138, 140, 141, 142,
       143, 144, 145, 146, 149, 150, 151, 154, 156, 157, 159, 162, 163,
       166, 167, 168, 173, 175, 177, 178, 179, 181, 182, 183, 184, 186,
       188, 189, 193, 194, 196, 197, 199, 200, 201, 203, 204, 205, 208,
       214, 217, 218, 220, 222, 228, 232, 233, 234, 236, 237, 240, 241,
       242, 244, 246, 249, 252, 253, 254, 255, 256, 258, 259, 260, 261,
       262, 264, 280, 289, 290, 296, 298])
```

In [218]:
```
data_reduced = pd.read_csv('featurization/top_800_image_asm_df.csv')
data_reduced.shape
```

Out[218]:
```
(10868, 11)
```

In [219]:
```
data_reduced.rename(columns={'filename': 'ID'}, inplace=True)
```

In [220]:
```
data_reduced.shape
```

Out[220]:
```
(10868, 11)
```

In [221]:
```
data_reduced.head()
```

Out[221]:

	ID	pixel_asm0	pixel_asm1	pixel_asm2	pixel_asm3	pixel_asm4	pixel_asm5	pixel_asm6	pixel_asm7	pixel_asm8	pixel_asm9
0	FSMegdBp4jGDmah6vCsr	48	48	52	48	49	48	48	48	32	67
1	fslD2jBe0w5tbhr7QF8G	48	48	52	48	49	48	48	48	32	56
2	6nCl3JdMAkKjXOgoN8U4	48	48	52	48	49	48	48	48	32	49
3	C0WhARs5qPD8xjczMgVF	48	48	52	48	49	48	48	48	32	56
4	64vsfK12yBanCNdexz35	48	48	52	48	49	48	48	48	32	51

#Implement asm image features + bytes uni-gram features #Merge asm image features + bytes uni-gram features

In [222]:
```
result_x = pd.merge(result.drop('size', axis=1), data_reduced,on='ID', how='left')
result_y = result_x['Class']
# result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x = result_x.drop(['ID', 'Class'], axis=1)
result_x.head()
```

Out[222]:

	0	1	2	3	4	5	6	7	8	9	...	pixel_asm0	pixel_asm1	pixel_asm2	pixel_asm3	pixel_asm4	pixel_asm5	pixel_asm6	pixel_asm7
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	48	48	52	48	49	48	48	48
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	48	48	52	48	49	48	48	48
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	48	48	52	48	49	48	48	48
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	49	48	48	48	49	48	48	48
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	48	48	52	48	49	48	48	48

5 rows × 267 columns

In [223]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

Random Forest Classifier asm image features + bytes uni-gram features

In [224]:
```
%%time
plt.close()


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

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 0.1055463503206448
log_loss for c =  50 is 0.09247793538613362
log_loss for c =  100 is 0.09092670716079057
log_loss for c =  500 is 0.08773710689601785
log_loss for c =  1000 is 0.08772795026262369
log_loss for c =  2000 is 0.0876599088284676
log_loss for c =  3000 is 0.08791606589382094

<IPython.core.display.Javascript object>


CPU times: user 11min 42s, sys: 15.4 s, total: 11min 57s
Wall time: 13min 46s
```

Random Forest Classifier (Best Hypher Parameters) asm image features + bytes uni-gram features

In [225]:
```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  2000 The train log loss is: 0.024462908572078182
For values of best alpha =  2000 The cross validation log loss is: 0.0876599088284676
For values of best alpha =  2000 The test log loss is: 0.07624852412614508
CPU times: user 3min 58s, sys: 7.2 s, total: 4min 5s
Wall time: 4min 17s
```

In [226]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
Number of misclassified points  1.7479300827966882
---------------------------------------------------- Confusion matrix -----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>

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

#Random Forest Classifier (Best Hypher Parameters) asm image features + bytes uni-gram features

In [266]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', 'Number of Misclassified Points']
table.add_row(['Random Forest Classifier', 10, 0.024462908572078182, 0.0876599088284676,0.07624852412614508, 1.7479300827966882])
print(table)
```

```
+--------------------------+---------------------+----------------------+--------------------+--------------------+--------------------------------+
|          Model           | Best HyperParameter |    Train Log Loss    |    CV Log Loss     |    Test Log Loss   | Number of Misclassified Points |
+--------------------------+---------------------+----------------------+--------------------+--------------------+--------------------------------+
| Random Forest Classifier |          10         | 0.024462908572078182 | 0.0876599088284676 | 0.07624852412614508 |       1.7479300827966882       |
+--------------------------+---------------------+----------------------+--------------------+--------------------+--------------------------------+
```

