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,    f
.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
.text:0040100D C7 06 08    BB 42 00                           mov     dword ptr [esi],    offset o
.text:00401013 8B C6                                   mov     eax, esi
```

```
.text:00401015 5E                                   pop     esi
.text:00401016 C2 04 00                                   retn    4
.text:00401016                               ; --------------------------------------------------
.text:00401019 CC CC CC    CC CC CC CC                           align 10h
.text:00401020 C7 01 08    BB 42 00                           mov     dword ptr [ecx],    offset o
.text:00401026 E9 26 1C    00 00                               jmp     sub_402C51
.text:00401026                               ; --------------------------------------------------
.text:0040102B CC CC CC    CC CC                               align 10h
.text:00401030 56                                   push    esi
.text:00401031 8B F1                                   mov     esi, ecx
.text:00401033 C7 06 08    BB 42 00                           mov     dword ptr [esi],    offset o
.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
.text:0040104B 83 C4 04                                   add     esp, 4
.text:0040104E
.text:0040104E                               loc_40104E:                   ; CODE XREF: .text:0040
.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 cl
```

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/

https://arxiv.org/pdf/1511.04317.pdf

First place solution in Kaggle competition: https://www.youtube.com/watch?v=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

```
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
import os
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 scipy.sparse import csr_matrix
import joblib
# get execution time for every cell DONT SHOW IT, AS IT COULD MESS PLOTS BEING DISPLAYED
!pip install ipython-autotime
%load_ext autotime
from tqdm import tqdm
```

```
Requirement already satisfied: ipython-autotime in c:\users\chiranjiv\.conda\envs\malware\lib\s
Requirement already satisfied: ipython in c:\users\chiranjiv\.conda\envs\malware\lib\site-packa
Requirement already satisfied: traitlets>=5 in c:\users\chiranjiv\.conda\envs\malware\lib\site-
Requirement already satisfied: matplotlib-inline in c:\users\chiranjiv\.conda\envs\malware\lib\
Requirement already satisfied: stack-data in c:\users\chiranjiv\.conda\envs\malware\lib\site-pa
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in c:\users\chiranj
Requirement already satisfied: backcall in c:\users\chiranjiv\.conda\envs\malware\lib\site-pack
Requirement already satisfied: pygments>=2.4.0 in c:\users\chiranjiv\.conda\envs\malware\lib\si
Requirement already satisfied: setuptools>=18.5 in c:\users\chiranjiv\.conda\envs\malware\lib\s
Requirement already satisfied: colorama in c:\users\chiranjiv\.conda\envs\malware\lib\site-pack
Requirement already satisfied: pickleshare in c:\users\chiranjiv\.conda\envs\malware\lib\site-p
Requirement already satisfied: decorator in c:\users\chiranjiv\.conda\envs\malware\lib\site-pac
Requirement already satisfied: jedi>=0.16 in c:\users\chiranjiv\.conda\envs\malware\lib\site-pa
Requirement already satisfied: parso<0.9.0,>=0.8.0 in c:\users\chiranjiv\.conda\envs\malware\li
Requirement already satisfied: wcwidth in c:\users\chiranjiv\.conda\envs\malware\lib\site-packa
Requirement already satisfied: executing in c:\users\chiranjiv\.conda\envs\malware\lib\site-pac
Requirement already satisfied: pure-eval in c:\users\chiranjiv\.conda\envs\malware\lib\site-pac
Requirement already satisfied: asttokens in c:\users\chiranjiv\.conda\envs\malware\lib\site-pac
Requirement already satisfied: six in c:\users\chiranjiv\.conda\envs\malware\lib\site-packages
time: 16 ms (started: 2022-06-08 21:21:14 +05:30)
```

```
from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
time: 39.8 s (started: 2022-06-07 15:19:17 +00:00)
```

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video) 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 oc


2. if you are having computation power limitations, you can try using google colab, with GPU optio


3. To Extract the .7z file in google cloud, once after you upload the file into server, in your ip
  a. !sudo apt-get install p7zip
  b. !7z x file_name.7z -o path/where/you/want/to/extract


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

```
# !unzip "/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/pickle files/bi_gram
```

```
Archive:  /content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/pickle files/bi
replace bi_gram_data.pkl? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

```
# !unzip "/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_data.z
```

```
Archive:  /content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_c
warning:  stripped absolute path spec from /content/gdrive/My Drive/Assignments AAIC/Assignment
  inflating: content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram
```

```
# # !sudo apt-get install fastjar
# !jar xvf "/content/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_g
```

```
java.util.zip.ZipException: zip END header not found
        at java.base/java.util.zip.ZipFile$Source.zerror(ZipFile.java:1607)
        at java.base/java.util.zip.ZipFile$Source.findEND(ZipFile.java:1497)
        at java.base/java.util.zip.ZipFile$Source.initCEN(ZipFile.java:1504)
        at java.base/java.util.zip.ZipFile$Source.<init>(ZipFile.java:1308)
        at java.base/java.util.zip.ZipFile$Source.get(ZipFile.java:1271)
        at java.base/java.util.zip.ZipFile$CleanableResource.<init>(ZipFile.java:733)
        at java.base/java.util.zip.ZipFile$CleanableResource.get(ZipFile.java:850)
        at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:248)
        at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:177)
        at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:148)
        at jdk.jartool/sun.tools.jar.Main.extract(Main.java:1388)
        at jdk.jartool/sun.tools.jar.Main.run(Main.java:410)
        at jdk.jartool/sun.tools.jar.Main.main(Main.java:1680)
```

```
req_cols= ['ID','Class', 'size'] # https://towardsdatascience.com/%EF%B8%8F-load-the-same-csv-file-1
result = pd.read_csv("/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/result_w
```

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/byte_data', '
  byte_data= joblib.load( files)
```

```
result.head()
```

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

```
%%time
# Now merge Bi-grams BoW and result with size and class features
# https://stackoverflow.com/questions/37697195/how-to-merge-two-data-frames-based-on-particular-colu
bi_gram_data= pd.merge(byte_data, result, on='ID', how='left')
print(f'Half of the Bytes files is:- {len(bi_gram_data)}')
```

```
Half of the Bytes files is:- 5436
```

```
CPU times: user 9.34 s, sys: 6.9 s, total: 16.2 s
Wall time: 20.3 s
```

```
bi_gram_data.head()
```

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	...	?? fa	?? fb
0	3639.0	3.0	0.0	4.0	1.0	0.0	1.0	0.0	2.0	0.0	...	0.0	0.0
1	5683.0	33.0	27.0	71.0	27.0	51.0	29.0	44.0	22.0	45.0	...	0.0	0.0
2	15234.0	58.0	20.0	110.0	7.0	10.0	2.0	6.0	5.0	0.0	...	0.0	0.0
3	6100.0	31.0	25.0	72.0	23.0	60.0	34.0	38.0	14.0	52.0	...	0.0	0.0
4	8796.0	148.0	70.0	31.0	29.0	7.0	8.0	8.0	46.0	26.0	...	0.0	0.0

5 rows × 66052 columns

```
# !unzip "/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_data.p
```

```
Archive:  /content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_c
  End-of-central-directory signature not found.  Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive.  In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
unzip:  cannot find zipfile directory in one of /content/gdrive/My Drive/Assignments AAIC/Assig
        /content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_dat
```

```
cols= list(bi_gram_data.columns) # Get col name of df
print(f'\nTotal cols before removing:-{len(cols)}')
cols_to_norm= [str(x) for x in cols if (str(x) != str('ID') and str(x)!=str('Class') and str(x)!=str
print(f'Columns of df {cols_to_norm[:10]}')
print(f'Total cols after removing:-{len(cols_to_norm)}\n')
```

```
Total cols before removing:-66052
Columns of df ['00 00', '00 01', '00 02', '00 03', '00 04', '00 05', '00 06', '00 07', '00 08',
Total cols after removing:-66049
```

```
# Normalize the data and store it in csv file
from tqdm import tqdm
def normalize(df):
    for index, feature_name in tqdm(enumerate(df.columns)):
        # Don't Normalize id or class or size
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class') and str(feature_name)
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            if (max_value-min_value)!=0:
                # if index<20:
                #     print(feature_name)
                #     print(df[feature_name])
                df[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return df
```

```
 time: 5.1 ms (started: 2022-06-07 15:45:01 +00:00)
```

```
bi_gram_normal = normalize(bi_gram_data)
bi_gram_normal.head()
```

```
5431     939.0
5432     297.0
5433      30.0
5434      15.0
5435      69.0
Name: 00 0f, Length: 5436, dtype: float64
00 10
0          4.0
1         14.0
2          3.0
3         29.0
4        121.0
         ...  
5431    1993.0
5432     549.0
5433      38.0
5434      15.0
5435     397.0
Name: 00 10, Length: 5436, dtype: float64
00 11
0         0.0
1        29.0
2         0.0
3        35.0
4         4.0
        ...  
5431    978.0
5432    292.0
5433     54.0
5434     16.0
5435     12.0
Name: 00 11, Length: 5436, dtype: float64
00 12
0          0.0
1         20.0
2          1.0
3         12.0
4          3.0
         ...  
5431    6203.0
5432     394.0
5433      47.0
5434      13.0
5435     285.0
Name: 00 12, Length: 5436, dtype: float64
00 13
0         0.0
1        31.0
2         4.0
3        37.0
4         2.0
        ...  
5431    526.0
5432    277.0
5433     49.0
5434     25.0
5435      3.0
Name: 00 13, Length: 5436, dtype: float64
66052it [00:47, 1402.68it/s]
```

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00
0	0.001698	0.000253	0.000000	0.000304	0.000086	0.000000	0.000097	0.000000	0.000165	0.0000

1	0.002651	0.002782	0.001853	0.005400	0.002312	0.005483	0.002816	0.005348	0.001811	0.0032
2	0.007107	0.004890	0.001372	0.008366	0.000599	0.001075	0.000194	0.000729	0.000412	0.0000
3	0.002846	0.002614	0.001715	0.005476	0.001969	0.006450	0.003302	0.004618	0.001153	0.0037
4	0.004104	0.012479	0.004803	0.002358	0.002483	0.000753	0.000777	0.000972	0.003787	0.0018

5 rows × 66052 columns

```
feat_arr = bi_gram_normal['00 00'].astype(float).values
print(feat_arr.shape)
print( feat_arr.min(), feat_arr.max())
```

```
(5436,)
0.0 1.0
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_nor
#     joblib.dump(bi_gram_normal, files)
```

```
time: 11.4 s (started: 2022-06-06 16:09:22 +00:00)
```

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_norma
    bi_gram_normal = joblib.load(f)
```

```
time: 21.2 s (started: 2022-06-07 05:55:45 +00:00)
```

```
bi_gram_normal.head()
```

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00
0	0.001698	0.000253	0.000000	0.000304	0.000086	0.000000	0.000097	0.000000	0.000165	0.0000
1	0.002651	0.002782	0.001853	0.005400	0.002312	0.005483	0.002816	0.005348	0.001811	0.0032
2	0.007107	0.004890	0.001372	0.008366	0.000599	0.001075	0.000194	0.000729	0.000412	0.0000
3	0.002846	0.002614	0.001715	0.005476	0.001969	0.006450	0.003302	0.004618	0.001153	0.0037
4	0.004104	0.012479	0.004803	0.002358	0.002483	0.000753	0.000777	0.000972	0.003787	0.0018

5 rows × 66052 columns

```
time: 81.1 ms (started: 2022-06-07 05:56:06 +00:00)
```

```
# Garbage collection so maybe Colab's RAM doesn't gets full
# import gc
# gc.collect()
```

TSNE before Feature Selection

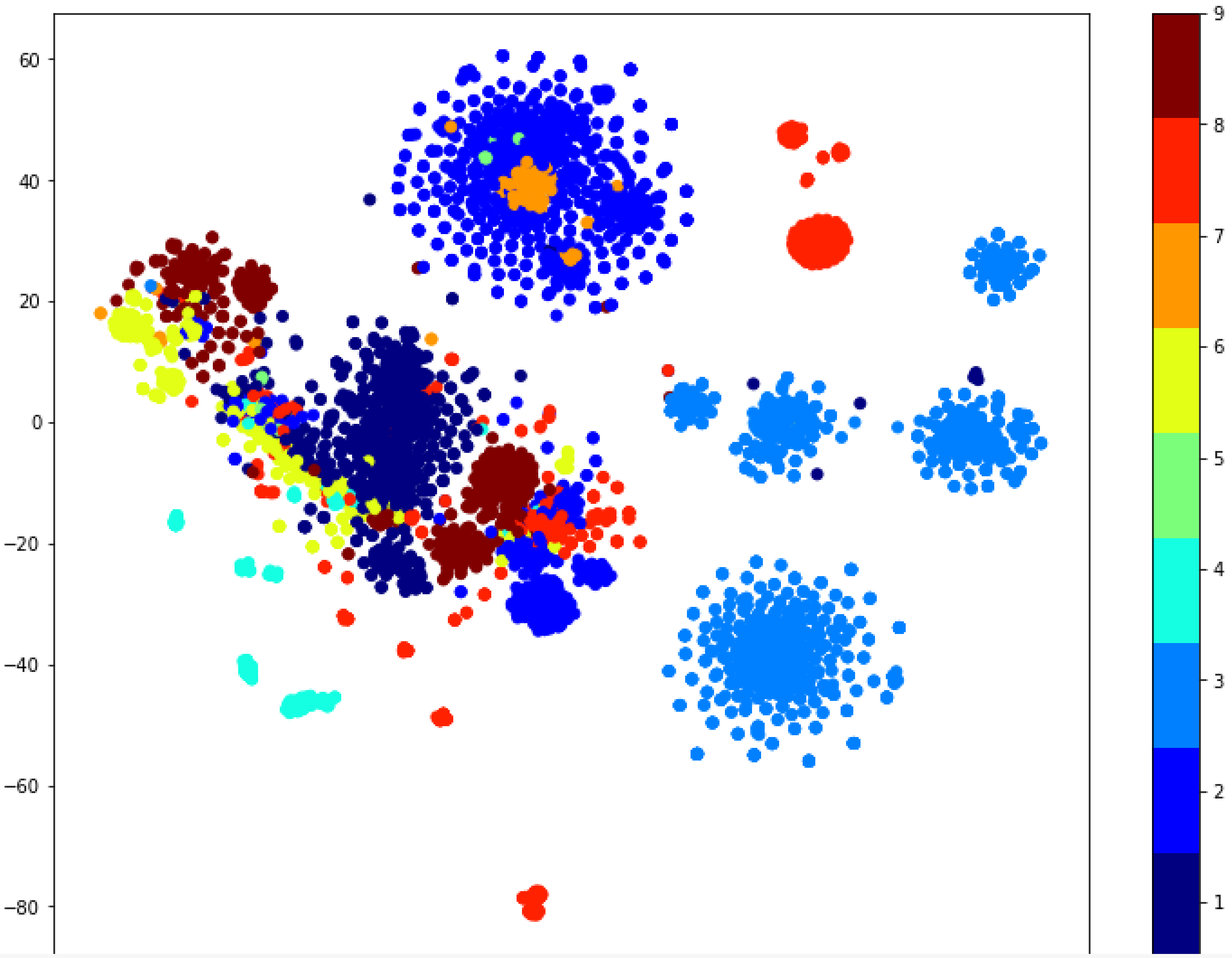
```
cols= list(bi_gram_normal.columns) # Get col name of df
cols_to_norm= [str(x) for x in cols if (str(x) != str('ID') and str(x)!=str('Class') and str(x)!=str
print(f'Columns of df after removing 3 features:-{cols_to_norm[:10]}')
```

```
Columns of df after removing 3 features:-['00 00', '00 01', '00 02', '00 03', '00 04', '00 05',
time: 92 ms (started: 2022-06-06 18:55:59 +00:00)
```

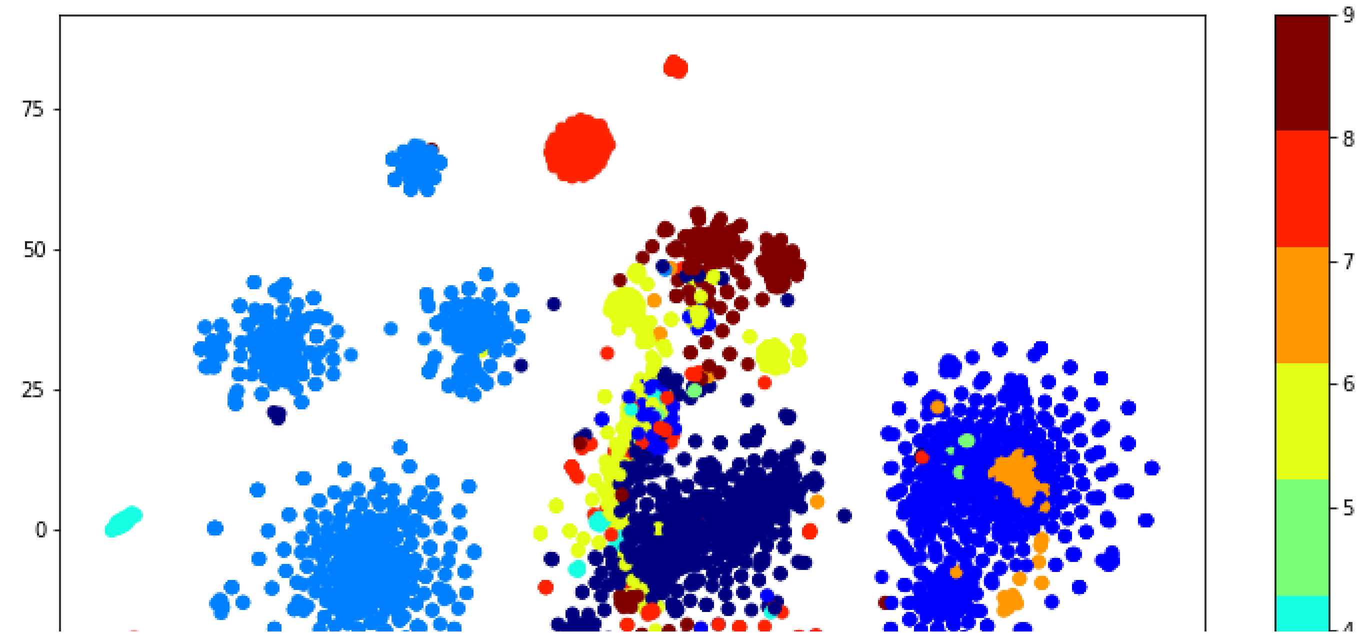
```
class_id= bi_gram_normal['ID']
data_y = bi_gram_normal['Class']
class_size= bi_gram_normal['size']
```

```
time: 9.97 ms (started: 2022-06-06 18:56:03 +00:00)
```

```
# Multiprocess TSNE
from sklearn.manifold import TSNE
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50, n_jobs=2)
results=xtsne.fit_transform(bi_gram_normal.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.figure(figsize=(12.8, 9.6))
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



```
# Multivariant TSNE
from sklearn.manifold import TSNE
#multivariate analysis on byte files
#this is with perplexity 30
xtsne=TSNE(perplexity=30, n_jobs=-1)
results=xtsne.fit_transform(bi_gram_normal.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.figure(figsize=(12.8, 9.6))
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



Feature Selection using Random Forest:-

1. I'm going to try to reduce the No of Features using Random Forest as the Kaggle Winners used it to improve their performance too.

```
# https://mljar.com/blog/feature-importance-in-random-forest/#:~:text=Random%20Forest%20Built%2Din%2
from sklearn.ensemble import RandomForestClassifier

def feature_importance(df, class_y):

    #First Drop ID and Class Columns
    df= df.drop(['ID','Class', 'size'], inplace=False , axis=1)
    rf = RandomForestClassifier(n_estimators= 300, n_jobs=-1,verbose=5)
    rf.fit(df, class_y)
    cols = list(df.columns)

    Sorted_features_imp = np.argsort(rf.feature_importances_)[::-1] # Descending Order of Top featur
    feature_selection = np.take(cols, Sorted_features_imp)

    return feature_selection, rf
```

```
time: 9.72 ms (started: 2022-06-06 18:55:06 +00:00)
```

```
feature_selection, rf= feature_importance(bi_gram_normal, data_y)
```

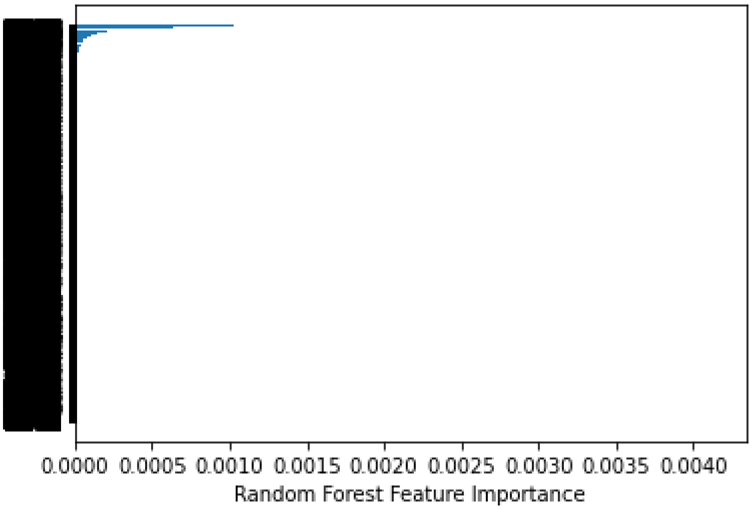
```
building tree 246 of 300
building tree 247 of 300
building tree 248 of 300
building tree 249 of 300
building tree 250 of 300
building tree 251 of 300
building tree 252 of 300
building tree 253 of 300
building tree 254 of 300
building tree 255 of 300

building tree 256 of 300
building tree 257 of 300
building tree 257 of 300
building tree 258 of 300
building tree 259 of 300
building tree 260 of 300
building tree 261 of 300
building tree 262 of 300
building tree 263 of 300
building tree 264 of 300
building tree 265 of 300
building tree 266 of 300
building tree 267 of 300
building tree 268 of 300
building tree 269 of 300
building tree 270 of 300
building tree 271 of 300
building tree 272 of 300
building tree 273 of 300
building tree 274 of 300
building tree 275 of 300
building tree 276 of 300
building tree 277 of 300
building tree 278 of 300
building tree 279 of 300
building tree 280 of 300
building tree 281 of 300
building tree 282 of 300
building tree 283 of 300
building tree 284 of 300
building tree 285 of 300
building tree 286 of 300
building tree 287 of 300
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed:   57.3s
building tree 288 of 300
building tree 289 of 300
building tree 290 of 300
building tree 291 of 300
building tree 292 of 300
building tree 293 of 300
building tree 294 of 300
building tree 295 of 300
building tree 296 of 300
building tree 297 of 300
building tree 298 of 300
building tree 299 of 300
building tree 300 of 300
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  1.0min finished
time: 1min 9s (started: 2022-06-06 18:56:06 +00:00)
```

```
sorted_idx = rf.feature_importances_.argsort() # get sorted index of each feature
top100= list(bi_gram_normal.columns[sorted_idx])[-100:]
print(f'top 100 Features with most Importance:-{top100}')
```

```
top 100 Features with most Importance:-['03 4d', 'ff 83', '09 c7', '00 c7', '85 00', '00 81',
time: 772 ms (started: 2022-06-06 19:02:02 +00:00)
```

```
plt.barh(bi_gram_normal.columns[sorted_idx], rf.feature_importances_[sorted_idx]) # get col name and

plt.xlabel("Random Forest Feature Importance")
```

```
[30441 35142 35143 ... 20117 63130 23212]
Text(0.5, 0, 'Random Forest Feature Importance')
```