Implement asm unigram + asm extracted image features

Merge asm unigram + asm extracted image features

In [227]:
```
print(data_reduced.shape)
print(result_asm.shape)
```

```
(10868, 11)
(10868, 53)
```

In [228]:
```
result_asm.columns
```

Out[228]:
```
Index(['ID', 'HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:',
       '.rdata:', '.edata:', '.rsrc:', '.tls:', '.reloc:', '.BSS:', '.CODE',
       'jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
       'dec', 'add', 'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror',
       'rol', 'jnb', 'jz', 'rtn', 'lea', 'movzx', '.dll', 'std::', ':dword',
       'edx', 'esi', 'eax', 'ebx', 'ecx', 'edi', 'ebp', 'esp', 'eip', 'Class'],
      dtype='object')
```

In [229]:
```
data_reduced.columns
```

Out[229]:
```
Index(['ID', 'pixel_asm0', 'pixel_asm1', 'pixel_asm2', 'pixel_asm3',
       'pixel_asm4', 'pixel_asm5', 'pixel_asm6', 'pixel_asm7', 'pixel_asm8',
       'pixel_asm9'],
      dtype='object')
```

In [230]:
```
result_x = pd.merge(result_asm, data_reduced,on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[230]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	pixel_asm0	pixel_asm1	pixel_asm2	pixel_asm3	pixel_asm4	pixel_asm5	pixel_asm6	pixel_asm7	pixel_asm8	pixel_asm9
0	19	744	0	127	57	0	323	0	3	0	...	49	48	48	48	49	48	48	48	32	54
1	17	838	0	103	49	0	0	0	3	0	...	54	67	53	67	49	48	48	48	32	63
2	17	427	0	50	43	0	145	0	3	0	...	49	48	48	48	49	48	48	48	32	53
3	17	227	0	43	19	0	0	0	3	0	...	49	49	48	70	49	48	48	48	32	63
4	17	402	0	59	170	0	0	0	3	0	...	55	54	51	53	49	48	48	48	32	63

5 rows × 58 columns

In [231]:
```
result_x.columns
```

Out[231]:
```
Index(['HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:', '.rdata:',
       '.edata:', '.rsrc:', '.tls:', '.reloc:', 'jmp', 'mov', 'retf', 'push',
       'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add', 'imul', 'xchg',
       'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb', 'jz', 'lea',
       'movzx', '.dll', 'std::', ':dword', 'edx', 'esi', 'eax', 'ebx', 'ecx',
       'edi', 'ebp', 'esp', 'eip', 'pixel_asm0', 'pixel_asm1', 'pixel_asm2',
       'pixel_asm3', 'pixel_asm4', 'pixel_asm5', 'pixel_asm6', 'pixel_asm7',
       'pixel_asm8', 'pixel_asm9'],
      dtype='object')
```

In [232]:
```
print(result_x.shape)
```

```
(10868, 58)
```

In [233]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

Random Forest Classifier with asm unigram + asm extracted image features

In [234]:
```
%%time
plt.close()


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

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 0.0330841610036583
log_loss for c =  50 is 0.030099098384935512
log_loss for c =  100 is 0.029989554749910257
log_loss for c =  500 is 0.029887089025185713
log_loss for c =  1000 is 0.03025560011927438
log_loss for c =  2000 is 0.030279907281670758
log_loss for c =  3000 is 0.0302672368653407