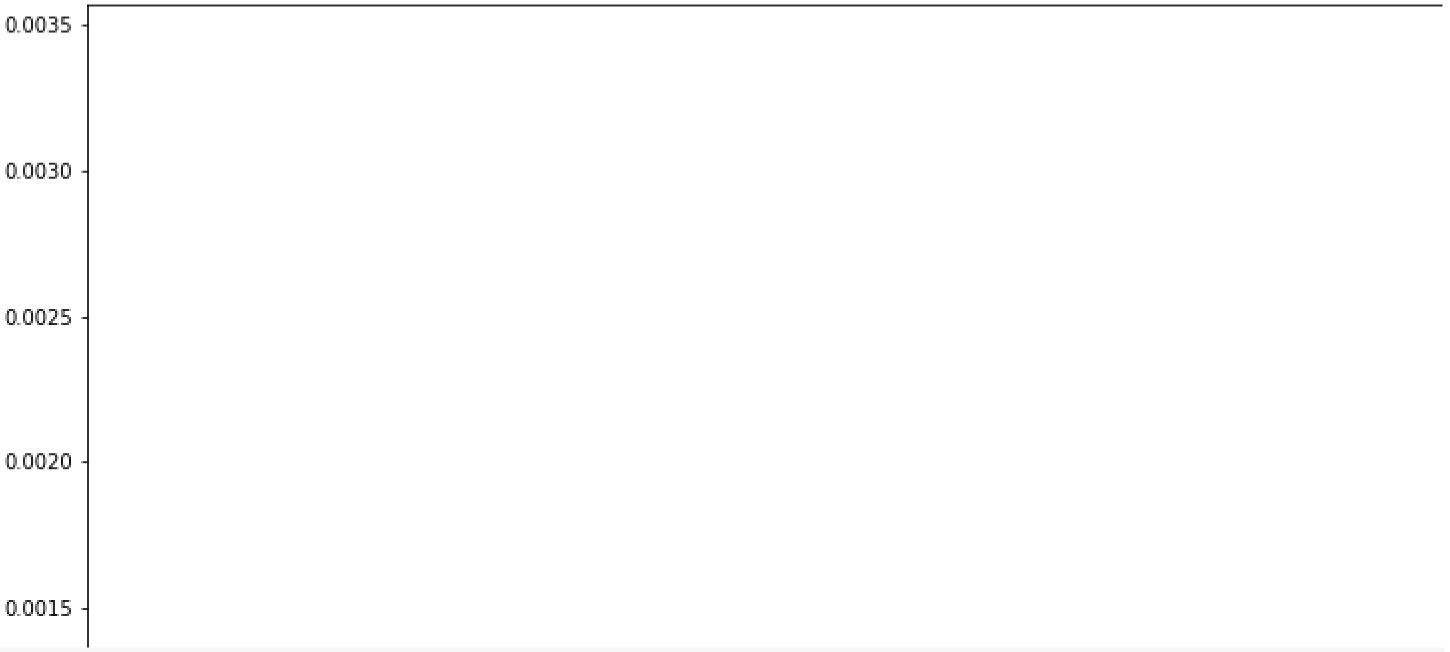


```
time: 12min 36s (started: 2022-06-06 17:30:35 +00:00)
```

```
fea_values= list(rf.feature_importances_[sorted_idx])
print(f'Top 100 feature values:- {fea_values[-100:]}')
```

```
Top 100 feature values:- [0.0009451446670426015, 0.0009454718500374713, 0.0009463196189341168,
time: 612 ms (started: 2022-06-06 19:03:41 +00:00)
```

```
# https://stackoverflow.com/questions/18153054/percentiles-on-x-axis-with-matplotlib
from matplotlib import mlab
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
p = np.array([0.0, 50, 75.0, 90.0, 95.0, 100.0])
perc = np.percentile(fea_values, p)
plt.figure(figsize=(15, 10))
plt.plot(fea_values)
# Place red dots on the percentiles
plt.plot((len(fea_values)-1) * p/100., perc, 'ro')
# Set tick locations and labels
plt.xticks((len(fea_values)-1) * p/100., map(str, p))
plt.show()
```



```
# Now, taking the Top 2 percentile Features
perc = np.percentile(fea_values, 98)
no_of_features= len([lambda x: x for fea in fea_values if float(fea)>perc])
print(f'Taking Top {no_of_features} of features for Modelling')
```

```
Taking Top 1321 of features for Modelling
time: 131 ms (started: 2022-06-06 19:04:15 +00:00)
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/rf_fea_sele
#     joblib.dump(rf, f)
```

```
time: 425 ms (started: 2022-06-06 19:06:41 +00:00)
time: 239 ms (started: 2022-06-06 19:04:02 +00:00)
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/feature_ind
#     joblib.dump(feature_selection, f)
```

```
time: 24.7 ms (started: 2022-06-06 19:07:02 +00:00)
```

Observations:-
1. It seems like very few features are actually non-zero, so using percentile to get no of Top 5 percentile Features. 2. Even in there only top 2 percentile features are most important so taking 1321 no. of features for Byte Files Modelling.

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/rf_fea_select
    rf= joblib.load( f)
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/feature_ind
#     feature_selection=joblib.load( f)
```

```
time: 1.23 ms (started: 2022-06-06 19:15:40 +00:00)
```

```
final_cols = list(feature_selection[:1321]) # Top 1321 bigram features
```

```
final_cols.append('ID')
final_cols.append('Class')
final_cols.append('size')

bi_gram_final = bi_gram_normal[final_cols]
print(f'Shape of the final Dataframe for Byte Files is:-{bi_gram_final.shape}')
```

```
Shape of the final Dataframe for Byte Files is:-(5436, 1324)
time: 42.8 ms (started: 2022-06-06 19:12:23 +00:00)
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_fin
#     joblib.dump(bi_gram_final, f)
```

```
time: 349 ms (started: 2022-06-06 19:12:59 +00:00)
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/data_y' , '
#     joblib.dump(data_y, f)
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/class_id' ,
#     joblib.dump(class_id, f)
```

```
time: 47 ms (started: 2022-06-06 19:20:24 +00:00)
```

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/bi_gram_final
    bi_gram_final= joblib.load(f)
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/data_y' , 'rb
    data_y= joblib.load(f)
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/class_id' , '
    class_id= joblib.load(f)



bi_gram_final.head()
```

	02 c1	93 ed	4e 47	41 44	a8 25	94 86	f5 a8	29 87	f7 8c	9a 1
0	0.000079	0.002886	0.000000	0.000647	0.000000	0.000000	0.000000	0.000000	0.00000	0.0000
1	0.001338	0.011544	0.000655	0.003665	0.015873	0.134058	0.089286	0.016779	0.03794	0.0277
2	0.000000	0.000000	0.000055	0.000431	0.005291	0.000000	0.005952	0.000000	0.00542	0.0019
3	0.001181	0.014430	0.000983	0.003449	0.014109	0.094203	0.107143	0.015101	0.03252	0.0198
4	0.000945	0.000000	0.000109	0.000216	0.000000	0.000000	0.000000	0.000000	0.00000	0.0000

5 rows × 1324 columns

```
time: 1.21 s (started: 2022-06-07 05:57:22 +00:00)
```

TSNE after Feature Selection

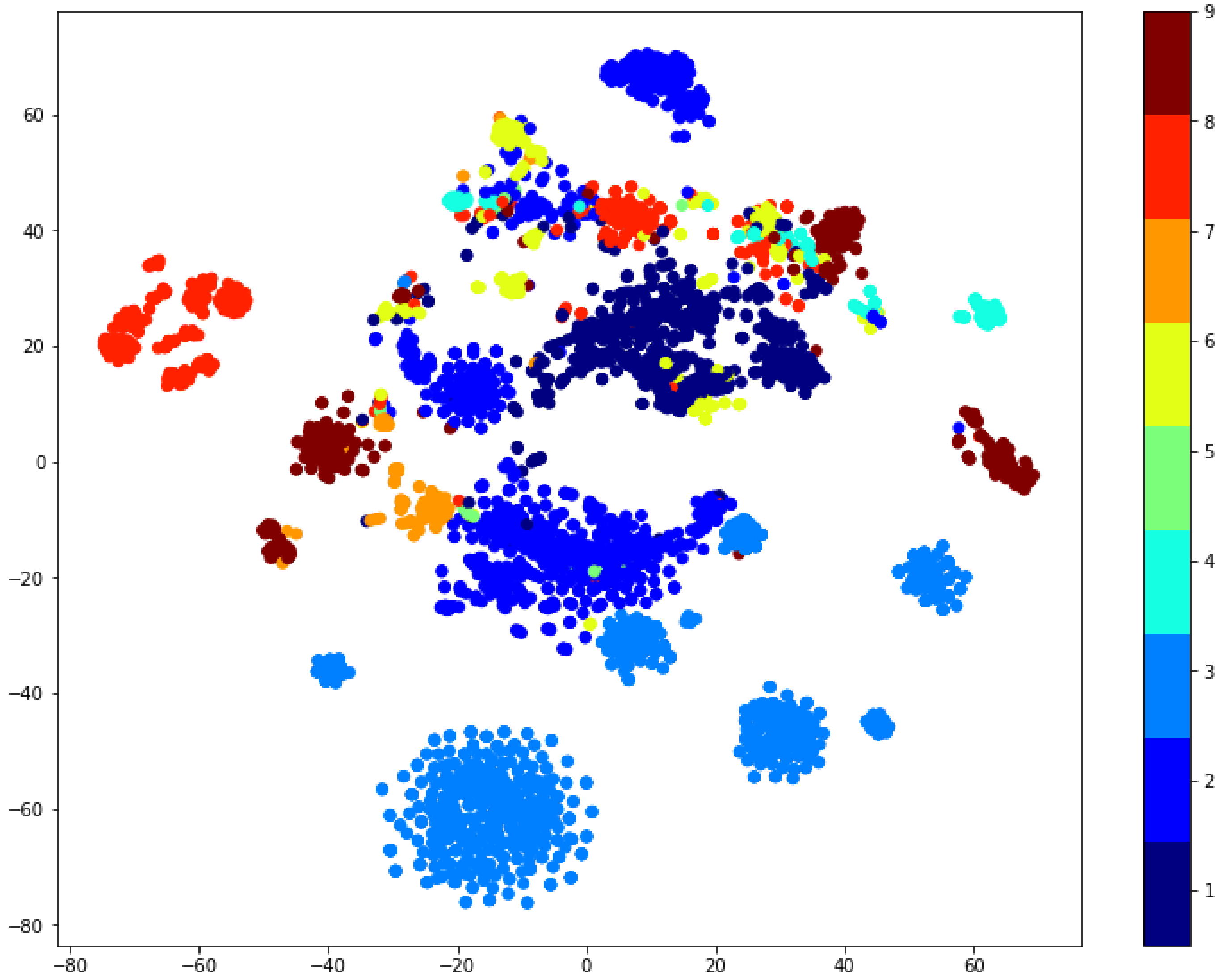
```
cols_final= list(bi_gram_final.columns) # Get col name of df
cols_to_norm_final= [str(x) for x in cols_final if (str(x) != str('ID') and str(x)!=str('Class') and
print(f'Columns of df after removing 3 features:-{cols_to_norm_final[:10]}')
```

```
Columns of df after removing 3 features:-['02 c1', '93 ed', '4e 47', '41 44', 'a8 25', '94 86',
time: 20.4 ms (started: 2022-06-07 10:35:20 +00:00)
```

```
class_id= bi_gram_final['ID']
data_y = bi_gram_final['Class']
class_size= bi_gram_final['size']
```

```
time: 2.22 ms (started: 2022-06-07 10:35:29 +00:00)
```

```
# Multiprocess TSNE
from sklearn.manifold import TSNE
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50, n_jobs=2)
results=xtsne.fit_transform(bi_gram_final.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.figure(figsize=(12.8, 9.6))
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



```
time: 54.7 s (started: 2022-06-07 10:35:44 +00:00)
```

4. Machine Learning Models

4.1. Machine Leaning Models on Bi-grams vectorized bytes files

```
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 diamension
    # 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 diamension
    # 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))
```

```
time: 31.7 ms (started: 2022-06-07 15:32:46 +00:00)
```

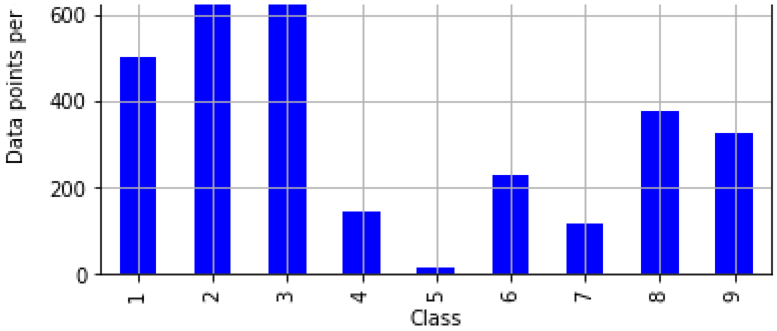
```
# Splitting Data

X_train, X_test, y_train, y_test = train_test_split(bi_gram_final.drop(['ID', 'Class'], axis=1), dat
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.20)
```

```
time: 176 ms (started: 2022-06-07 05:57:48 +00:00)
```

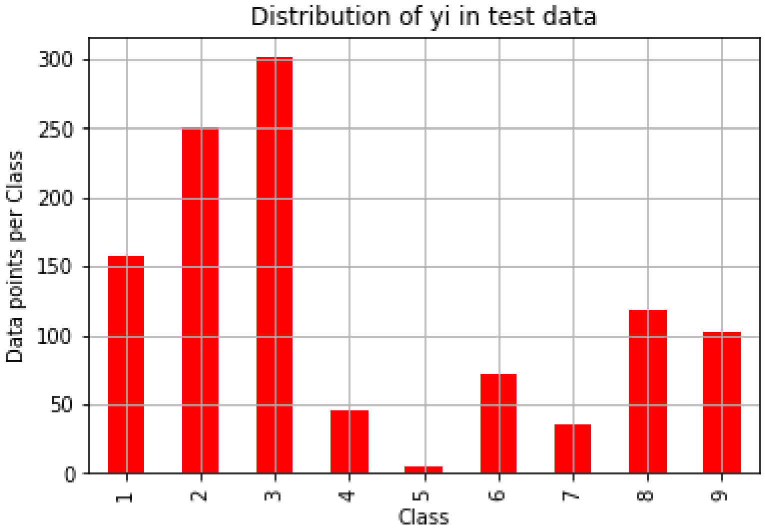
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
Number of data points in train data: 3478
Number of data points in test data: 1088
Number of data points in cross validation data: 870
time: 2.48 ms (started: 2022-06-07 06:14:04 +00:00)
```

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

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

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


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

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

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

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

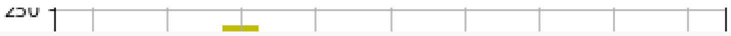


```
Number of data points in class 3 : 961 ( 27.631 %)
Number of data points in class 2 : 801 ( 23.03 %)
Number of data points in class 1 : 504 ( 14.491 %)
Number of data points in class 8 : 378 ( 10.868 %)
Number of data points in class 9 : 326 ( 9.373 %)
Number of data points in class 6 : 230 ( 6.613 %)
Number of data points in class 4 : 146 ( 4.198 %)
Number of data points in class 7 : 117 ( 3.364 %)
Number of data points in class 5 : 15 ( 0.431 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class 3 : 301 ( 27.665 %)
Number of data points in class 2 : 250 ( 22.978 %)
Number of data points in class 1 : 158 ( 14.522 %)
Number of data points in class 8 : 118 ( 10.846 %)
Number of data points in class 9 : 102 ( 9.375 %)
Number of data points in class 6 : 72 ( 6.618 %)
Number of data points in class 4 : 46 ( 4.228 %)
Number of data points in class 7 : 36 ( 3.309 %)
```

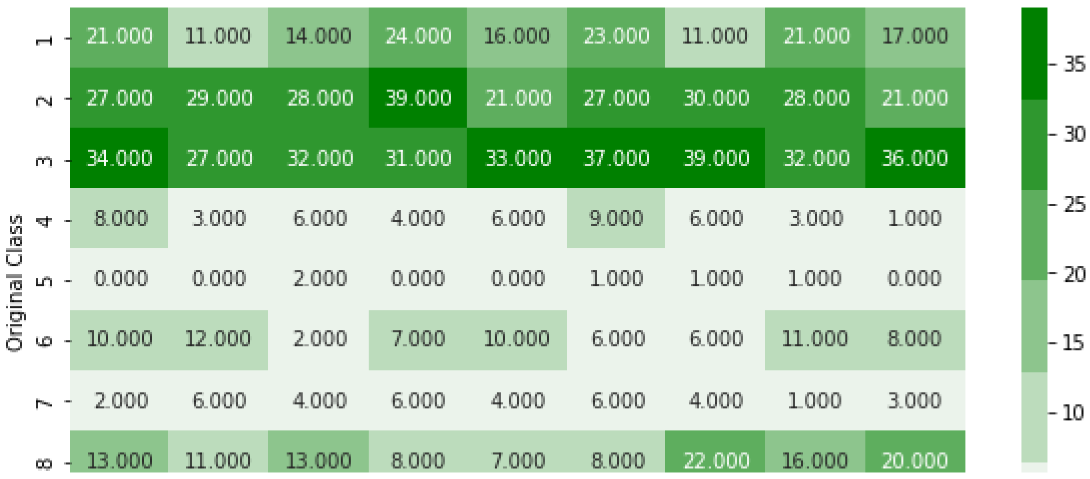
4.1.1. Random Model



```
# 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)
```

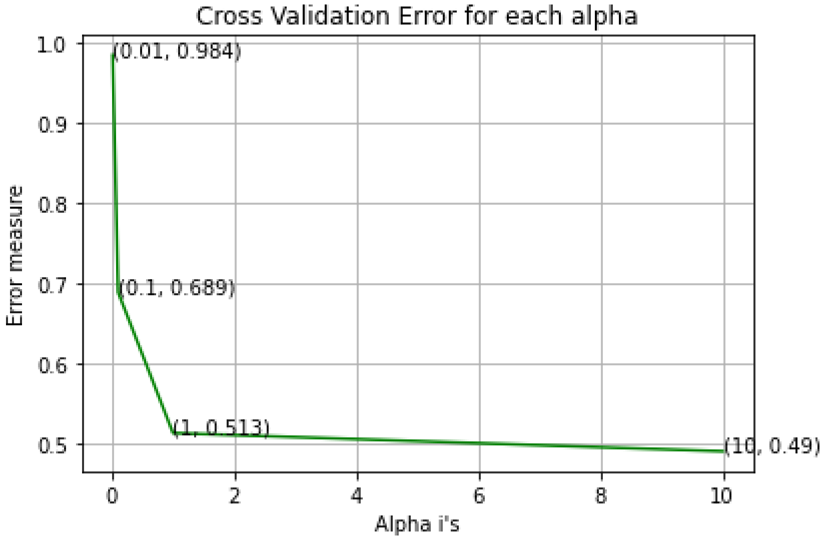
```
Log loss on Cross Validation Data using Random Model 2.482581275703057
Log loss on Test Data using Random Model 2.513005945816048
Number of misclassified points  88.41911764705883
-------------------------------------------------- Confusion matrix ----------------------------
```



4.1.3. Logistic Regression

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=’optimal’, eta0=0
# 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-intu
#------------------------------

alpha = [10 ** x for x in range(-2, 2)] # from 10^-2 to 100
cv_log_error_array=[]
for i in tqdm(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 tqdm(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()
```

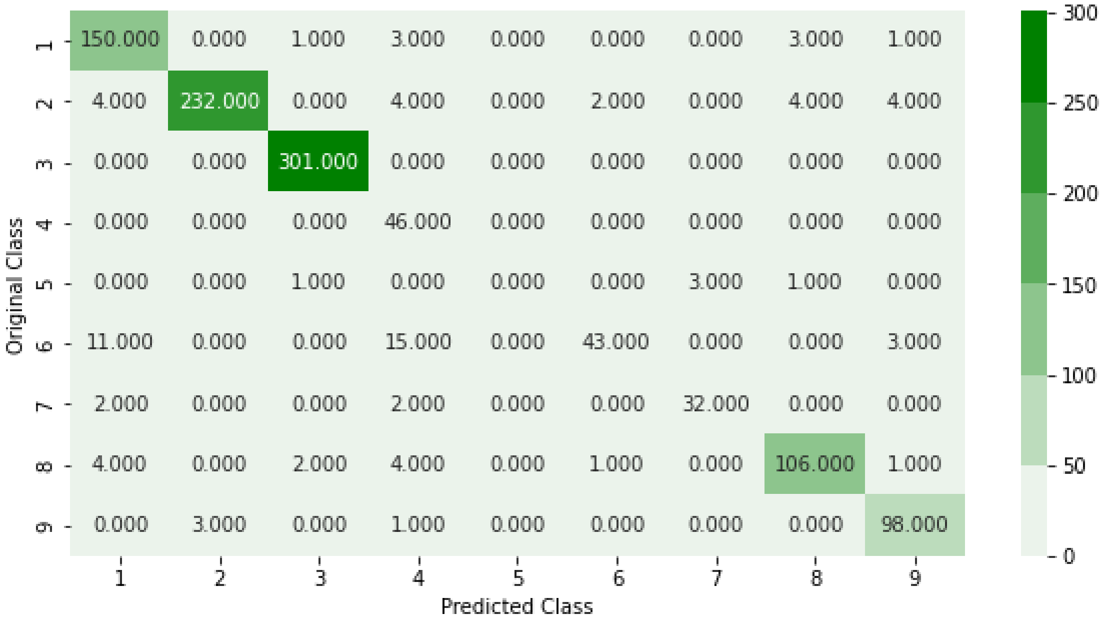
```
100%|██████████| 4/4 [02:37<00:00, 39.41s/it]
log_loss for c =  0.01 is 0.9843354954977341
log_loss for c =  0.1 is 0.6886942921928384
log_loss for c =  1 is 0.5128807376228741
log_loss for c =  10 is 0.49014903800259746
4it [00:00, 5433.04it/s]
```



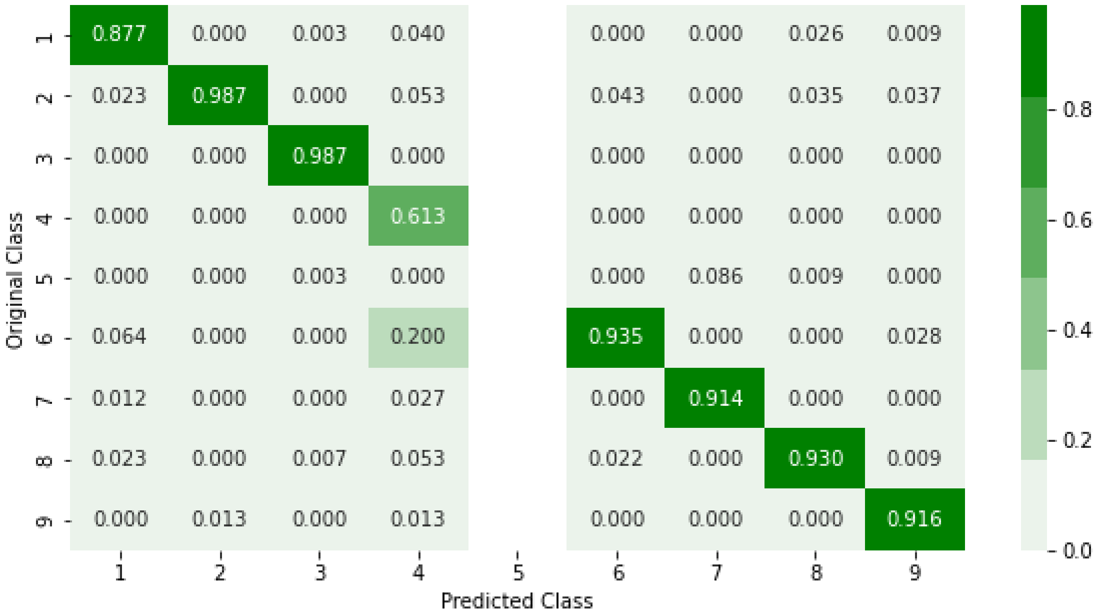
```
time: 2min 37s (started: 2022-06-07 05:57:55 +00:00)
```

```
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))
```

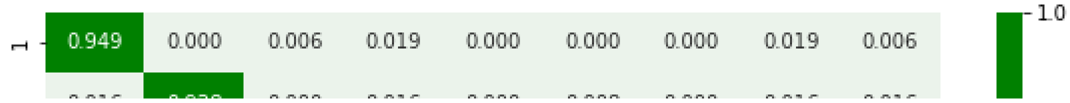
```
log loss for train data 0.42123282440687076
log loss for cv data 0.49014903800259746
log loss for test data 0.4800492258045204
Number of misclassified points  7.352941176470589
-------------------------------------------------- Confusion matrix ----------------------------
```



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



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



Observation:-

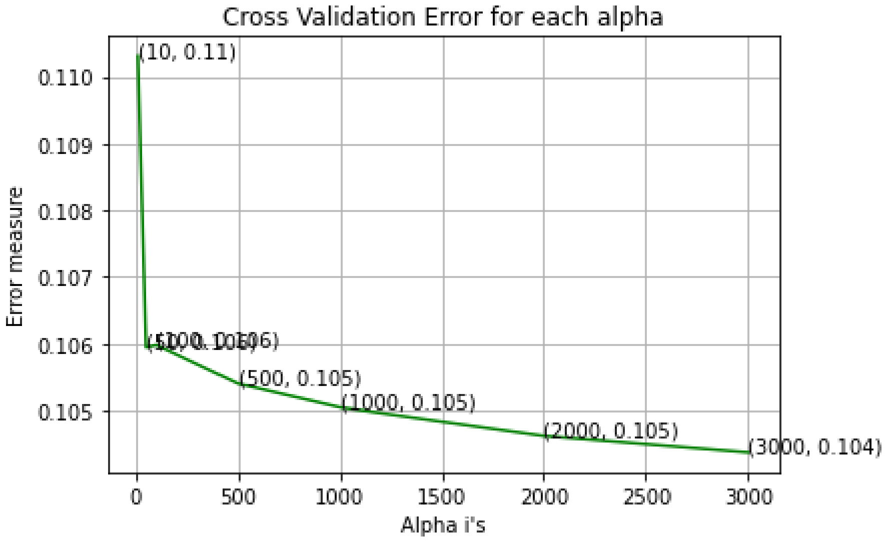
1. Logistic Regression is working better than random Model as even it's worst Loss value is better than Random's Log Loss.
2. But, it simple can't predict even a single datapoint which has Class=5, mostly as the no. of points for Class 5 is very less.

4.1.4. Random Forest Classifier

Predicted Class

```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier

for i in tqdm(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()
```

```
100%|██████████| 7/7 [15:51<00:00, 135.95s/it]
log_loss for c =  10 is 0.1103102243108969
log_loss for c =  50 is 0.10594184107588737
log_loss for c =  100 is 0.10597912479829234
log_loss for c =  500 is 0.10539897923525321
log_loss for c =  1000 is 0.10504481350592867
log_loss for c =  2000 is 0.10462070016956801
log_loss for c =  3000 is 0.10437329055908233
```



```
time: 15min 51s (started: 2022-06-07 06:25:03 +00:00)
```

```
best_alpha= 3000
```

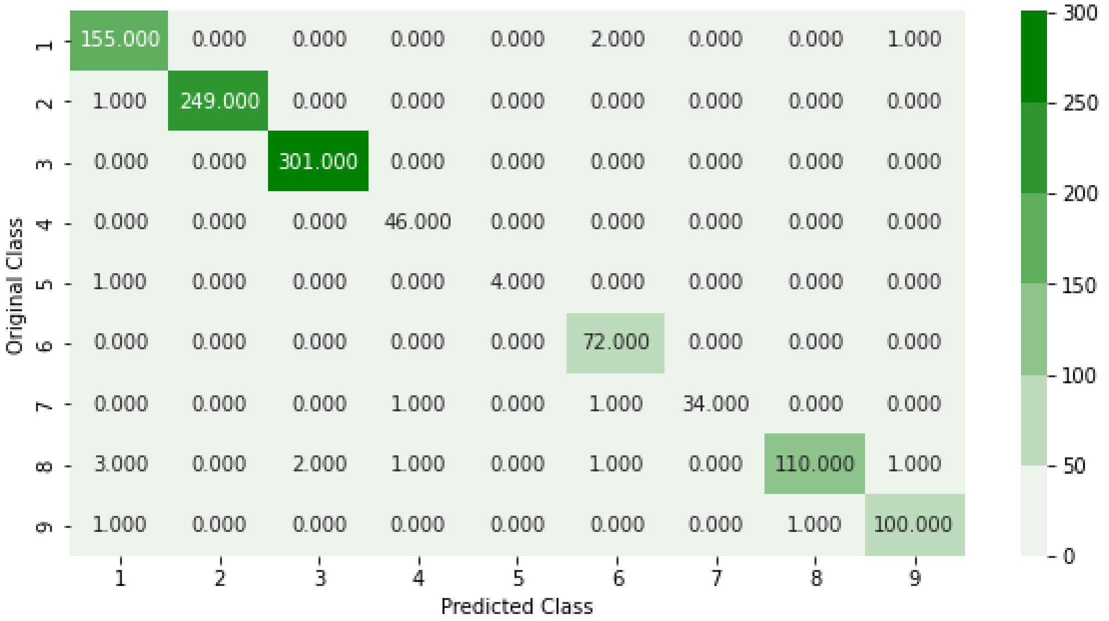
```
time: 4.99 ms (started: 2022-06-07 06:43:57 +00:00)
```

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

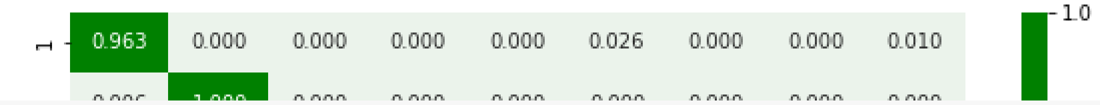
```
CalibratedClassifierCV(base_estimator=RandomForestClassifier(n_estimators=3000,
                                                             n_jobs=-1,
                                                             random_state=42))time: 7min (start
```

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

```
For values of best alpha =  3000 The train log loss is: 0.027513497558377793
For values of best alpha =  3000 The cross validation log loss is: 0.10437329055908233
For values of best alpha =  3000 The test log loss is: 0.07604453520379631
Number of misclassified points  1.5625
-------------------------------------------------- Confusion matrix --------------------------
```



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



```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/sig_clf' ,
#     joblib.dump(sig_clf, f)
```

```
time: 15.3 s (started: 2022-06-07 07:03:01 +00:00)
```

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/sig_clf' , 'r
    sig_clf= joblib.load(f)
```

```
time: 24.7 ms (started: 2022-06-06 19:07:02 +00:00)
```

Observation:-

1. Random Forest is performing better than LR Model.
2. Unlike LR, it can predict 80% datapoint which belong to Class=5 correctly, but, still 20% are misclassified.

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

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

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

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
RandomizedSearchCV(estimator=XGBClassifier(), n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5],
                                        'learning_rate': [0.01, 0.05, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500],
                                        'subsample': [0.1, 0.3, 0.5]},
                   verbose=10)time: 39min 45s (started: 2022-06-07 08:38:25 +00:00)
```

```
xgb_bytes.best_estimator_
```

```
XGBClassifier(colsample_bytree=0.5, learning_rate=0.2, max_depth=10,
              n_estimators=200, objective='multi:softprob', subsample=0.3)time: 7.22 ms (starte
```

```
best_xgb = XGBClassifier(colsample_bytree=0.5, learning_rate=0.2,
                         max_depth=10, n_estimators=200,
                         objective='multi:softprob', subsample=0.3,
                         n_jobs=-1, random_state=4)



best_xgb.fit(X_train,y_train)
```

```
XGBClassifier(colsample_bytree=0.5, learning_rate=0.2, max_depth=10,
              n_estimators=200, n_jobs=-1, objective='multi:softprob',
              random_state=4, subsample=0.3)time: 1min 43s (started: 2022-06-07 09:40:22 +00:0
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/best_xgb' ,
#     joblib.dump(best_xgb, f)
```

```
time: 56.9 ms (started: 2022-06-07 09:46:54 +00:00)
```

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/best_xgb' , '
    best_xgb= joblib.load(f)
```

```
time: 24.7 ms (started: 2022-06-06 19:07:02 +00:00)
```

```
best_n_estimator= 200
```

```
time: 1.11 ms (started: 2022-06-07 09:54:20 +00:00)
```

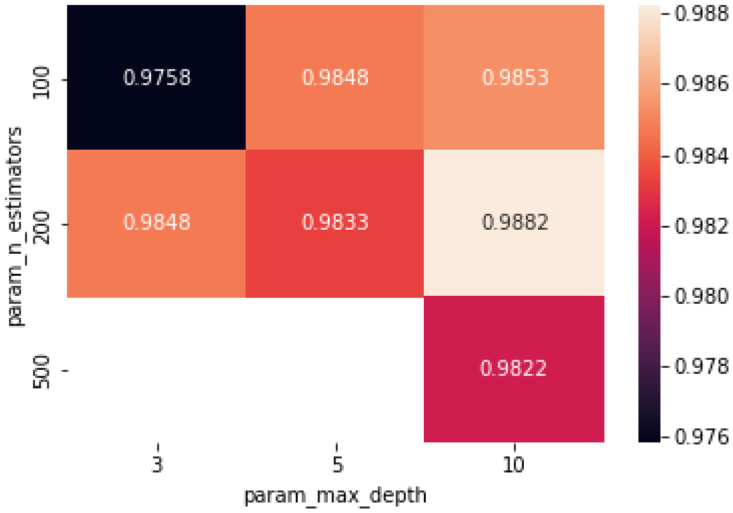
```
predict_y = best_xgb.predict_proba(X_train)
print('For values of best alpha = ', best_n_estimator, "The train log loss is:",log_loss(y_train, pr
predict_y = best_xgb.predict_proba(X_cv)
print('For values of best alpha = ', best_n_estimator, "The cross validation log loss is:",log_loss(
predict_y = best_xgb.predict_proba(X_test)
```

```
print('For values of best alpha = ', best_n_estimator, "The test log loss is:",log_loss(y_test, pred
plot_confusion_matrix(y_test, best_xgb.predict(X_test))
```

```
For values of best alpha =  200 The train log loss is: 0.0037966182536617087
For values of best alpha =  200 The cross validation log loss is: 0.084767432570289860
For values of best alpha =  200 The test log loss is: 0.04844573094156351
Number of misclassified points  1.2867647058823528
-------------------------------------------------- Confusion matrix ----------------------------
```

```
import seaborn as sn
# Ref:- https://stackoverflow.com/questions/56302647/how-to-plot-a-heatmap-and-find-best-hyperparame

results = pd.DataFrame.from_dict(xgb_bytes.cv_results_)
max_scores = results.groupby(['param_n_estimators', 'param_max_depth']).max()
max_scores = max_scores.unstack()[['mean_test_score']]

print(f' {max_scores}')
sn.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
# NOTE:- Some Blocks empty as it's Randomsearch so it didn't fit those combinations
```

```
                    mean_test_score
param_max_depth                   3         5         10
param_n_estimators
100                        0.975846  0.984761  0.985336
200                        0.984761  0.983323  0.988211
500                             NaN       NaN  0.982171
```



```
time: 378 ms (started: 2022-06-07 10:08:48 +00:00)
```

Observation:-

1. XGBoost performed the best out of them all with the least Log Loss of 0.06
2. But, it like RF still simply can't predict Class 5 that well.
3. Biggest disadvantage of XGBoost is the computation time.

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


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
```

4.3 Train and test split

```
result_asm_size= pd.read_csv("/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/asm_wi
result_asm_size.head()
```

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

```
time: 27.7 ms (started: 2022-06-07 15:47:41 +00:00)
```

```
result_asm_uni= pd.read_csv("/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/asmoutp
result_asm_uni.head()
```

```
print(result_asm_uni.shape)
print(result_asm_size.shape)
```

```
(10868, 52)
(10868, 4)
time: 6.34 ms (started: 2022-06-07 15:47:43 +00:00)
```

3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0

```
result_asm = pd.merge(result_asm_uni,result_asm_size.drop(['Unnamed: 0'], axis=1) , on='ID', how='le
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

```
54it [00:00, 634.04it/s]
```

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

5 rows × 54 columns

```
time: 182 ms (started: 2022-06-07 15:48:08 +00:00)
```

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

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...
0	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	0.0	...
1	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	0.0	...
2	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	0.0	...
3	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	0.0	...
4	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	0.0	...

5 rows × 49 columns

```
time: 60.1 ms (started: 2022-06-07 15:48:24 +00:00)
```

```
asm_x.shape
```

```
(10868, 49)time: 4.07 ms (started: 2022-06-07 16:17:26 +00:00)
```

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,test
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y_
```

```
time: 35.9 ms (started: 2022-06-07 15:48:39 +00:00)
```

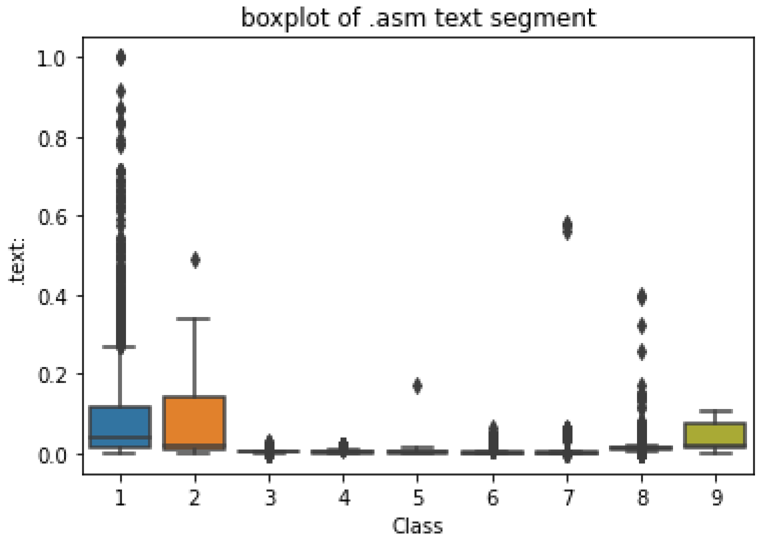
```
print( X_cv_asm.isnull().all())
```

```
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_asm    False
dtype: bool
time: 7.34 ms (started: 2022-06-07 15:48:42 +00:00)
```

4.2.2 Univariate analysis on asm file features

```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
```

```
plt.title("boxplot of .asm text segment")
plt.show()
```

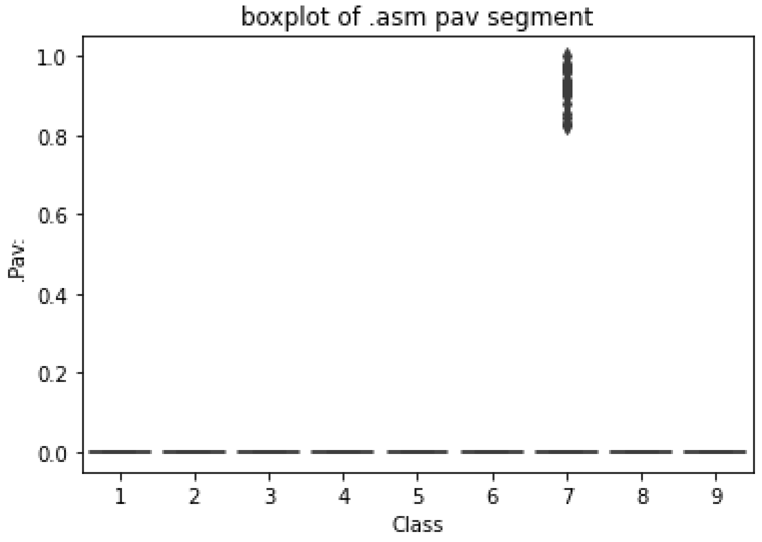


```
time: 231 ms (started: 2022-06-07 16:09:06 +00:00)
```

The plot is between Text and class

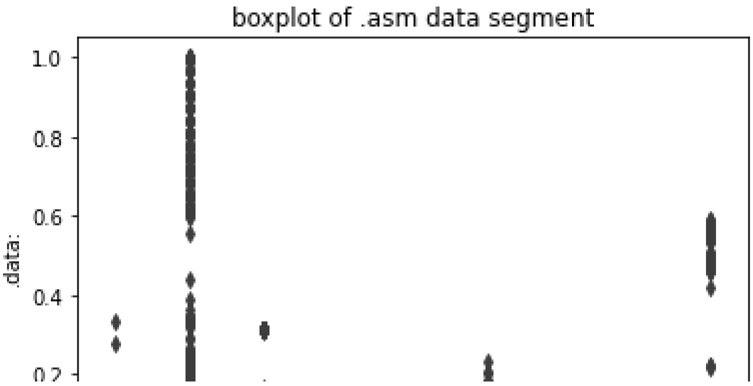
Class 1,2 and 9 can be easly separated

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



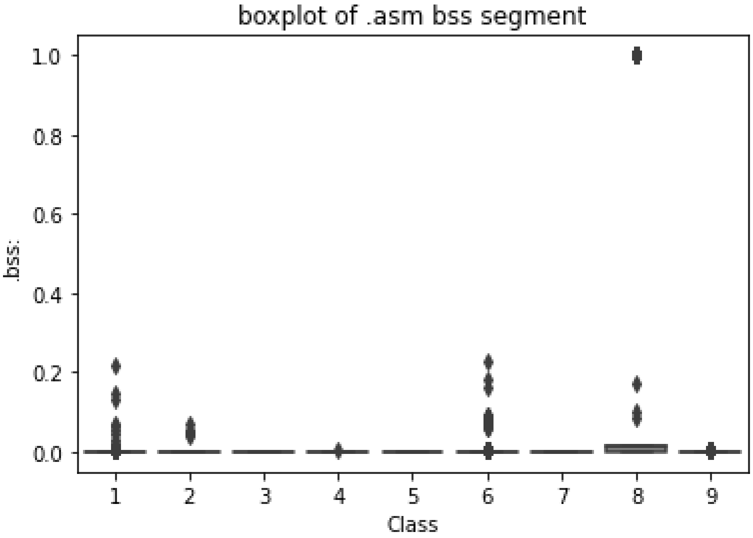
```
time: 218 ms (started: 2022-06-07 16:09:14 +00:00)
```

```
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
```

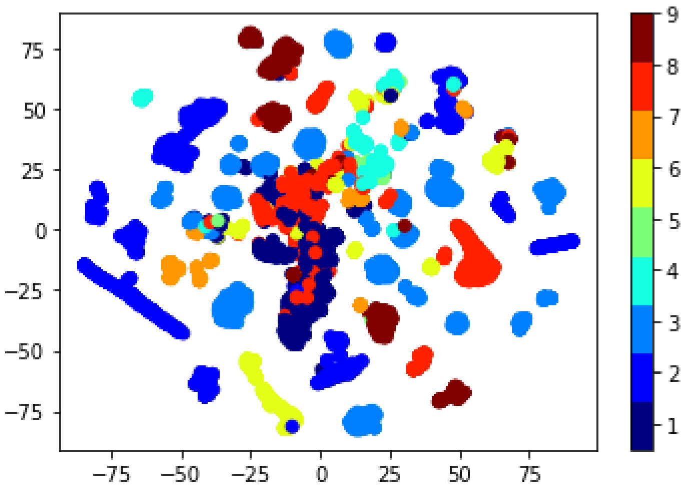
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



```
time: 208 ms (started: 2022-06-07 16:09:27 +00:00)
```

4.2.2 Multivariate Analysis on .asm file features

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

#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=asm_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



```
time: 1min 46s (started: 2022-06-07 16:15:40 +00:00)
```

4.4. Machine Learning models on features of .asm files

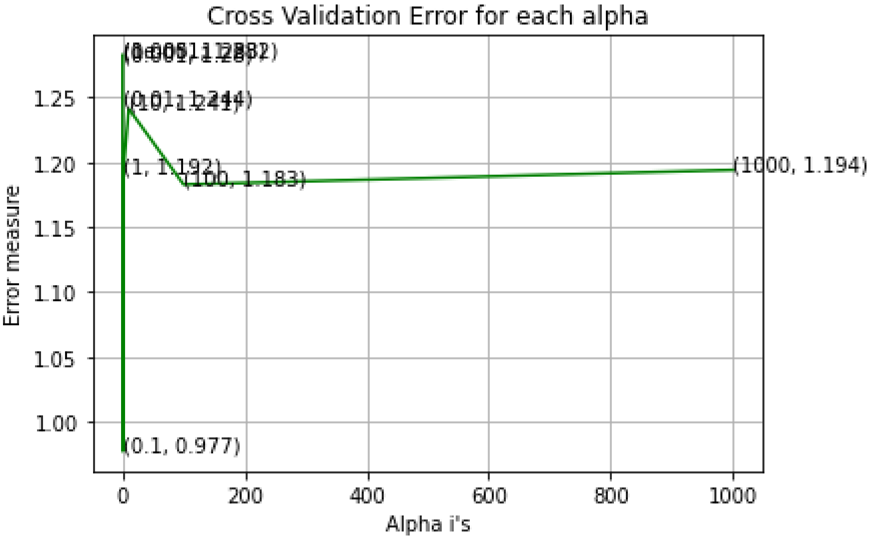
4.4.2 Logistic Regression

```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in tqdm(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])
```

```
100%|██████████| 9/9 [00:25<00:00,  2.88s/it]log_loss for c =  1e-05 is 1.2826638789405893
log_loss for c =  0.0001 is 1.2824218223389359
log_loss for c =  0.001 is 1.2796779462770196
log_loss for c =  0.01 is 1.244323265977828
log_loss for c =  0.1 is 0.9773054102627817
log_loss for c =  1 is 1.1923809137791286
log_loss for c =  10 is 1.2412234466585486
log_loss for c =  100 is 1.182767178893285
log_loss for c =  1000 is 1.1940048604112525
time: 26 s (started: 2022-06-07 16:07:03 +00:00)
```

```
from matplotlib import mlab
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
best_alpha = np.argmin(cv_log_error_array)

best_alpha = np.argmin(cv_log_error_array)
```

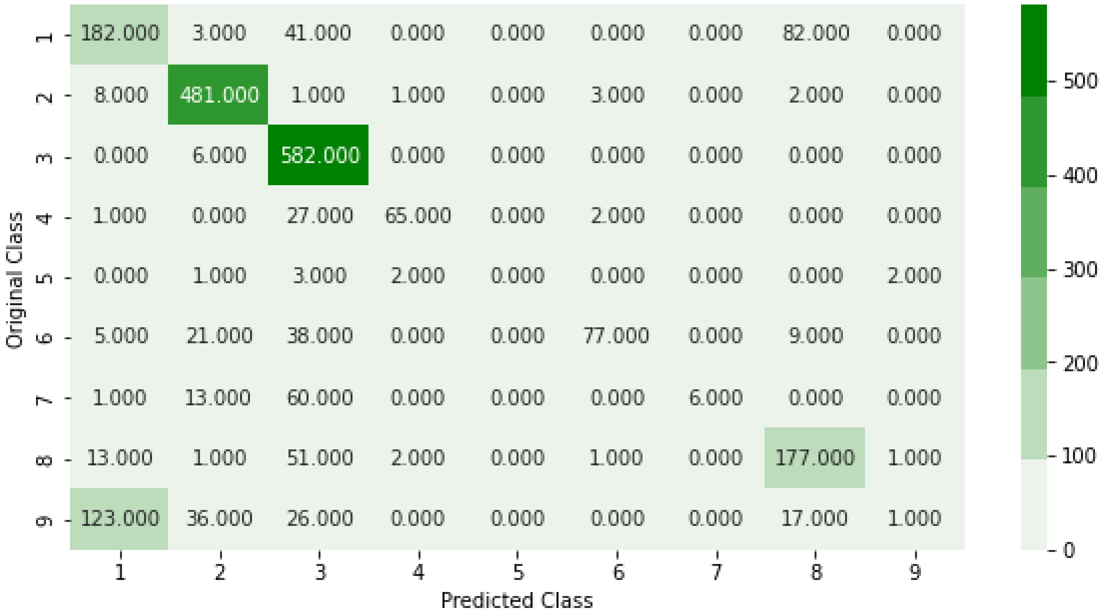
```
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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



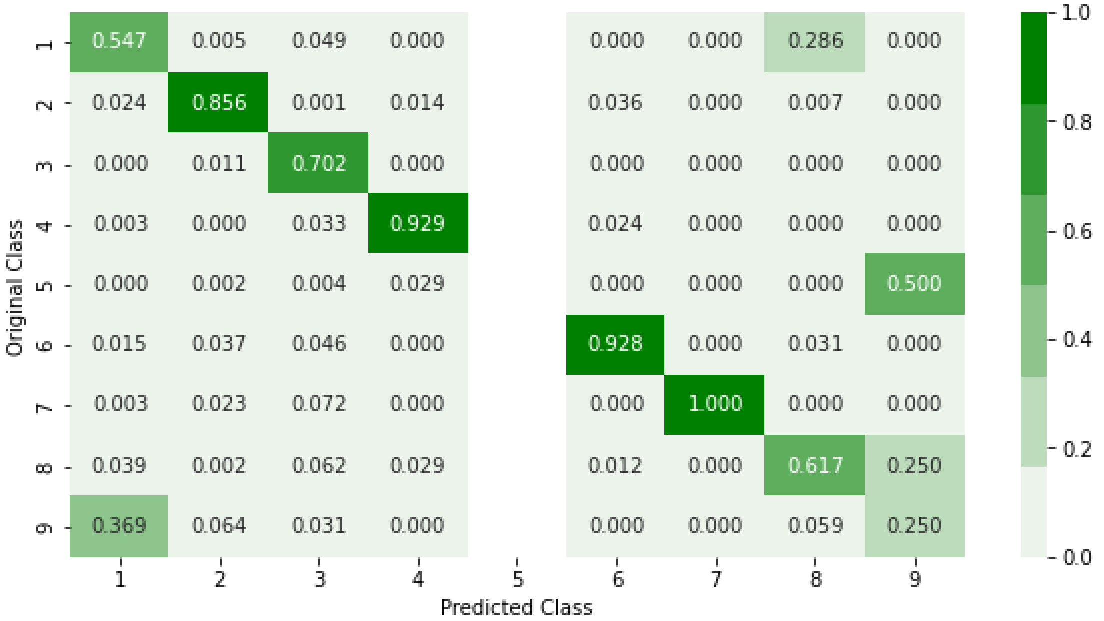
```
time: 305 ms (started: 2022-06-07 16:07:32 +00:00)
```

```
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
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-1
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

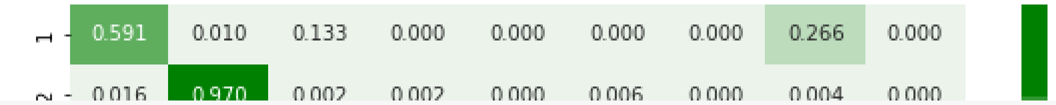
```
log loss for train data 0.9983711674002989
log loss for cv data 0.9773054102627817
log loss for test data 0.9798208056609188
Number of misclassified points  27.736890524379028
-------------------------------------------------- Confusion matrix --------------------------
```



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

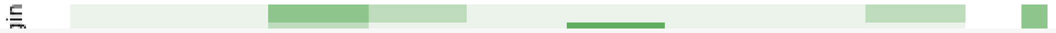


```
Sum of columns in precision matrix [ 1.   1.   1.   1. nan  1.   1.   1.   1.]
-------------------------------------------------- Recall matrix ------------------------------
```



```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/asm/sig_clf_uni' ,
    joblib.dump(sig_clf, f)
```

```
time: 15.3 s (started: 2022-06-07 07:03:01 +00:00)
```



```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/sig_clf_uni'
    sig_clf= joblib.load(f)
```

```
time: 24.7 ms (started: 2022-06-06 19:07:02 +00:00)
```

Observation For Unigram ASM Files:-

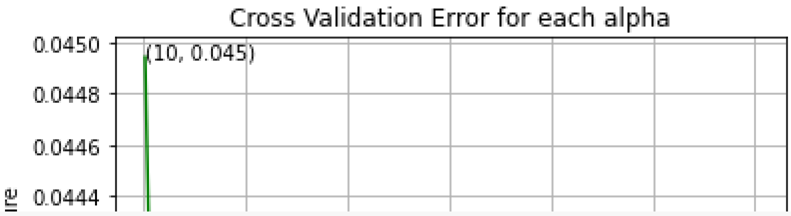
1. Random Forest is performing better than LR Model.
2. Unlike LR, it can predict 80% datapoint which belong to Class=5 correctly, but, still 20% are misclassified.

4.4.3 Random Forest Classifier

```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in tqdm(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])
```

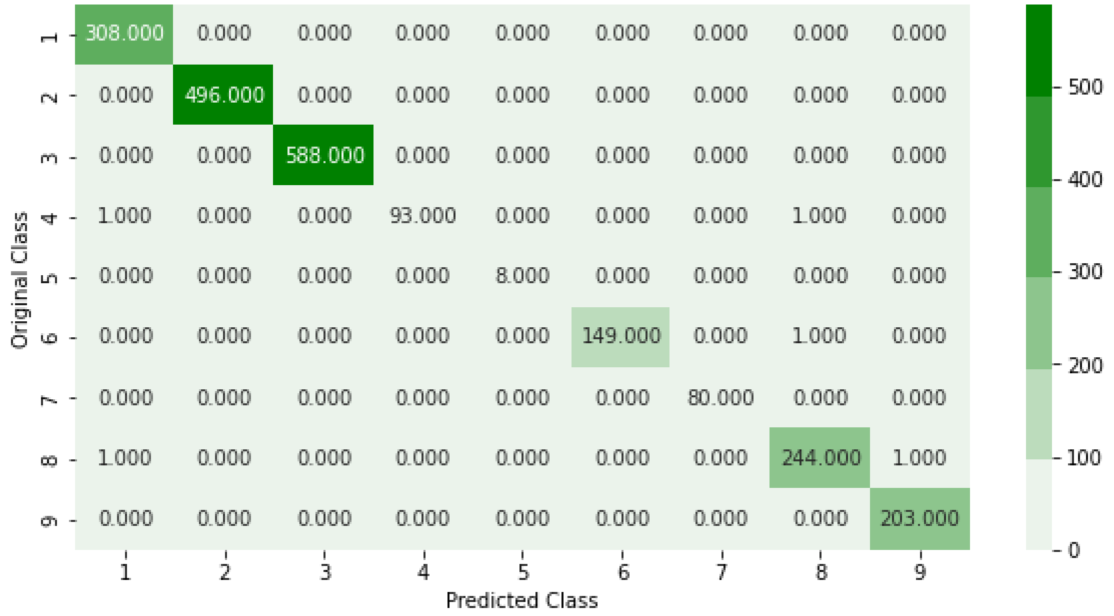
```
100%|██████████| 7/7 [07:33<00:00, 64.72s/it] log_loss for c =  10 is 0.044942264056051326
log_loss for c =  50 is 0.04344539795250946
log_loss for c =  100 is 0.04339087592601071
log_loss for c =  500 is 0.043812809298841904
log_loss for c =  1000 is 0.044035942016339355
log_loss for c =  2000 is 0.044165647214066954
log_loss for c =  3000 is 0.0441974844936852
time: 7min 33s (started: 2022-06-07 16:58:55 +00:00)
```

```
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-1
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 train data 0.015161653206078792
log loss for cv data 0.04339087592601071
log loss for test data 0.026253744327192956
Number of misclassified points  0.22999080036798528
-------------------------------------------------- Confusion matrix --------------------------
```



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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

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

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

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
RandomizedSearchCV(estimator=XGBClassifier(), n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5],
                                        'learning_rate': [0.01, 0.05, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500],
                                        'subsample': [0.1, 0.3, 0.5]},
                   verbose=10)time: 6min 14s (started: 2022-06-07 17:09:47 +00:00)
```

```
xgb_bytes.best_estimator_
```

```
XGBClassifier(colsample_bytree=0.3, learning_rate=0.05, n_estimators=500,
              objective='multi:softprob', subsample=0.3)time: 6.19 ms (started: 2022-06-07 17:
```

```
best_xgb =XGBClassifier(colsample_bytree=0.3, learning_rate=0.05, n_estimators=500,
              objective='multi:softprob', subsample=0.3)



best_xgb.fit(X_train_asm,y_train_asm)
```

```
XGBClassifier(colsample_bytree=0.3, learning_rate=0.05, n_estimators=500,
              objective='multi:softprob', subsample=0.3)time: 23.3 s (started: 2022-06-07 17:1
```

```
# with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/best_xgb' ,
#     joblib.dump(best_xgb, f)
```

```
time: 56.9 ms (started: 2022-06-07 09:46:54 +00:00)
```

```
with open('/content/gdrive/My Drive/Assignments AAIC/Assignment 16 Malware/Final/bytes/best_xgb' , '
    best_xgb= joblib.load(f)
```

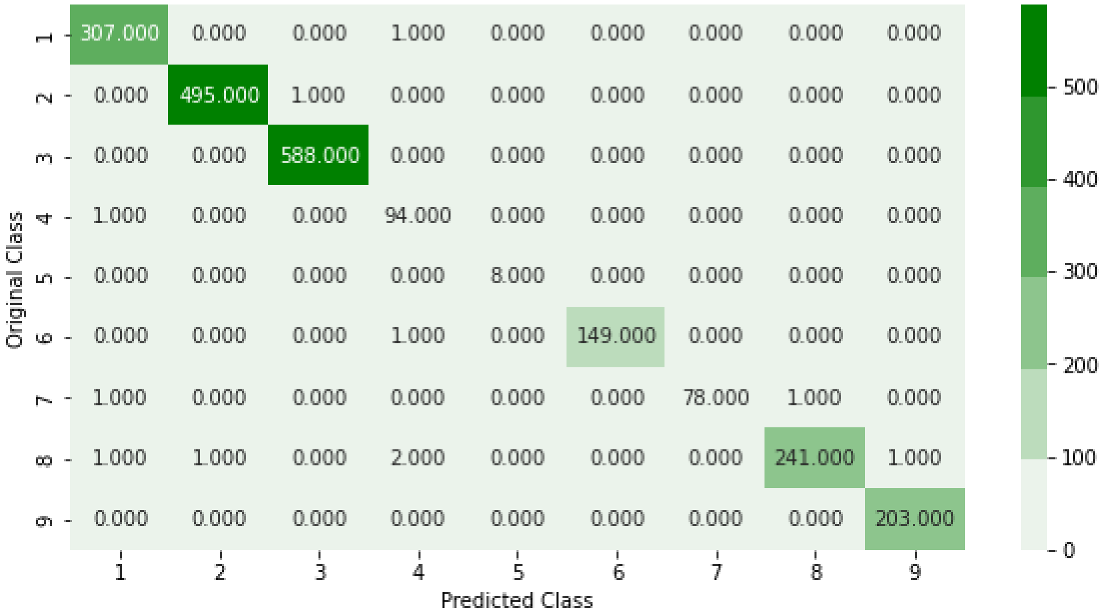
```
time: 24.7 ms (started: 2022-06-06 19:07:02 +00:00)
```

```
best_n_estimator= 500
```

```
time: 1.24 ms (started: 2022-06-07 17:22:26 +00:00)
```

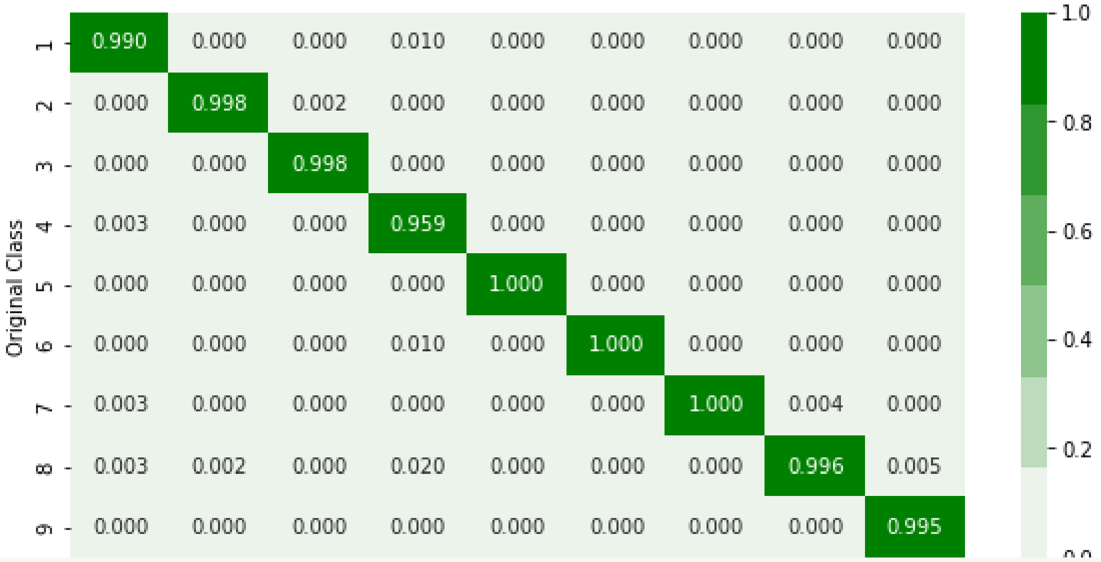
```
predict_y = best_xgb.predict_proba(X_train_asm)
print('For values of best alpha = ', best_n_estimator, "The train log loss is:",log_loss(y_train_asm
predict_y = best_xgb.predict_proba(X_cv_asm)
print('For values of best alpha = ', best_n_estimator, "The cross validation log loss is:",log_loss(
predict_y = best_xgb.predict_proba(X_test_asm)
print('For values of best alpha = ', best_n_estimator, "The test log loss is:",log_loss(y_test_asm,
plot_confusion_matrix(y_test_asm, best_xgb.predict(X_test_asm))
```

```
For values of best alpha =  500 The train log loss is: 0.00646627399239934
For values of best alpha =  500 The cross validation log loss is: 0.02163638917661411
For values of best alpha =  500 The test log loss is: 0.014576602488400186
Number of misclassified points  0.5059797608095675
-------------------------------------------------- Confusion matrix ---------------------------
```



	1	2	3	4	5	6	7	8	9
1	307.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
2	0.000	495.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	1.000	0.000	0.000	94.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	8.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	1.000	0.000	149.000	0.000	0.000	0.000
7	1.000	0.000	0.000	0.000	0.000	0.000	78.000	1.000	0.000
8	1.000	1.000	0.000	2.000	0.000	0.000	0.000	241.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	203.000

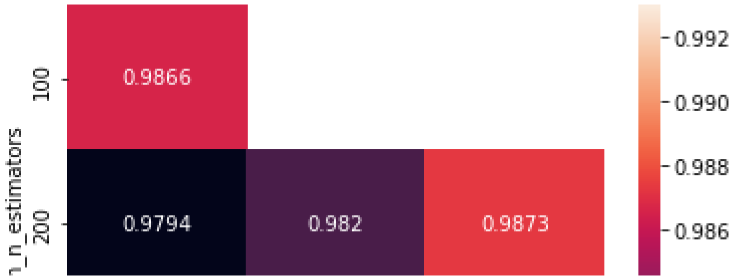
```
-------------------------------------------------- Precision matrix ---------------------------
```



1	0.990	0.000	0.000	0.010	0.000	0.000	0.000	0.000	0.000
2	0.000	0.998	0.002	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.998	0.000	0.000	0.000	0.000	0.000	0.000
4	0.003	0.000	0.000	0.959	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.010	0.000	1.000	0.000	0.000	0.000
7	0.003	0.000	0.000	0.000	0.000	0.000	1.000	0.004	0.000
8	0.003	0.002	0.000	0.020	0.000	0.000	0.000	0.996	0.005
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.995

```
import seaborn as sn
# Ref:- https://stackoverflow.com/questions/56302647/how-to-plot-a-heatmap-and-find-best-hyperparame

results = pd.DataFrame.from_dict(xgb_bytes.cv_results_)
max_scores = results.groupby(['param_n_estimators', 'param_max_depth']).max()
max_scores = max_scores.unstack()[['mean_test_score']]

print(f' {max_scores}')
sn.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
# NOTE:- Some Blocks empty as it's Randomsearch so it didn't fit those combinations
```

```
                    mean_test_score
param_max_depth                   3         5        10
param_n_estimators
100                        0.986628       NaN       NaN
200                        0.979439  0.982027  0.987347
500                        0.992955  0.989648       NaN
```



Observation:-

1. XGBoost performed the best out of them all with the least Log Loss of 0.06
2. But, it like RF still simply can't predict Class 5 that well.

```
time: 290 ms (started: 2022-06-07 17:36:21 +00:00)
```

Steps to convert files into images:-

1. Find First 800 Pixels as the Kaggle Winners found it to be the best through CV.

```
# Make 8 Folders for moving byte files
location = 'asmfiles/'
files = os.listdir('asmfiles/')
data = list(range(0,10868))

for i in tqdm(range(10868)):
    if i%8 == 0:
        shutil.move(location+files[data[i]],'asm1/')
    if i%8 == 1:
        shutil.move(location+files[data[i]],'asm2/')
    if i%8 == 2:
        shutil.move(location+files[data[i]],'asm3/')
    if i%8 == 3:
        shutil.move(location+files[data[i]],'asm4/')
    if i%8 == 4:
        shutil.move(location+files[data[i]],'asm5/')
    if i%8 == 5:
        shutil.move(location+files[data[i]],'asm6/')
    if i%8 == 6:
        shutil.move(location+files[data[i]],'asm7/')
    if i%8 == 7:
        shutil.move(location+files[data[i]],'asm8/')
```

```
100%|██████████| 10868/10868 [00:38<00:00, 278.91it/s]time: 39 s (started: 2022-06-08 17:30:09
```

```
cd ..
```

```
d:\Applied Ai Course\Assignments\Assgn 16 Malware Detection
time: 15 ms (started: 2022-06-08 19:08:18 +05:30)
```

```
# No of byte files per folder
for folder in range(1,9):
    f = os.listdir('asm'+str(folder)+'/')
    print(f'Folder = {folder} and length {len(f)}')
```

```
Folder = 1 and length 1360
Folder = 2 and length 1359
Folder = 3 and length 1359
Folder = 4 and length 1359
Folder = 5 and length 1358
Folder = 6 and length 1358
Folder = 7 and length 1358
Folder = 8 and length 1358
time: 156 ms (started: 2022-06-08 19:08:23 +05:30)
```

```
# https://www.sharpsightlabs.com/blog/skimage-imread/
import array
import io
import cv2
```

```
time: 0 ns (started: 2022-06-08 19:08:28 +05:30)
```

```
pwd
```

```
'd:\\Applied Ai Course\\Assignments\\Assgn 16 Malware Detection'time: 0 ns (started: 2022-06-0
```

```
asmfiles= os.listdir("asm1")
asmfiles[:5]
```

```
['01azqd4InC7m9JpocGv5.asm',
 '01azqd4InC7m9JpocGv5.png',
 '02mlBLHZTDFXGa7Nt6cr.asm',
 '04mcPSei852tgIKUwTJr.asm',
 '05rJTUWYAKNegBk2wE8X.asm']time: 0 ns (started: 2022-06-08 19:08:40 +05:30)
```

Example for creation of Image from ASM Text File stored in Final Folder

```
# https://stackoverflow.com/questions/5250744/difference-between-open-and-codecs-open-in-python
import io

asmfile="0ACDbR5M3ZhBJajygTuf.asm"
file_name = asmfile.split('.')[0]
# Open the ASM file
file = io.open('asm1/'+asmfile,'rb')
# Get it's size
file_size = os.path.getsize('asm1/'+asmfile)
print(file_size)

# width = sqrt(file size)
# So that we can make width * width ~ file size ie a square image with roughly same no of Pixels as
```

```
width = int(file_size**0.5)
print(width)
rem = (file_size/width)
print(rem)

# 'B'  is for 8 bit values ie 0-255 value only allowed per array cell
ar = array.array('B')
print(type(ar))
print(ar[:100])
# https://stackoverflow.com/questions/55225542/how-to-create-an-image-from-a-string
ar.frombytes(file.read()) # Create an image object
print(len(ar))

file.close()
print(ar)
reshaped = np.reshape(ar[:width * width], (width, width))  # creating the shape of image
reshaped = np.uint8(reshaped)
# print(reshaped.shape)

print(reshaped[:100])
```

```
12153703
3486
3486.4323006310956
<class 'array.array'>
array('B')
12153703
array('B', [72, 69, 65, 68, 69, 82, 58, 48, 48, 52, 48, 48, 48, 48, 48, 9, 9, 9, 9, 9, 9, 9, 59
[[ 72  69  65 ...  49  52  32]
 [ 48  48  32 ... 104  44  32]
 [ 48  65  52 ...  32  48  65]
 ...
 [ 50  54  49 ...  48  49  69]
 [ 67 104  44 ... 101 120 116]
 [ 58  48  48 ...  48  48  52]]
time: 2.11 s (started: 2022-06-08 19:03:32 +05:30)
```

```
def asm_image(folder):
    # https://stackoverflow.com/questions/5250744/difference-between-open-and-codecs-open-in-python

    for asm_file in tqdm(os.listdir(folder_name+'/')):

        file_name = asm_file.split('.')[0]
        file = io.open(folder + '/'+ asm_file,'rb')
        # Get it's size
        file_size = os.path.getsize(folder_name+ '/'+asmfile)
        width=0
        # width = sqrt(file size)
        # So that we can make width * width ~ file size ie a square image with roughly same no of Pi
        width = int(file_size**0.5)
        ar = array.array('B')
        # https://stackoverflow.com/questions/55225542/how-to-create-an-image-from-a-string
        ar.frombytes(file.read()) # Create an image object
        file.close()

        # creating the shape of image eg:- for one asm file:- ar[row : 7678 * 7678] reshaped to ar(7
        reshaped = np.reshape(ar[:width * width], (width, width))
        reshaped = np.reshape(ar[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        cv2.imwrite(folder_name + '/' + file_name + '.png',reshaped)
```

```
time: 0 ns (started: 2022-06-08 18:11:40 +05:30)
```

```
pwd
```

```
'd:\\Applied Ai Course\\Assignments\\Assgn 16 Malware Detection'time: 0 ns (started: 2022-06-0
```

```
%%time
# Create Images and store it in asm_image folder
for folder in range(1,9):
    folder_name= 'asm'+ str(folder)
    print(f'Processing Folder {folder_name}')
    asm_image(folder_name)
```

```
Processing Folder asm1
  0%|          | 1/1360 [00:00<15:40,  1.44it/s]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File <timed exec>:5, in <module>

d:\Applied Ai Course\Assignments\Assgn 16 Malware Detection\Final\Copy of MicrosoftMalwareDetec
     <a href='vscode-notebook-cell:/d%3A/Applied%20Ai%20Course/Assignments/Assgn%2016%20Malware
     <a href='vscode-notebook-cell:/d%3A/Applied%20Ai%20Course/Assignments/Assgn%2016%20Malware
asm file:- ar[row : 7678 * 7678] reshaped to ar(7678,7678)
```

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

4.5.1. Merging both asm and byte file features

```
    <a href='file:///c%3A/Users/chiranjiv/.conda/envs/malware/lib/site-packages/numpy/core/from
```

```
result.head()
```

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148

5 rows × 260 columns

```
---> <a href='file:///c%3A/Users/chiranjiv/.conda/envs/malware/lib/site-packages/numpy/core/fro
```

```
result_asm.head()
```

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

5 rows × 54 columns

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