<IPython.core.display.Javascript object>


CPU times: user 3min 12s, sys: 13.6 s, total: 3min 26s
Wall time: 3min 46s
```

In [235]:
```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  500 The train log loss is: 0.017855207524103286
For values of best alpha =  500 The cross validation log loss is: 0.029887089025185713
For values of best alpha =  500 The test log loss is: 0.031583841657828536
CPU times: user 22.9 s, sys: 2.01 s, total: 24.9 s
Wall time: 22.8 s
```

In [236]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
Number of misclassified points  0.45998160073597055
---------------------------------------------------- Confusion matrix -----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>

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

#2.3.5 Conclusion and Model Comparision (asm unigram + asm extracted image features)

In [257]:
```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', 'Number of Misclassified Points']
table.add_row(['Random Forest Classifier', 1000, 0.017855207524103286, 0.029887089025185713, 0.031583841657828536, 0.45998160073597055])
print(table)
```

```
+--------------------------+---------------------+----------------------+----------------------+----------------------+--------------------------------+
|          Model           | Best HyperParameter |    Train Log Loss    |     CV Log Loss      |    Test Log Loss     | Number of Misclassified Points |
+--------------------------+---------------------+----------------------+----------------------+----------------------+--------------------------------+
| Random Forest Classifier |         1000        | 0.017855207524103286 | 0.029887089025185713 | 0.031583841657828536 |      0.45998160073597055       |
+--------------------------+---------------------+----------------------+----------------------+----------------------+--------------------------------+
```

Implemented ASM unigram + ASM image features + ByteFile unigram

Prepare, Merge and Split (ASM unigram + ASM image features + ByteFile unigram) data

In [238]:
```
print(data_reduced.shape)
print(result_asm.shape)
print(result.shape)
```

```
(10868, 11)
(10868, 53)
(10868, 260)
```

In [239]:
```
data_reduced.columns
```

Out[239]:
```
Index(['ID', 'pixel_asm0', 'pixel_asm1', 'pixel_asm2', 'pixel_asm3',
       'pixel_asm4', 'pixel_asm5', 'pixel_asm6', 'pixel_asm7', 'pixel_asm8',
       'pixel_asm9'],
      dtype='object')
```

In [240]:
```
result_asm.columns
```

Out[240]:
```
Index(['ID', 'HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:',
       '.rdata:', '.edata:', '.rsrc:', '.tls:', '.reloc:', '.BSS:', '.CODE',
       'jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc',
       'dec', 'add', 'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror',
       'rol', 'jnb', 'jz', 'rtn', 'lea', 'movzx', '.dll', 'std::', ':dword',
       'edx', 'esi', 'eax', 'ebx', 'ecx', 'edi', 'ebp', 'esp', 'eip', 'Class'],
      dtype='object')
```

In [241]:
```
result.columns
```

Out[241]:
```
Index(['ID', '0', '1', '2', '3', '4', '5', '6', '7', '8',
       ...
       'f9', 'fa', 'fb', 'fc', 'fd', 'fe', 'ff', '??', 'size', 'Class'],
      dtype='object', length=260)
```

In [242]:
```
result_y
```

Out[242]:
```
0        1
1        1
2        1
3        1
4        1
        ..
10863    2
10864    2
10865    2
10866    2
10867    2
Name: Class, Length: 10868, dtype: int64
```

In [243]:
```
result_x = pd.merge(result_asm, data_reduced, on='ID', how='left')
result_x = pd.merge(result_x, result.drop('size', axis=1), on='ID', how='left')
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE'], axis=1)
result_x.head()
```

Out[243]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	f8	f9	fa	fb	fc	fd	fe	ff	??	Class_y
0	19	744	0	127	57	0	323	0	3	0	...	0.004728	0.002121	0.001886	0.002272	0.013032	0.002211	0.003957	0.010904	0.000984	1
1	17	838	0	103	49	0	0	0	3	0	...	0.001804	0.001264	0.000972	0.001255	0.004003	0.001333	0.001799	0.004187	0.000006	1
2	17	427	0	50	43	0	145	0	3	0	...	0.002258	0.001194	0.001094	0.003702	0.004244	0.001343	0.002865	0.004556	0.000003	1
3	17	227	0	43	19	0	0	0	3	0	...	0.001766	0.001259	0.000963	0.001250	0.003853	0.001235	0.001654	0.003977	0.000978	1
4	17	402	0	59	170	0	0	0	3	0	...	0.002107	0.001294	0.001061	0.001105	0.004509	0.001268	0.001794	0.004061	0.000834	1

5 rows × 317 columns

In [244]:
```
result_x.shape
```

Out[244]:
```
(10868, 317)
```

In [245]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

Random Forest Classifier ASM unigram + ASM image features + ByteFile unigram

In [246]:
```
%%time
plt.close()


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

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 0.026688825680914637
log_loss for c =  50 is 0.02410699455284527
log_loss for c =  100 is 0.02410929494078561
log_loss for c =  500 is 0.02437965200306683
log_loss for c =  1000 is 0.024135634465117777
log_loss for c =  2000 is 0.024300039558738596
log_loss for c =  3000 is 0.024259185712369975

<IPython.core.display.Javascript object>


CPU times: user 9min 32s, sys: 13.3 s, total: 9min 45s
Wall time: 11min 19s
```

In [247]:
```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  50 The train log loss is: 0.01178359319323112
For values of best alpha =  50 The cross validation log loss is: 0.02410699455284527
For values of best alpha =  50 The test log loss is: 0.025010388332094964
CPU times: user 5.92 s, sys: 326 ms, total: 6.24 s
Wall time: 7.18 s
```

In [248]:
```
plt.close()
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
Number of misclassified points  0.36798528058877644
---------------------------------------------------- Confusion matrix -----------------------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>

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

Conclusion and Model Comparision (ASM unigram + ASM image features + ByteFile unigram)

In [256]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', 'Number of Misclassified Points']
table.add_row(['Random Forest Classifier', 10, 0.01178359319323112, 0.02410699455284527, 0.025010388332094964, 0.36798528058877644])
print(table)
```

```
+-------------------------+--------------------+--------------------+--------------------+--------------------+--------------------------------+
|          Model          | Best HyperParameter |    Train Log Loss   |     CV Log Loss     |    Test Log Loss     | Number of Misclassified Points |
+-------------------------+--------------------+--------------------+--------------------+--------------------+--------------------------------+
| Random Forest Classifier |         10         | 0.01178359319323112 | 0.02410699455284527 | 0.025010388332094964 |      0.36798528058877644       |
+-------------------------+--------------------+--------------------+--------------------+--------------------+--------------------------------+
```

OBSERVATION

AFTER ADDING THE ASM UNIGRAM AND ASM IMAGE FEATURE + BYTEGRAM UNIGRAM WE HAVE REDUCED THE LOG LOSS 0.01

SINCE THERE IS SOME ISSUE IN XG BOOST PREDICT PROBA SO I HAVEN'T DONE THAT IF I USED OLD VERSION OF XGBOOST THEN I NEED TO RESTART NOTEBOOK I HAVE TO WAIT FOR AGAIN I DONT HAVE MORE COMPUTATIONAL UNIT 10 TO 15 HOURS

SINCE I HAVE GOT LOG LOSS 0.01 WITH RANDOM FOREST SO I M NOT GOING FURTHER

PLS NOTE I DONT HAVE HIGH COMPUTATION POWER AND I HAVE DONE THIS ASSIGNMENT WITH COLAB PRO AND MY ALL COMPUTATION UNITS ARE ALMOST OVER AND AS A STUDENET I CANT AFFORD TO BUY GOOGLE PRO AGAIN

IF I HAD MORE COMPUTATIONAL POWER THEN I TAKE MORE ASM FEATUES AND I THINK WE WERE ABLE TO REDUCE THE LOG LOSS MORE

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: