Installing missing Packages

In [1]:

```
!pip install pyunpack patool dask fsspec
```

```
Collecting pyunpack
  Downloading pyunpack-0.2.2-py2.py3-none-any.whl (3.8 kB)
Collecting patool
  Downloading patool-1.12-py2.py3-none-any.whl (77 kB)
     |████████████████████████████████| 77 kB 3.9 MB/s
Requirement already satisfied: dask in /usr/local/lib/python3.7/dist-packages (2.12.
0)
Collecting fsspec
  Downloading fsspec-2021.8.1-py3-none-any.whl (119 kB)
     |████████████████████████████████| 119 kB 21.6 MB/s
Collecting easyprocess
  Downloading EasyProcess-0.3-py2.py3-none-any.whl (7.9 kB)
Collecting entrypoint2
  Downloading entrypoint2-0.2.4-py3-none-any.whl (6.2 kB)
Installing collected packages: entrypoint2, easyprocess, pyunpack, patool, fsspec
Successfully installed easyprocess-0.3 entrypoint2-0.2.4 fsspec-2021.8.1 patool-1.12
pyunpack-0.2.2
```

Importing Packages

In [32]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
%matplotlib inline
import dask.dataframe as dd
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import seaborn as sns
from tqdm import tqdm
from array import array
import numpy as np
import pickle
from IPython.display import display, HTML
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 collections import Counter
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.preprocessing import normalize
from sklearn.feature_selection import SelectFromModel
```

Byte Features

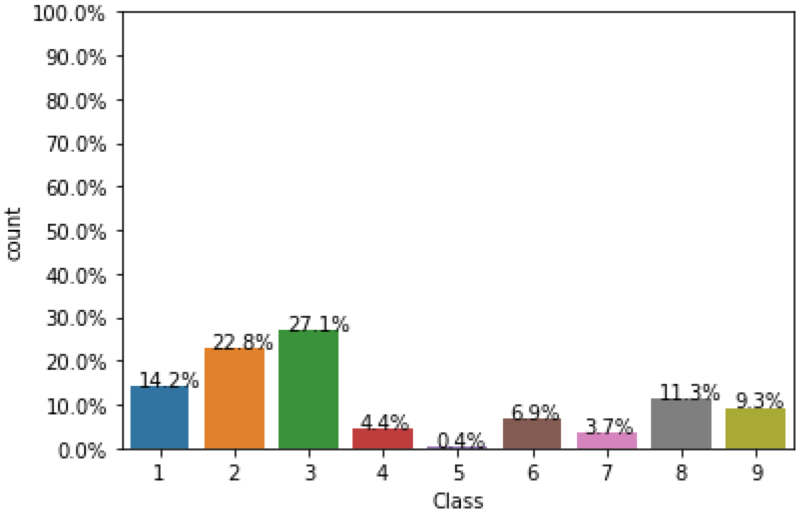
Importing Data

In [ ]:

```
Y = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware Detectio
```

In [ ]:

```
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_heig

ax.yaxis.set_ticks(np.linspace(0, total, 11))

ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
plt.show()
```



Feature Extraction

File Size of Byte Files as a feature

In [ ]:

```
byteFiles = '/content/drive/Shareddrives/colab/byteFiles/'
```

In [ ]:

```
files = os.listdir(byteFiles)
filenames = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in files:
  # os.stat() performs a stat system call on the given path i.e., returns some infor
  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])
    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  fpiZ6no01V8gydTe4UFw  1.851562      4
1  dKt4HhezElT2nBIP6c5F  6.703125      3
2  eGwk8W6m4NIzsAaHvfx3  0.222656      4
3  GYo8tD76OWx0jkyIB1iL  0.832031      8
4  bRPa6hIrozuSpfAGyOXT  4.183594      7
```

In [ ]:

```
ax = sns.boxplot(x = 'Class', y= 'Size', data = data_size_byte)
plt.title('Boxplot of .bytes file sizes')
plt.show()
```

Feature Extraction from Byte Files

Unigrams

1. Removal of address from each Byte File

2. Convert the Hex Codes to Bag of Words.

3. Unigrams and Bigrams

In [ ]:

```
files = os.listdir(byteFiles)
filenames = []
array = []

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

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

byte_feature_file=open('/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware D
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i, row in enumerate(feature_matrix[k]):
        if i!=len(feature_matrix[k])-1:
            byte_feature_file.write(str(row)+",")
        else:
            byte_feature_file.write(str(row))
    byte_feature_file.write("\n")
    k += 1

byte_feature_file.close()
```

In [ ]:

```
byte_features = pd.read_csv('/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malw
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)
```

Out[ ]:

	ID	0	1	2	3	4	5	6	7	8	9	0a	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	3205	3211	35
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	358	340	66

2 rows × 258 columns

In [ ]:

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

Out[ ]:

	ID	Size	Class
0	fpiZ6no01V8gydTe4UFw	1.851562	4
1	dKt4HhezEIT2nBIP6c5F	6.703125	3

In [ ]:

```
byte_features_with_size = byte_features.merge(data_size_byte, on = 'ID')
byte_features_with_size.to_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft M
byte_features_with_size.head(2)
```

Out[ ]:

	ID	0	1	2	3	4	5	6	7	8	9	0a	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	3205	3211	35
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	358	340	66

2 rows × 260 columns

In [ ]:

```
byte_features_with_size = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Micr
```

In [ ]:

```
# Normalizing Columns
def normalize(df):
  result_copy = 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()
      result_copy[feature_name] = (df[feature_name] - min_value)/(max_value - min_va
  return result_copy
```

In [ ]:

```
result = normalize(byte_features_with_size)
```

In [ ]:

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

Out[ ]:

	Unnamed: 0	ID	0	1	2	3	4	5	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0
2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0
3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0
4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0

5 rows × 261 columns

Bigrams

1. Create vocab: String with all possible bigram combinations. This will be used as column heading as well.

2. Depending on the number of cores available, divide the dataset into x number of parts. Each part will be processed by one core.

3. Each process will parse the byte file,

In [ ]:

```
hexadecimal_alphabet = list("0123456789abcdef")
vocab = []
for i in hexadecimal_alphabet:
  for j in hexadecimal_alphabet:
    vocab.append(i+j)
vocab = list(set(vocab))
vocab.append("??")
vocab.sort()
```

In [ ]:

```
vocab_string = ','.join(vocab)
print(vocab_string)
```

```
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
```

In [ ]:

```
bigrams_list = []
for i in vocab:
  for j in vocab:
    bigrams_list.append(i+"_"+j)
bigrams_list = sorted(bigrams_list)
bigrams_string = ",".join(bigrams_list)
```

In [ ]:

```
def bigrams_from_line(text):
  """
  This function takes a line and returns all bigrams in that line.
  https://stackoverflow.com/questions/21844546/forming-bigrams-of-words-in-list-of-s
  """
  bigrams_in_line = [bigram for bigram in zip(text.split(" ")[:-1], text.split(" ")[
  bigrams_in_line = ["_".join(bigram) for bigram in bigrams_in_line]
  return bigrams_in_line
```

In [ ]:

```
def bigrams_from_file(loc, bigrams_list):
  """
  This function takes the path to a file and returns a counter with all bigrams coun
  """
  with open(byteFiles + loc, 'r') as byteFile:
    counter = Counter()
    counter.update({x:0 for x in bigrams_list})
    for line in byteFile:
      line_lowercase = line.rstrip().lower()
      line_bigrams = bigrams_from_line(line_lowercase)
      counter.update(line_bigrams)
  byteFile.close()
  return counter
```

In [ ]:

```
!lscpu | grep 'Core(s) per socket:'
```

```
Core(s) per socket:  2
```

In [ ]:

```
def singleprocess():
  files = os.listdir(byteFiles)
  output_file = open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware Det
  output_file.write("ID,"+bigrams_string+"\n")
  for file in tqdm(files):
    output_file.write(file.split('.')[0]+",")
    counter = bigrams_from_file(file, bigrams_list)
    line_bigrams = [pair[1] for pair in sorted(counter.items())]
    line_bigrams_str = ','.join(str(i) for i in line_bigrams)
    output_file.write(line_bigrams_str+"\n")
  output_file.close()
```

In [ ]:

```
singleprocess()
```

```
100%|██████████| 10868/10868 [3:57:04<00:00,  1.31s/it]
```

Selecting top 2500 bigrams

In [ ]:

```
bigrams = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware De
```

In [ ]:

```
bigrams.head()
```

Out[ ]:

	ID	00_00	00_01	00_02	00_03	00_04	00_05	00_06	00_07	00_08	00_09
0	dKt4HhezElT2nBIP6c5F	6426	42	24	79	20	52	25	36	38	49
1	eGwk8W6m4NIzsAaHvfx3	2263	13	3	85	3	1	3	1	1	6
2	GYo8tD76OWx0jkyIB1iL	26575	538	55	78	38	11	10	41	60	23
3	bRPa6hIrozuSpfAGyOXT	3492	15	3	3	0	1	1	25	2	1
4	BSafFJTth4U3uibE6sZO	1663	3	11	5	1	0	1	0	79	2

5 rows × 66050 columns

In [ ]:

```
bigrams.sort_values('ID', ignore_index=True, inplace=True)
```

In [ ]:

```
Y.sort_values('Id', ignore_index=True, inplace = True)
```

In [ ]:

```
X_train, X_test, y_train, y_test  = train_test_split(bigrams, Y, test_size = 0.3)
```

In [ ]:

```
X_train.head()
```

Out[ ]:

	ID	00_00	00_01	00_02	00_03	00_04	00_05	00_06	00_07	00_08	0
1846	5MK9AFWTf6s8vQtD2yNO	20229	83	33	109	17	10	18	42	13	
9544	gjOIy9sRbtFGoWASY16c	27160	799	192	153	1857	96	155	174	125	
787	2M9jHWhCOGBtY4Jbsvcy	13934	178	60	49	147	25	10	57	18	
4081	BiKc6IFPEovX59sgzLp4	26571	813	270	257	332	186	177	165	500	
3141	8va102hpJn5DVLe9i6Fq	2774	149	92	68	105	45	35	14	107	

5 rows × 66050 columns

In [ ]:

```
X_train.drop("ID", inplace=True, axis = 1)
X_test.drop("ID", inplace=True, axis = 1)
y_train.drop("Id", inplace=True, axis = 1)
y_test.drop("Id", inplace=True, axis = 1)
```

In [ ]:

```
clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
clf.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [ ]:

```
sfm = SelectFromModel(clf, max_features = 2500)
```

```
sfm.fit(X_train, y_train)
```

Out[ ]:

```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                 class_weight=None,
                                                 criterion='gini',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 max_samples=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100, n_jobs=-1,
                                                 oob_score=False,
                                                 random_state=0, verbose=0,
                                                 warm_start=False),
                max_features=2500, norm_order=1, prefit=False, threshold=None)
```

In [ ]:

```
sfm.get_support(indices=True)
```

Out[ ]:

```
array([    0,     1,     2, ..., 66038, 66045, 66048])
```

In [ ]:

```
important_bigrams = [X_train.columns[i] for i in sfm.get_support(indices=True)]
len(important_bigrams)
```

Out[ ]:

```
2500
```

In [ ]:

```
bigrams_only_important = bigrams[important_bigrams]
```

In [ ]:

```
bigrams_only_important["ID"] = bigrams["ID"]
bigrams_only_important.head()
```

Out[ ]:

	00_00	00_01	00_02	00_03	00_04	00_05	00_06	00_07	00_08	00_09	00_0a	00_0c	00_0d
0	19852	719	64	43	159	10	6	10	35	8	12	23	17
1	15288	58	20	110	8	11	3	5	8	2	0	7	0
2	273053	1002	801	1170	943	840	1125	1003	860	987	973	1278	997
3	16032	592	157	144	509	590	551	146	523	154	155	525	168
4	9903	204	59	69	103	34	19	21	55	14	21	66	14

5 rows × 2501 columns

In [ ]:

```
bigrams_only_important.to_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Ma
```

Importing Bigrams CSV

In [ ]:

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

```
Mounted at /content/drive
```

In [ ]:

```
from sklearn.preprocessing import normalize
```

In [ ]:

```
bigrams = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware De
```

In [ ]:

```
byte_features = bigrams.merge(result, on = "ID")
```

In [ ]:

```
 from sklearn.preprocessing import normalize
```

In [ ]:

```
byte_features_no_id = byte_features.drop(["ID", "Class"], axis = 1)
byte_features_no_id_norm = normalize(byte_features_no_id)
bigram_column_names = byte_features_no_id.columns
byte_features_norm = pd.DataFrame(data=byte_features_no_id_norm, columns = bigram_co
byte_features_ids = byte_features["ID"]
byte_features_classes = byte_features["Class"]
byte_features_norm.insert(loc = 0, column = "ID", value = byte_features_ids)
byte_features_norm.insert(loc = 0, column = "Class", value = byte_features_classes)
y = byte_features_norm["Class"]
byte_features_norm.drop("Unnamed: 0_x", axis = 1, inplace =True)
byte_features_norm.drop("Class", inplace=True, axis = 1)
byte_features_norm.drop("ID", inplace = True, axis = 1)
byte_features_norm.head()
```

Out[ ]:

	00_00	00_01	00_02	00_03	00_04	00_05	00_06	00_07	00_08	00_09
0	0.448546	0.016245	0.001446	0.000972	0.003593	0.000226	0.000136	0.000226	0.000791	0.000181
1	0.865108	0.003282	0.001132	0.006225	0.000453	0.000622	0.000170	0.000283	0.000453	0.000113
2	0.996383	0.003656	0.002923	0.004269	0.003441	0.003065	0.004105	0.003660	0.003138	0.003602
3	0.587520	0.021695	0.005754	0.005277	0.018653	0.021622	0.020192	0.005350	0.019166	0.005644
4	0.597321	0.012305	0.003559	0.004162	0.006213	0.002051	0.001146	0.001267	0.003317	0.000844

5 rows × 2759 columns

Multivariate Analysis

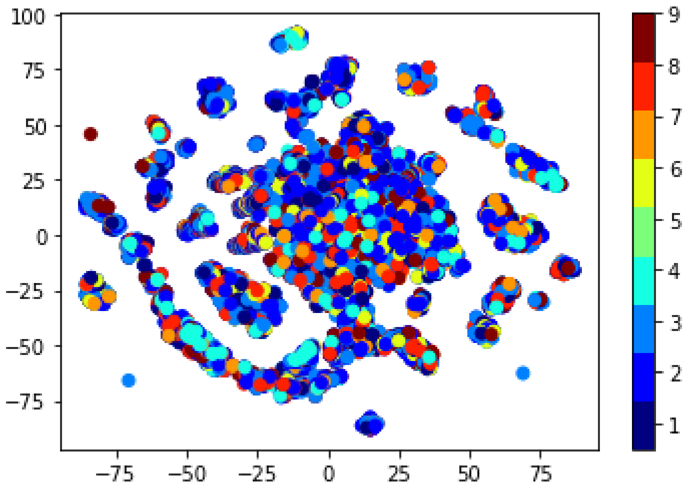
T-SNE to check whether the features made so far using the Byte Files are helpful or not in classifying malware.

In [ ]:

```
xtsne = TSNE(perplexity=50)
results = xtsne.fit_transform(byte_features_norm)
```

In [ ]:

```
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()
```

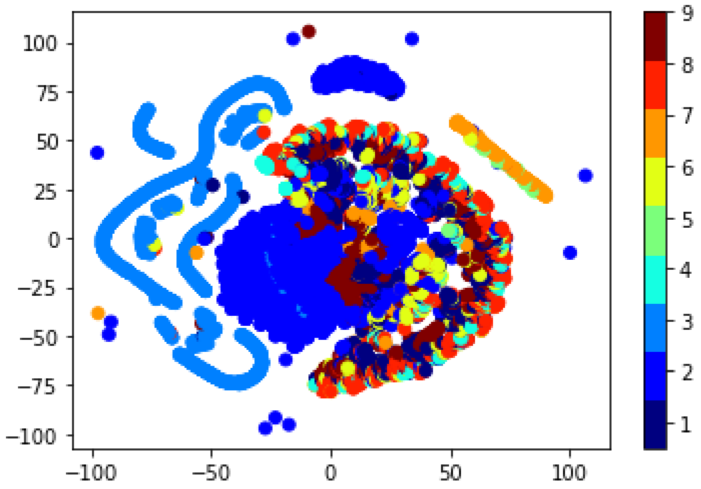


In [ ]:

```
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
```

In [ ]:

```
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 is able to seperate them to some extent. This shows that the results are quite useful.

Train CV Test Split

Random split since the dataset is not of temporal nature.

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(byte_features_norm, y ,stratify=
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,te
```

In [ ]:

```
print(f"Number of datapoints in train data: {X_train.shape[0]}")
print(f"Number of datapoints in cross validation data: {X_cv.shape[0]}")
print(f"Number of datapoints in test data: {X_test.shape[0]}")
```

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

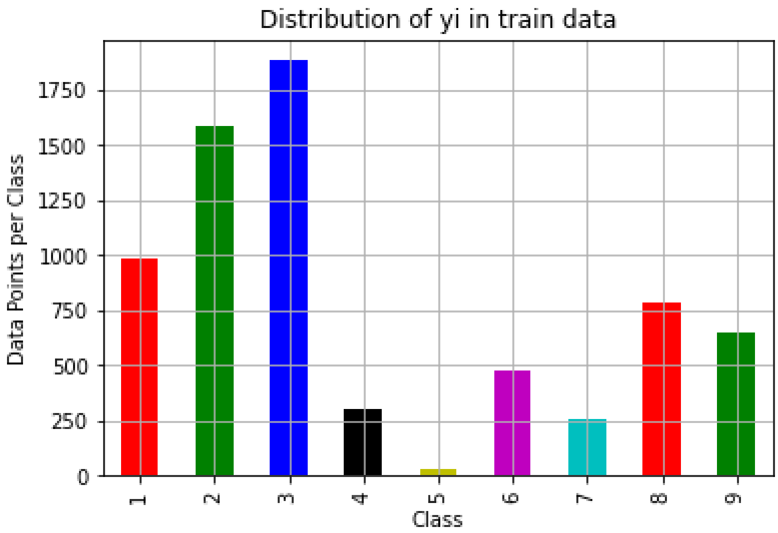
Plotting the Class Distribution among the three datasets.

In [ ]:

```
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()
```

In [ ]:

```
plot_colors = list('rgbkymc')
train_class_distribution.plot(kind = 'bar', color = plot_colors)
plt.xlabel('Class')
plt.ylabel('Data Points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
```



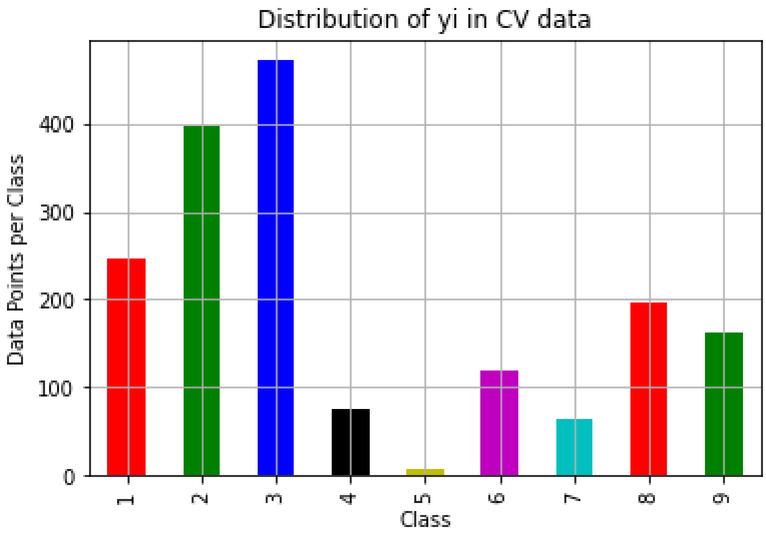
In [ ]:

```
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
```

```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
```

In [ ]:

```
plot_colors = list('rgbkymc')
cv_class_distribution.plot(kind = 'bar', color = plot_colors)
plt.xlabel('Class')
plt.ylabel('Data Points per Class')
plt.title('Distribution of yi in CV data')
plt.grid()
plt.show()
```



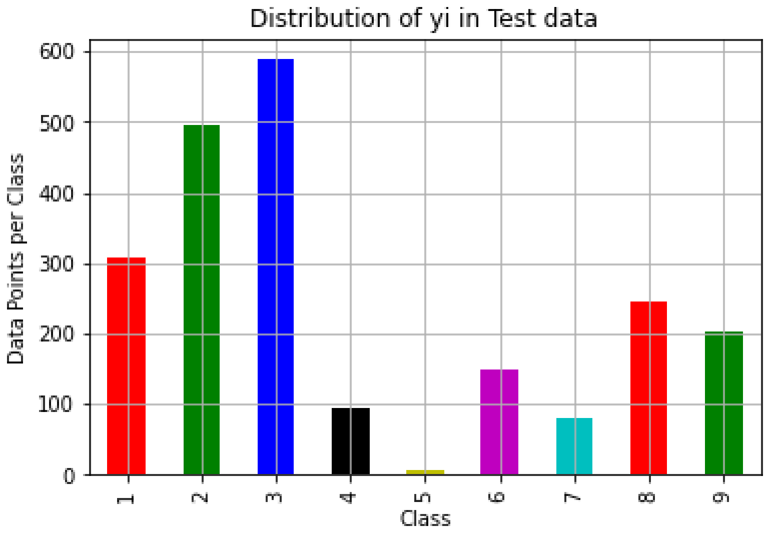
In [ ]:

```
sorted_yi = np.argsort(-cv_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i]
```

```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

In [ ]:

```
plot_colors = list('rgbkymc')
test_class_distribution.plot(kind = 'bar', color = plot_colors)
plt.xlabel('Class')
plt.ylabel('Data Points per Class')
plt.title('Distribution of yi in Test data')
plt.grid()
plt.show()
```



In [ ]:

```
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[
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
```

In [3]:

```python
def plot_confusion_matrix(test_y, predict_y):
  C = confusion_matrix(test_y, predict_y)
  print(f"Number of misclassified points: {(len(test_y)-np.trace(C))/len(test_y)*100
  # Recall Matrix
  A = (((C.T)/(C.sum(axis=1))).T)
  # Precision Matrix
  B = (C/C.sum(axis=0))

  labels = [1,2,3,4,5,6,7,8,9]
  cmap = sns.light_palette('green')

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

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

Machine Learning Models

Machine Learning Models on Byte Files

Random Model

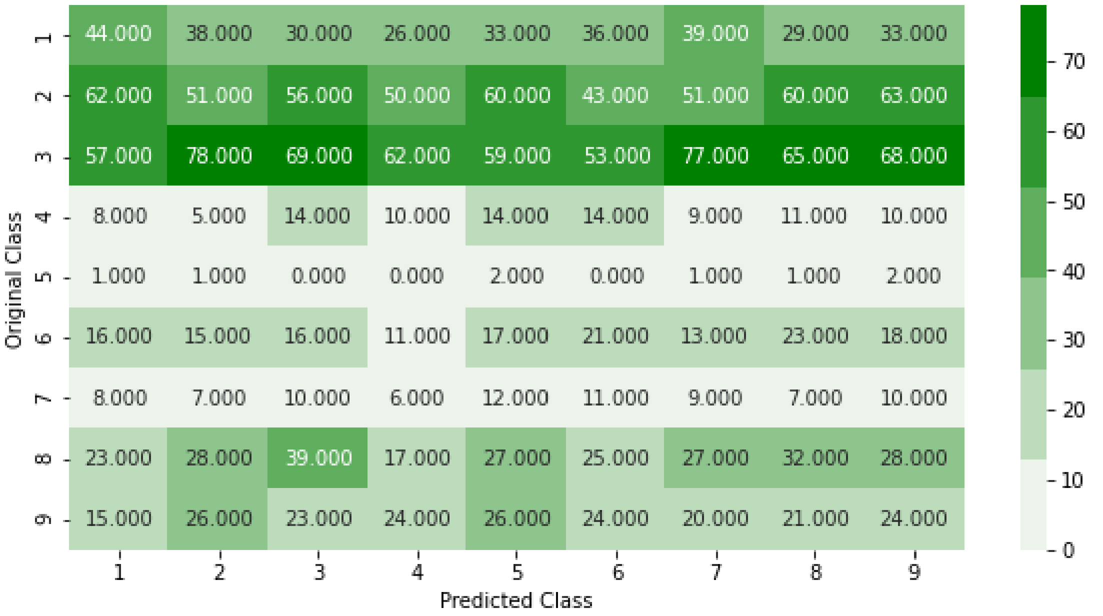
Generate 9 numbers and their sum should be 1.

In [ ]:

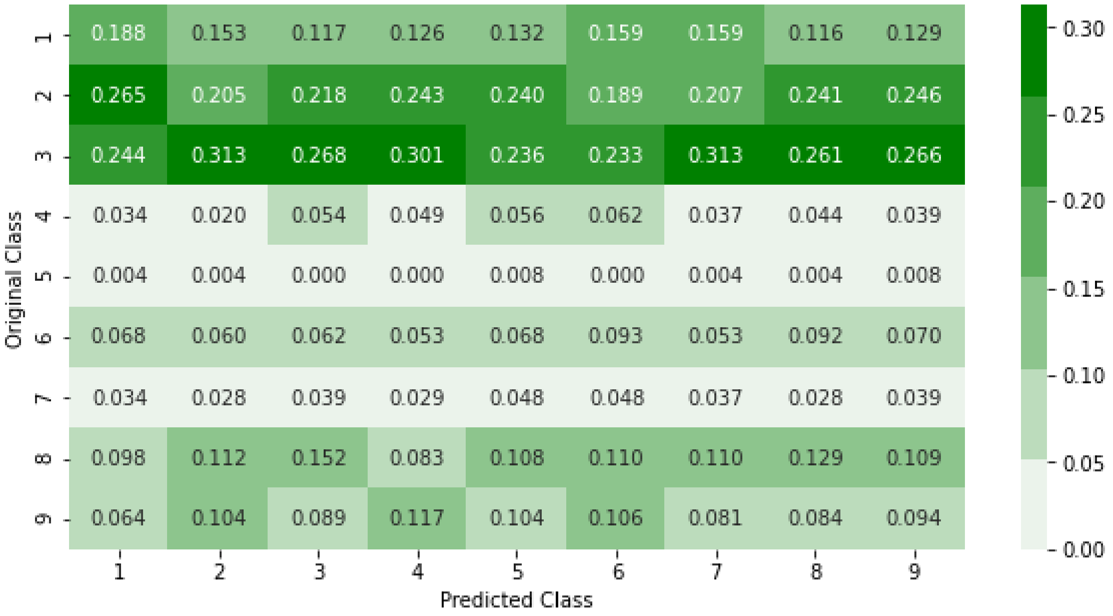
```python
test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

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(f"Log Loss on Cross Validation Data using Random Model: {log_loss(y_cv, cv_pre

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(f"Log Loss on Test Data using Random Model: {log_loss(y_test, test_predicted_y

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

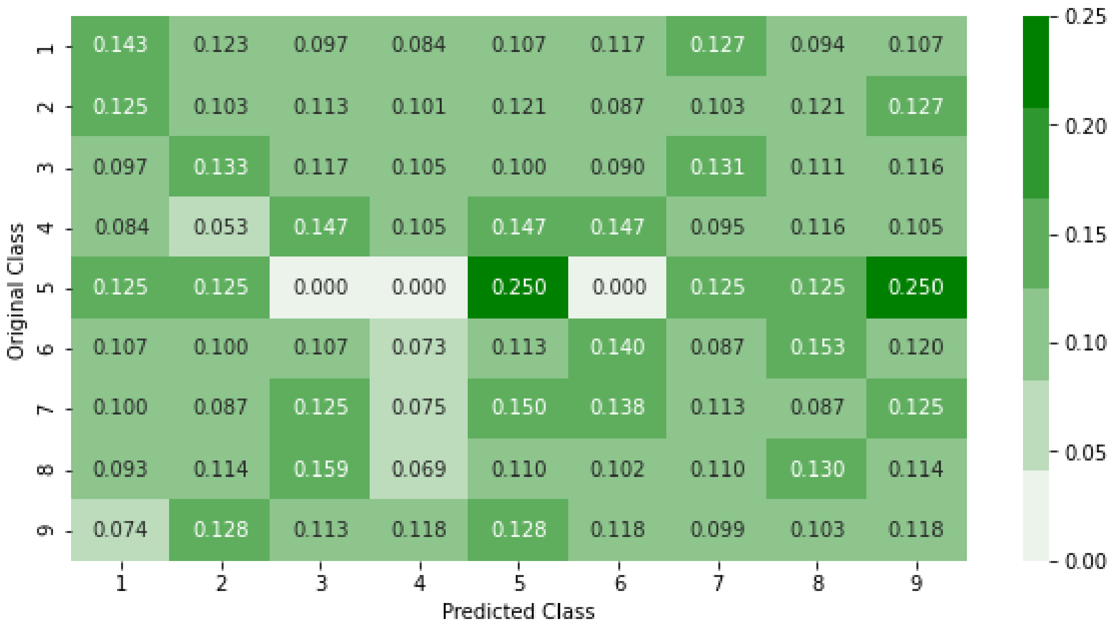
```
Log Loss on Cross Validation Data using Random Model: 2.4923939091527463
Log Loss on Test Data using Random Model: 2.439751753254197
Number of misclassified points: 87.94848206071757
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```



```
-------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



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



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

K Nearest Neighbour Classification

In [ ]:

```
alpha = [x for x in range(1, 15, 2)]

cv_log_error_array = []

for i in alpha:
  k_cfl = KNeighborsClassifier(n_neighbors = i)
  k_cfl.fit(X_train, y_train)
  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 =
```

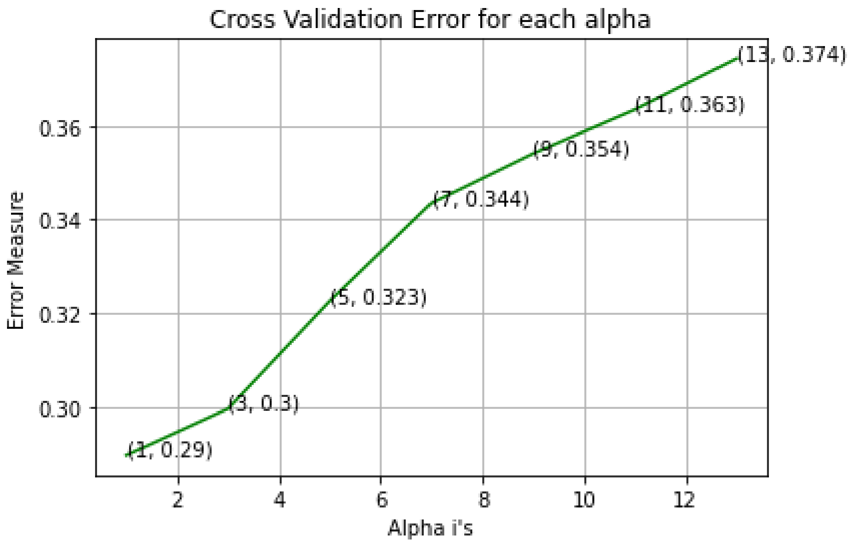
In [ ]:

```
for i in range(len(cv_log_error_array)):
  print(f"Log Loss for k  = {alpha[i]} is {cv_log_error_array[i]}")
```

```
Log Loss for k  = 1 is 0.28968574407859427
Log Loss for k  = 3 is 0.29959449528358967
Log Loss for k  = 5 is 0.3225576557920493
Log Loss for k  = 7 is 0.34353725584043776
Log Loss for k  = 9 is 0.35410061064364745
Log Loss for k  = 11 is 0.363440646255275O6
Log Loss for k  = 13 is 0.37434924493862076
```

In [ ]:

```
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()
```



In [ ]:

```
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)
```

Out[ ]:

```
CalibratedClassifierCV(base_estimator=KNeighborsClassifier(algorithm='auto',
                                                           leaf_size=30,
                                                           metric='minkowski',
                                                           metric_params=None,
                                                           n_jobs=None,
                                                           n_neighbors=1, p=2,
                                                           weights='uniform'),
                       cv=None, method='sigmoid')
```

In [ ]:

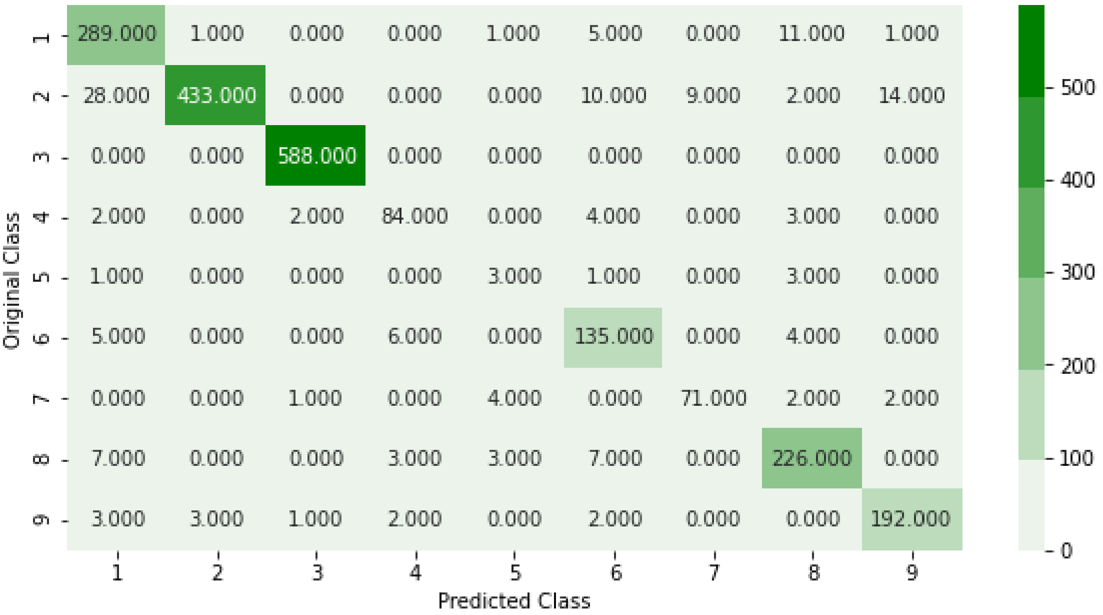
```
predict_y = sig_clf.predict_proba(X_train)
print(f"For value of best alpha: {alpha[best_alpha]}, the train log loss is {log_los
predict_y = sig_clf.predict_proba(X_cv)
print(f"For value of best alpha: {alpha[best_alpha]}, the cv log loss is {log_loss(y
predict_y = sig_clf.predict_proba(X_test)
print(f"For value of best alpha: {alpha[best_alpha]}, the test log loss is {log_loss
```

```
For value of best alpha: 1, the train log loss is 0.09838115619905251
For value of best alpha: 1, the cv log loss is 0.28968574407859427
For value of best alpha: 1, the test log loss is 0.32779866520344664
```

In [ ]:

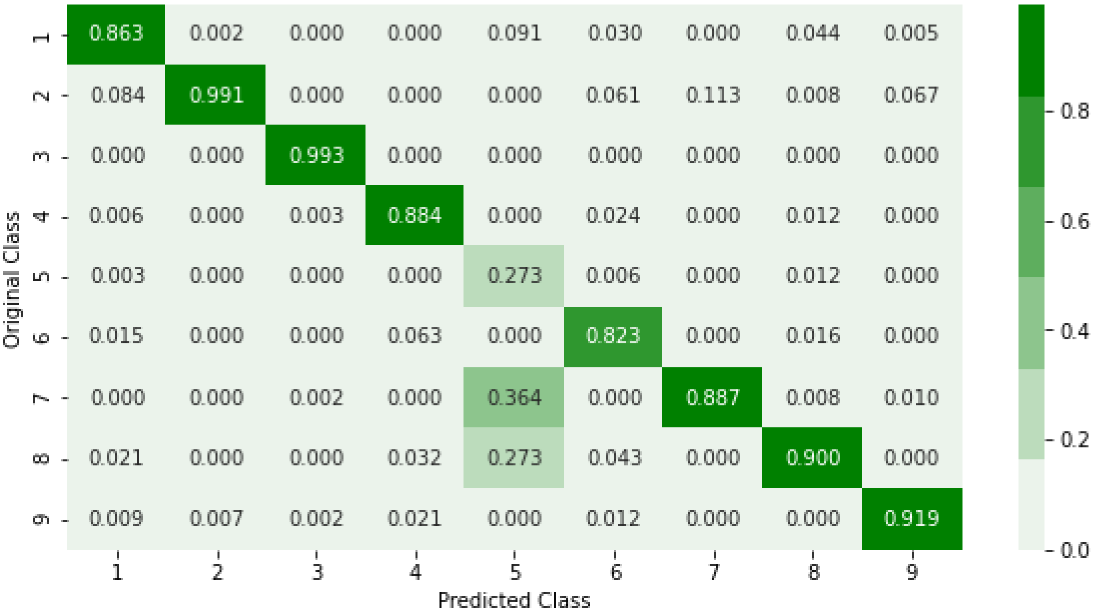
```
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
Number of misclassified points: 7.0377184912603505
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```



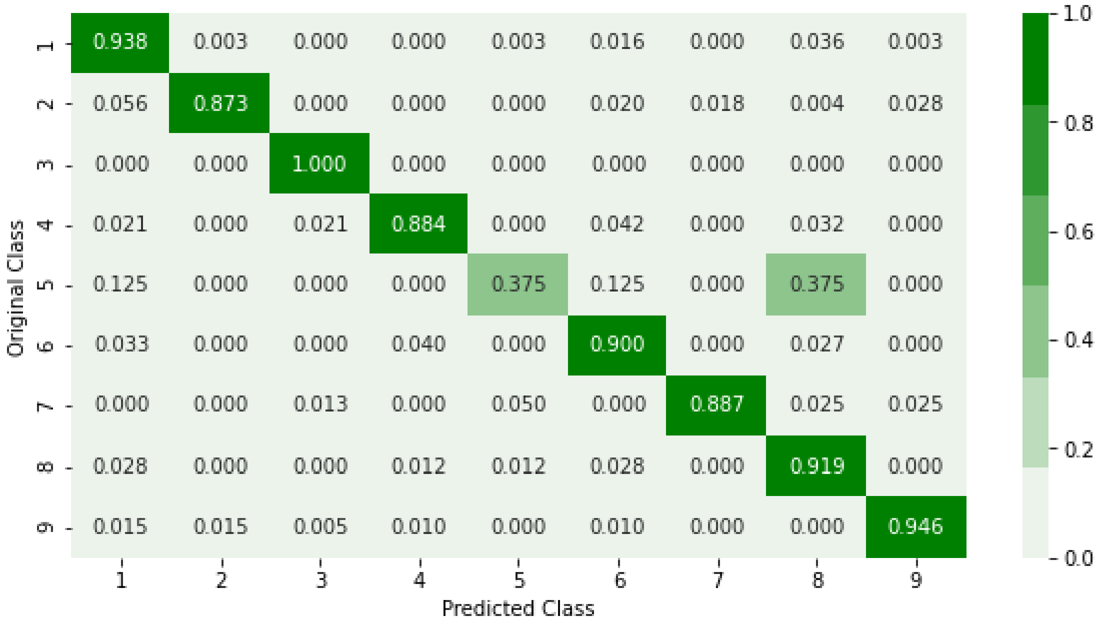
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	289.000	1.000	0.000	0.000	1.000	5.000	0.000	11.000	1.000
2	28.000	433.000	0.000	0.000	0.000	10.000	9.000	2.000	14.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	2.000	0.000	2.000	84.000	0.000	4.000	0.000	3.000	0.000
5	1.000	0.000	0.000	0.000	3.000	1.000	0.000	3.000	0.000
6	5.000	0.000	0.000	6.000	0.000	135.000	0.000	4.000	0.000
7	0.000	0.000	1.000	0.000	4.000	0.000	71.000	2.000	2.000
8	7.000	0.000	0.000	3.000	3.000	7.000	0.000	226.000	0.000
9	3.000	3.000	1.000	2.000	0.000	2.000	0.000	0.000	192.000

```
-------------------------------------------------- Precision Matrix -----------------
---------------------------------
```



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.863	0.002	0.000	0.000	0.091	0.030	0.000	0.044	0.005
2	0.084	0.991	0.000	0.000	0.000	0.061	0.113	0.008	0.067
3	0.000	0.000	0.993	0.000	0.000	0.000	0.000	0.000	0.000
4	0.006	0.000	0.003	0.884	0.000	0.024	0.000	0.012	0.000
5	0.003	0.000	0.000	0.000	0.273	0.006	0.000	0.012	0.000
6	0.015	0.000	0.000	0.063	0.000	0.823	0.000	0.016	0.000
7	0.000	0.000	0.002	0.000	0.364	0.000	0.887	0.008	0.010
8	0.021	0.000	0.000	0.032	0.273	0.043	0.000	0.900	0.000
9	0.009	0.007	0.002	0.021	0.000	0.012	0.000	0.000	0.919

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



	1	2	3	4	5	6	7	8	9
1	0.938	0.003	0.000	0.000	0.003	0.016	0.000	0.036	0.003
2	0.056	0.873	0.000	0.000	0.000	0.020	0.018	0.004	0.028
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.021	0.000	0.021	0.884	0.000	0.042	0.000	0.032	0.000
5	0.125	0.000	0.000	0.000	0.375	0.125	0.000	0.375	0.000
6	0.033	0.000	0.000	0.040	0.000	0.900	0.000	0.027	0.000
7	0.000	0.000	0.013	0.000	0.050	0.000	0.887	0.025	0.025
8	0.028	0.000	0.000	0.012	0.012	0.028	0.000	0.919	0.000
9	0.015	0.015	0.005	0.010	0.000	0.010	0.000	0.000	0.946

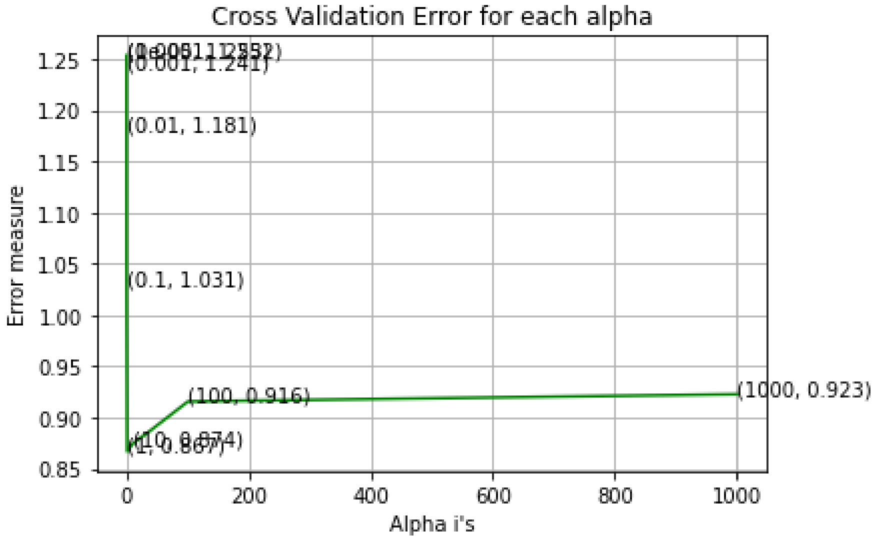
```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Logistic Regression

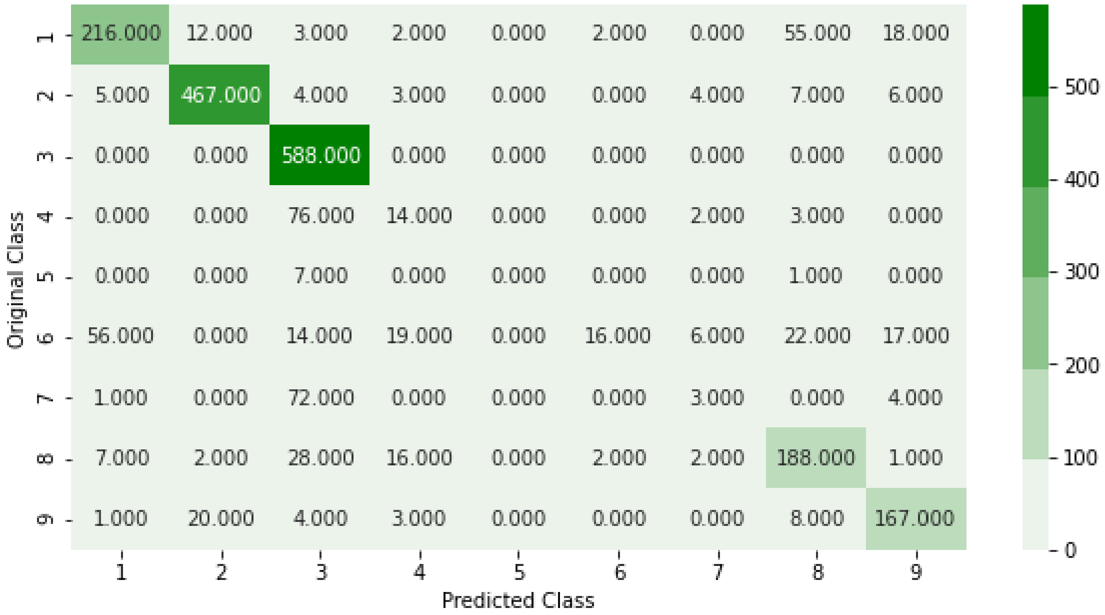
In [ ]:

```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,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_, e

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.class
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, e
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.2534802986311433
log_loss for c =  0.0001 is 1.252204823886248
log_loss for c =  0.001 is 1.2413169060876394
log_loss for c =  0.01 is 1.1808822301579556
log_loss for c =  0.1 is 1.0306299931440208
log_loss for c =  1 is 0.8669011409737292
log_loss for c =  10 is 0.8742243777878516
log_loss for c =  100 is 0.9157148668909443
log_loss for c =  1000 is 0.9227644967308084
```

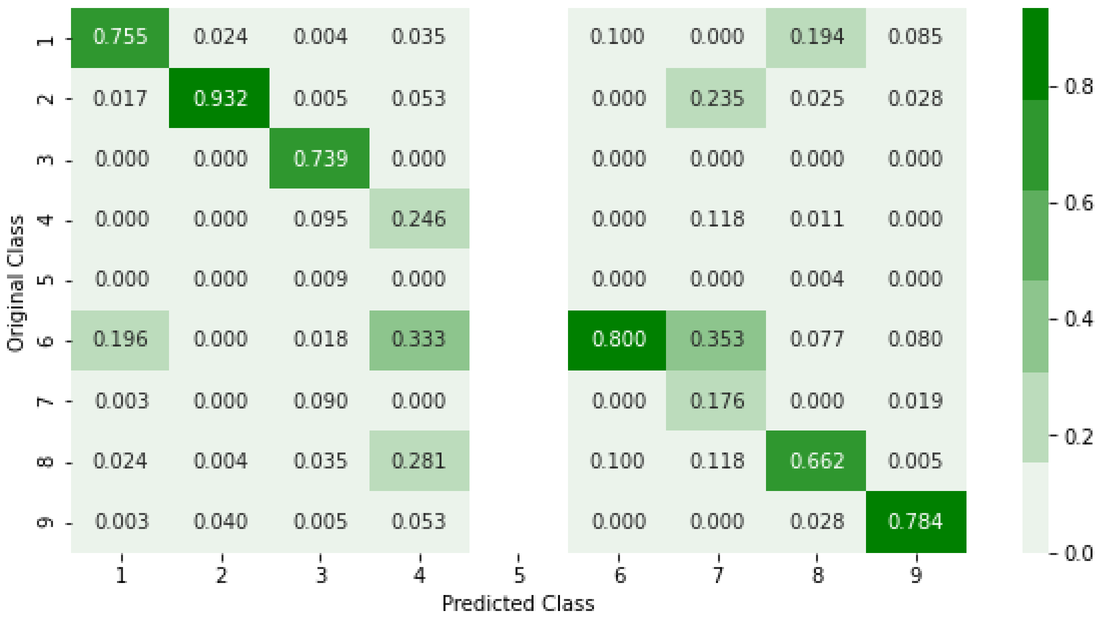


```
log loss for train data 0.8845064073858054
log loss for cv data 0.8669011409737292
log loss for test data 0.887893568833425
Number of misclassified points: 23.689052437902482
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```

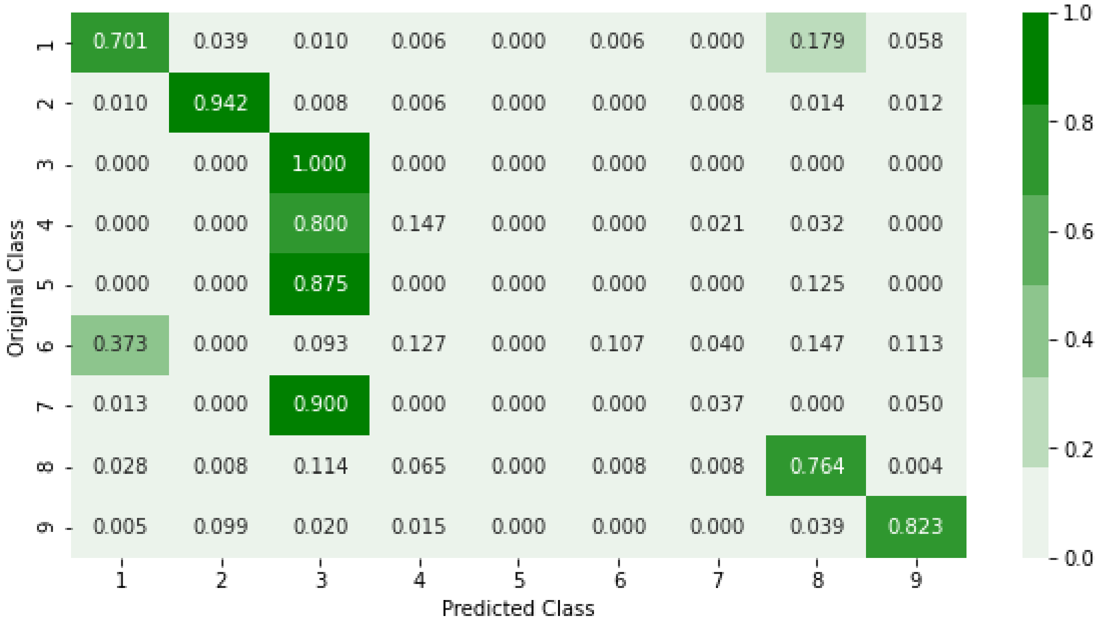


	1	2	3	4	5	6	7	8	9
1	216.000	12.000	3.000	2.000	0.000	2.000	0.000	55.000	18.000
2	5.000	467.000	4.000	3.000	0.000	0.000	4.000	7.000	6.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	76.000	14.000	0.000	0.000	2.000	3.000	0.000
5	0.000	0.000	7.000	0.000	0.000	0.000	0.000	1.000	0.000
6	56.000	0.000	14.000	19.000	0.000	16.000	6.000	22.000	17.000
7	1.000	0.000	72.000	0.000	0.000	0.000	3.000	0.000	4.000
8	7.000	2.000	28.000	16.000	0.000	2.000	2.000	188.000	1.000
9	1.000	20.000	4.000	3.000	0.000	0.000	0.000	8.000	167.000

```
-------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



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



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

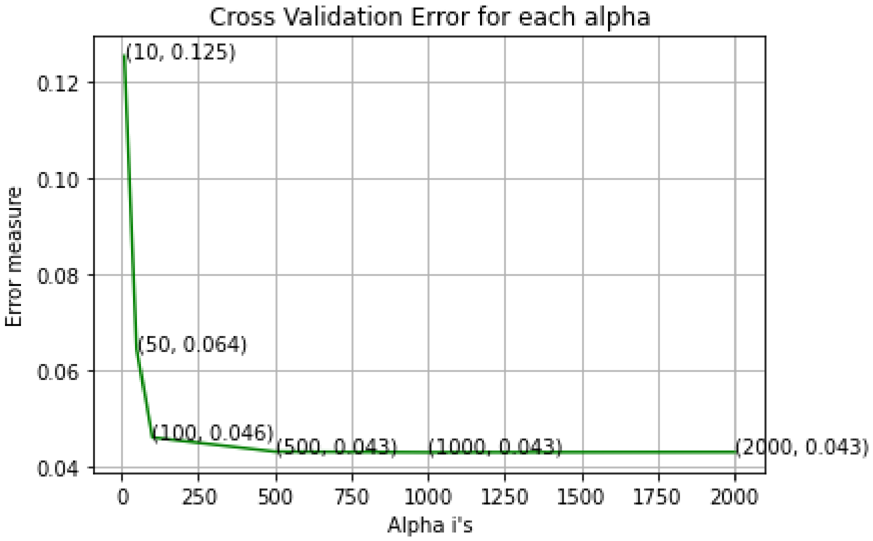
XGBoost

In [ ]:

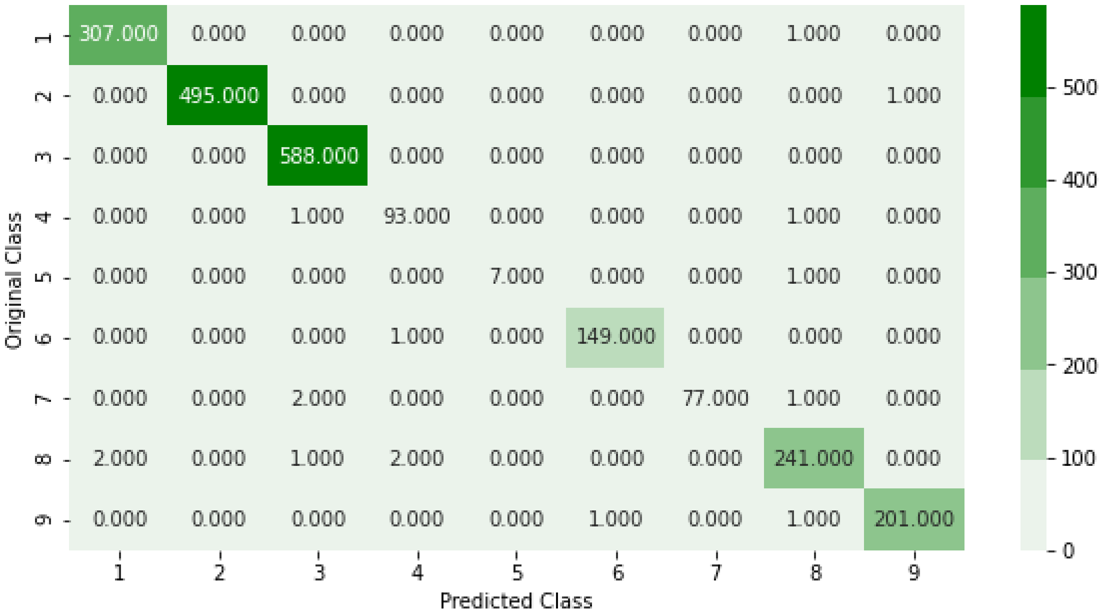
```
alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv,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

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
```

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

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

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

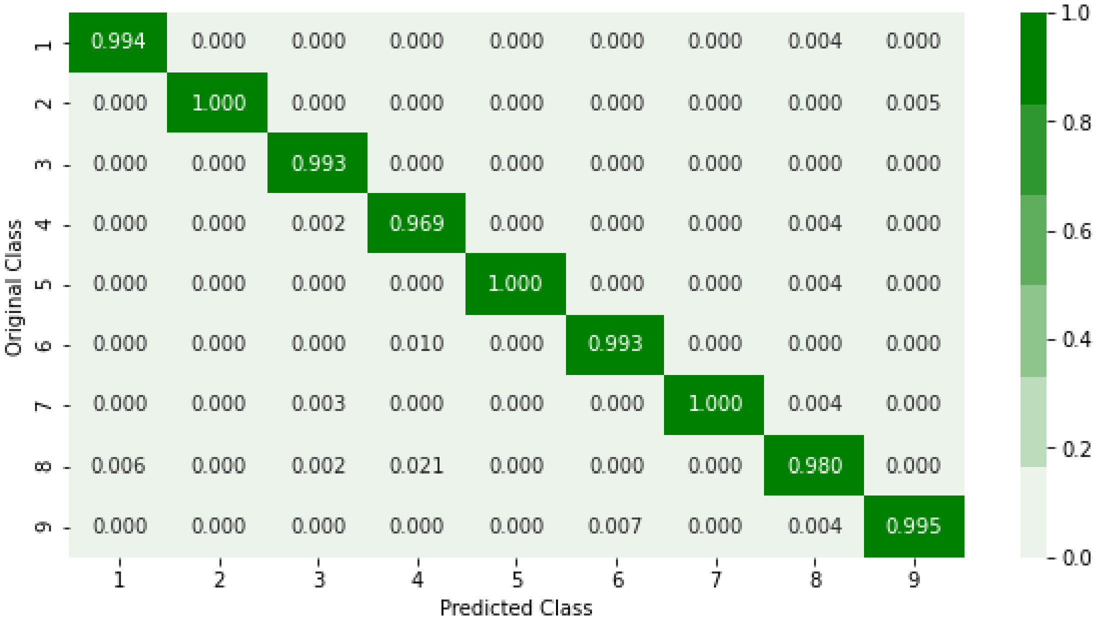
```
log_loss for c =  10 is 0.1253381117814775
log_loss for c =  50 is 0.06441239171630905
log_loss for c =  100 is 0.0460672975526283
log_loss for c =  500 is 0.04311691555092733
log_loss for c =  1000 is 0.04296207899450181
log_loss for c =  2000 is 0.04301768653203457
```



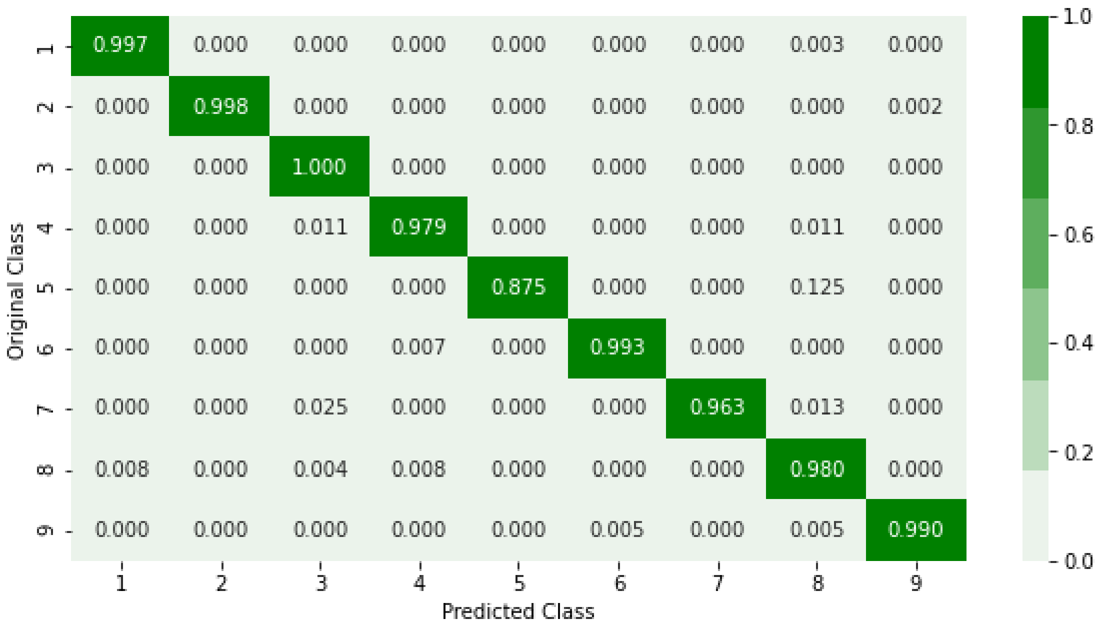
```
For values of best alpha =  1000 The train log loss is: 0.01842583998536047
For values of best alpha =  1000 The cross validation log loss is: 0.042962078994501
81
For values of best alpha =  1000 The test log loss is: 0.04934209459453473
Number of misclassified points: 0.7359705611775529
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```



```
-------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



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



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

Saving and Loading XGBoost Models

In [ ]:

```
pickle.dump(x_cfl, open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware 
pickle.dump(sig_clf, open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malwar
```

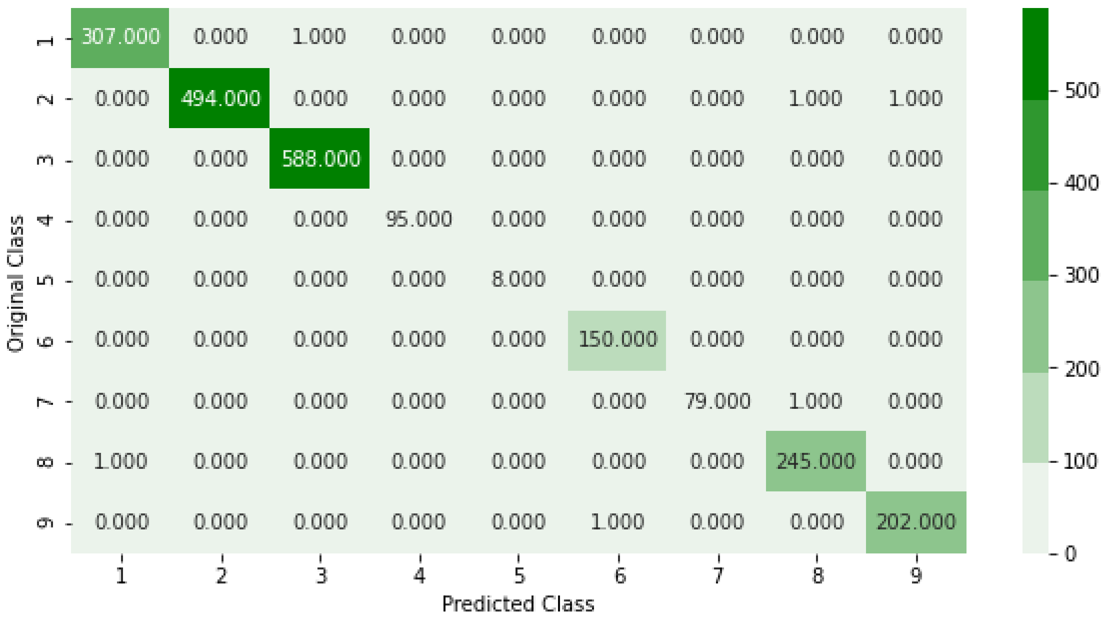
In [ ]:

```
sig_clf = pickle.load(open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malwa
```

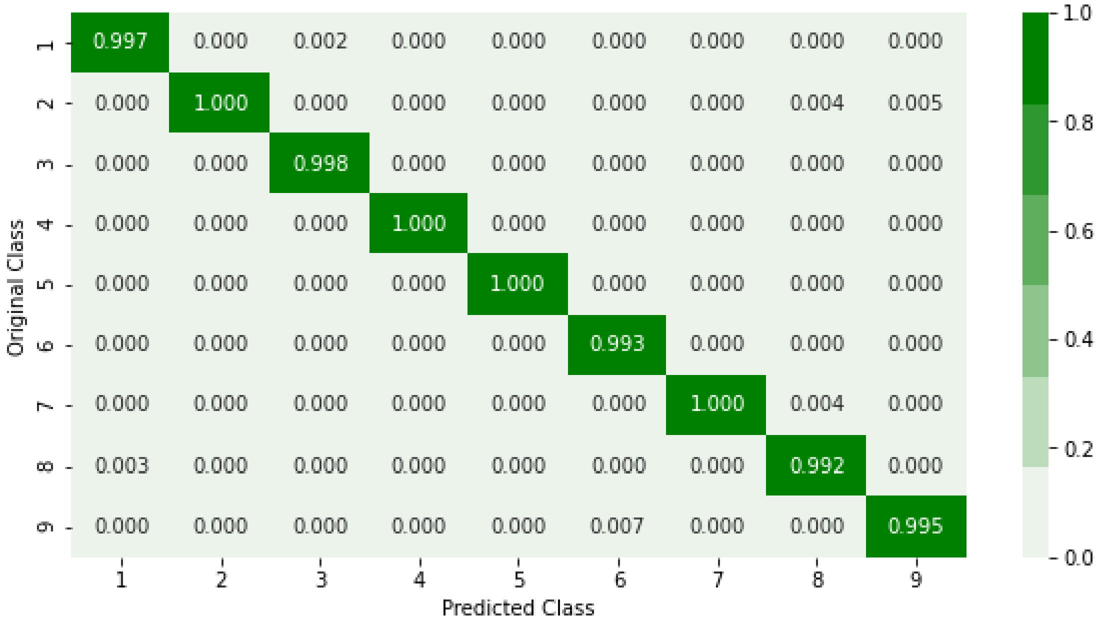
In [ ]:

```
alpha=[10,50,100,500,1000,2000]
best_alpha = 4

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

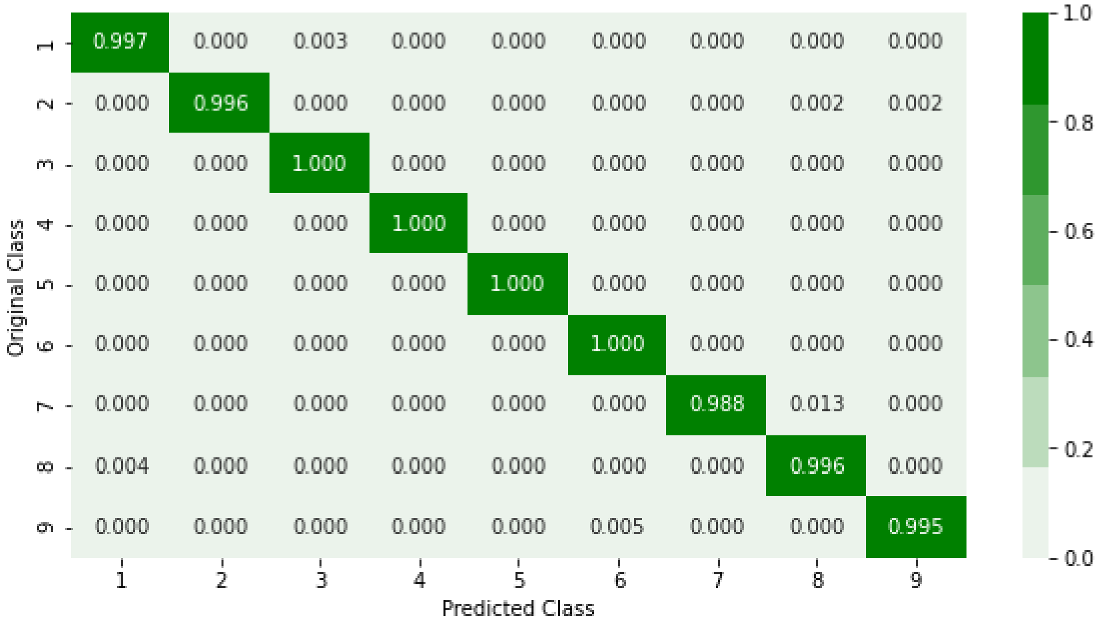
```
For values of best alpha =  1000 The train log loss is: 0.026898008896593332
For values of best alpha =  1000 The cross validation log loss is: 0.034281037122212
525
For values of best alpha =  1000 The test log loss is: 0.029182203627785227
Number of misclassified points: 0.27598896044158233
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```



```
-------------------------------------------------- Precision Matrix ----------------
--------------------------------
```



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



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

ASM Features

Feature Extraction from ASM Files

OPCode Features

There is about 150GB of Data that needs to be processed.

In [ ]:

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

source='/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware Detection/train'
files = os.listdir('train')
data=range(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'first')
    elif i%5==1:
        shutil.move(source+files[data[i]],'second')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'third')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fifth')
```

In [ ]:

```
#http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html
def firstprocess():
    #The prefixes tells about the segments that are present in the asm files
    #There are 450 segments(approx) present in all asm files.
    #this prefixes are best segments that gives us best values.

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.ed
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

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

def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.ed
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'in
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()

def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.ed
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'in
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('third')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('third/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()

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

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


def main():
    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
    p3=Process(target=thirdprocess)
    p4=Process(target=fourthprocess)
    p5=Process(target=fifthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

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

In [ ]:

```
dfasm = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware Dete
Y.columns = ['ID','Class']
result_asm = pd.merge(dfasm, Y, on = 'ID', how = 'left')
result_asm.head()
```

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	0
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	0

File Size Feature

In [ ]:

```
asmFiles = '/content/drive/Shareddrives/colab/asmFiles/'
```

In [ ]:

```
files = os.listdir(asmFiles)
filenames = Y['ID'].tolist()
class_Y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []
```

In [ ]:

```
for file in files:
  statinfo = os.stat(asmFiles + file)
  file = file.split('.')[0]
  if any(file == filename for filename in filenames):
    i = filenames.index(file)
    class_bytes.append(class_Y[i])
    sizebytes.append(statinfo.st_size/(1024.0*1024.0))
    fnames.append(file)
asm_size_byte = pd.DataFrame({'ID':fnames, 'Size':sizebytes, 'Class':class_bytes})
asm_size_byte.head()
```

Out[ ]:

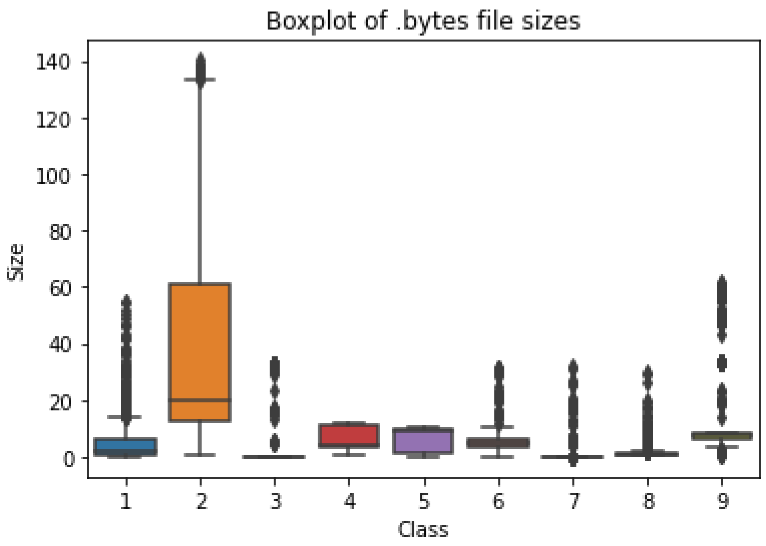
	ID	Size	Class
0	ec5wGtnrTOjUmXx3QqKL	83.731307	2

	ID	Size	Class
1	ECjwlxQoZPl8a61NvByq	0.159330	3
2	ecFS36DyA9qlifz0NCub	36.555618	2
3	ecjUHgzDC7ryXu2sNwJf	1.092697	9
4	ECiA7GPQj6MNZtSJvRqL	0.918860	8

Distribution of .asm file sizes

In [ ]:

```
ax = sns.boxplot(x="Class", y="Size", data=asm_size_byte)
plt.title("Boxplot of .bytes file sizes")
plt.show()
```



In [ ]:

```
# Adding 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
result_asm.head()
```

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

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	0
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	0

In [ ]:

```
# Normalizing Columns
def normalize(df):
  result_copy = 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()
```

```
        result_copy[feature_name] = (df[feature_name] - min_value)/(max_value - min_va
    return result_copy
```

In [ ]:

```
result_asm = normalize(result_asm)
result_asm.head()
```

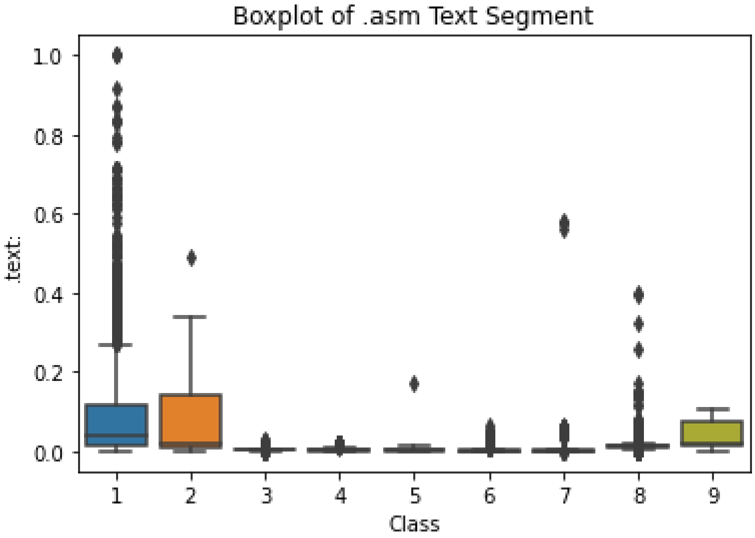
Out[ ]:

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

Univariate Analysis on .asm file features

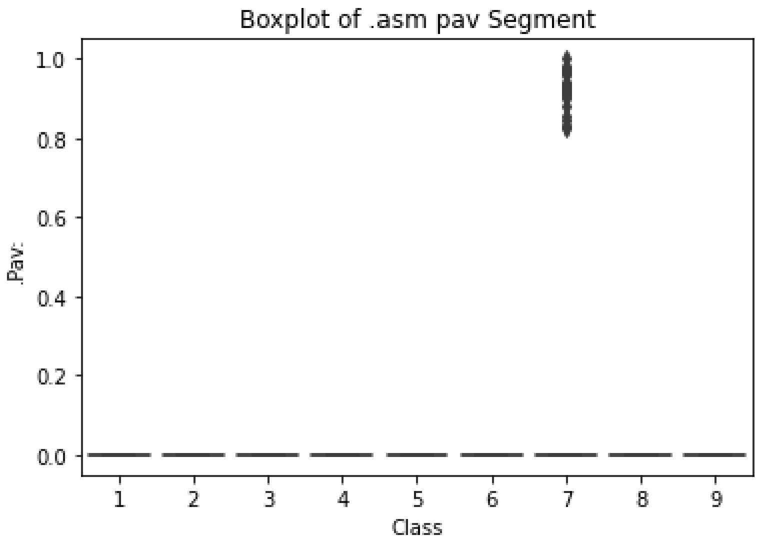
In [ ]:

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



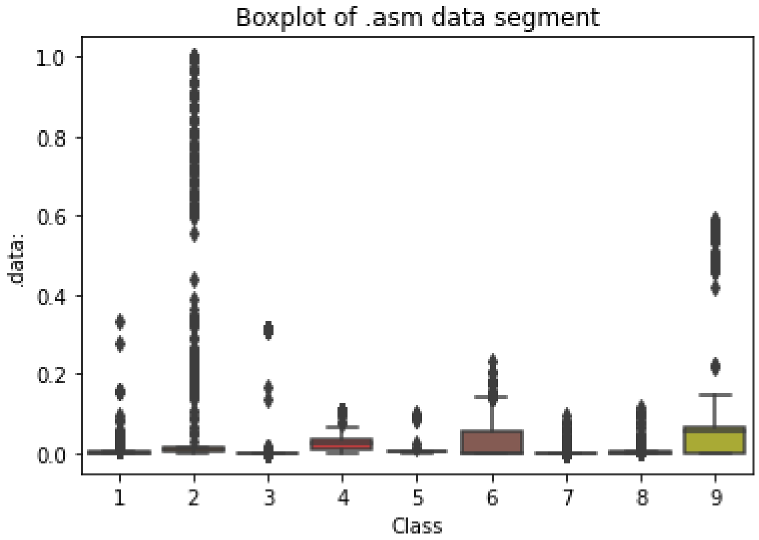
In [ ]:

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



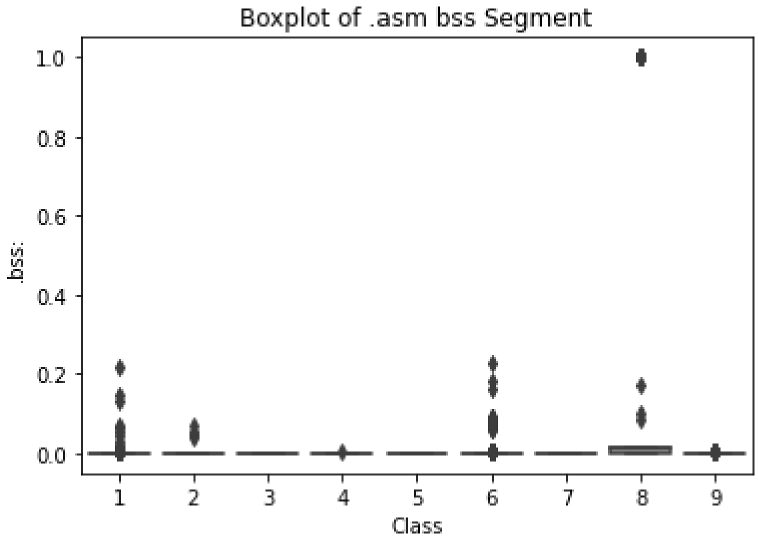
In [ ]:

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



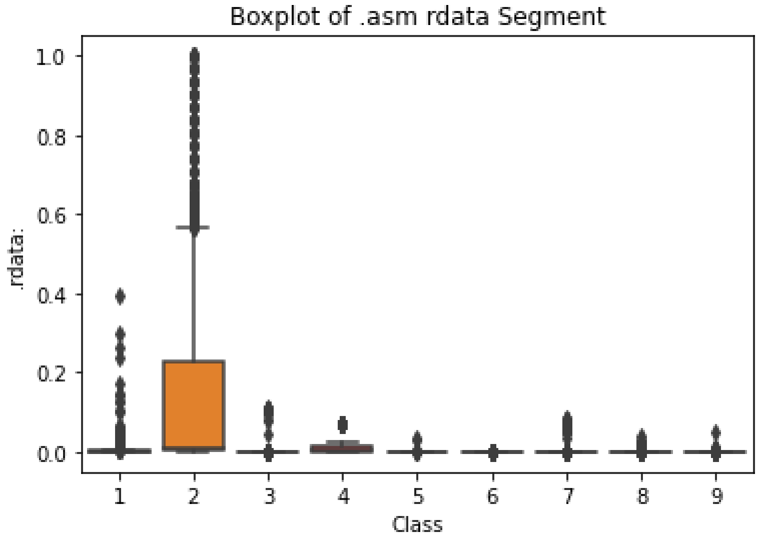
In [ ]:

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



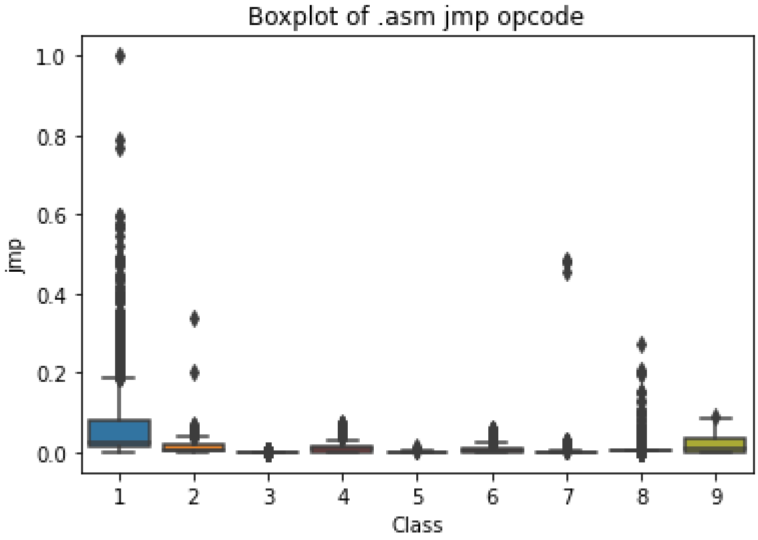
In [ ]:

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



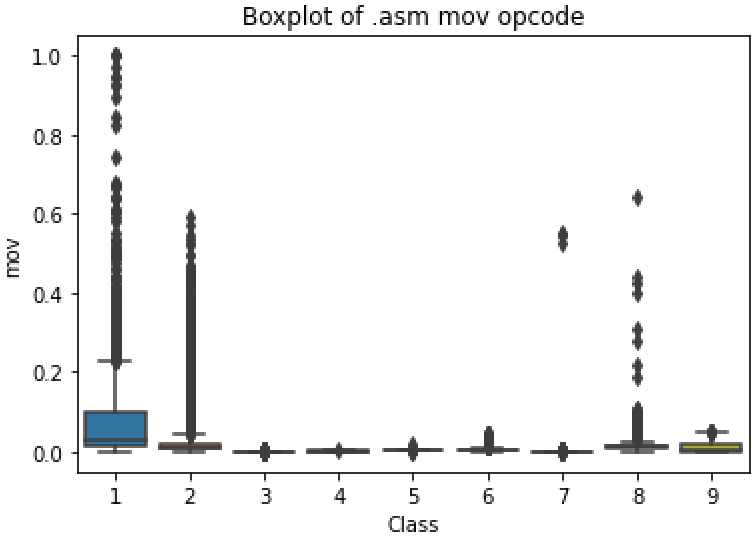
In [ ]:

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



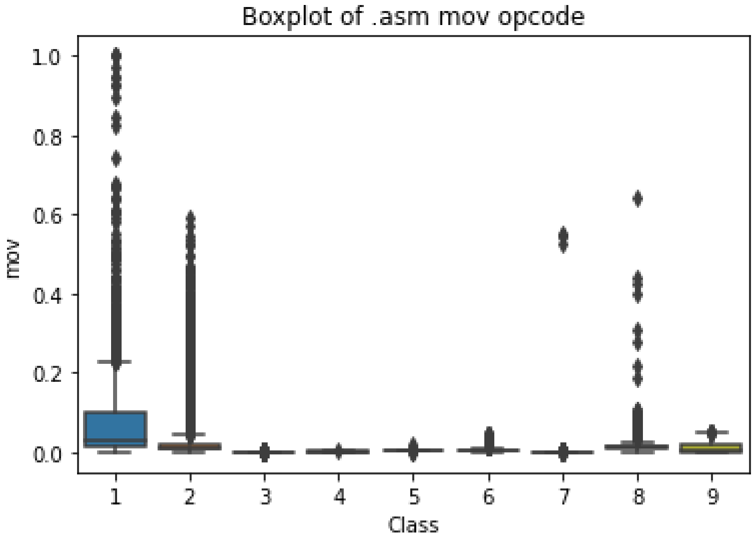
In [ ]:

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



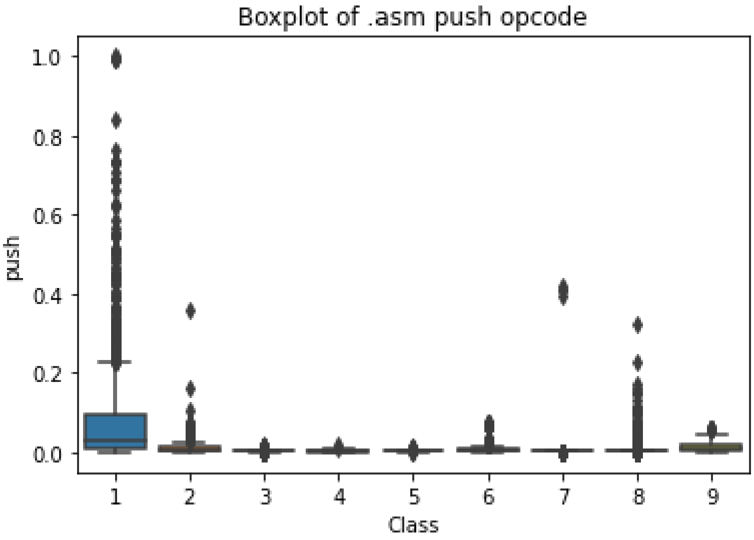
In [ ]:

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



In [ ]:

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



Pixel Density Feature

In [ ]:

```
import array
# def get_800_pixel(source):
#     if source.endswith(".asm"):
#         file=open(source,"rb")
#         ln=os.path.getsize(source)
#         width=int(ln*0.5)
#         rem=ln%width
#         a=array.array("B")# unit8 array
#         a.fromfile(file,ln-rem)
#         file.close()
#         return np.array(list(a[:800]))
pxl_colmns=["ID"]+["pxl_"+str(i) for i in range(800)]
def get_800_pixel(source):
    if source.endswith(".asm"):
        source="asmFiles/"+source
        file=open(source,"rb")
        ln=os.path.getsize(source)
        width=int(ln*0.5)
        rem=ln%width
        a=array.array("B")# unit8 array
        a.fromfile(file,ln-rem)
        file.close()
        g=np.reshape(a,(int(len(a)/width),width))
        g=np.uint8(g)
        #print(800-len(g[0][:300]))
        if len(g[0])>800:
            return np.array(g[0][:800])
        else:
            return np.pad(g[0],(0,800-len(g[0])),mode="constant",constant_values=(0,

files = os.listdir('asmFiles')
pxl_intensity=[]
for file in tqdm(files):
    temp=get_800_pixel(file)
    temp=np.concatenate([[file.split(".")[0]],temp])
#     print(temp)
    pxl_intensity.append(temp)

pxl_intensity_df=pd.DataFrame(pxl_intensity,columns=pxl_colmns)
```

In [ ]:

```
def pixel_density(file):
  """
  http://sarvamblog.blogspot.ca/2014/08/supervised-classification-with-k-fold.html
  Padding: https://stackoverflow.com/questions/45422000/add-n-zeros-to-the-end-of-an
  800 asm features: https://www.kaggle.com/c/malware-classification/discussion/13490
  """
  f = open(file,"rb")
  ln = os.path.getsize(file)
  width = int(ln**0.5)
  rem = ln%width
  a = array("B")
  a.fromfile(f, ln-rem)
  f.close()
  g = np.reshape(a,(int(len(a)/width),width))
  g = np.uint8(g)
  if g.shape[0] > 800:
    return g[0][:800]
  else:
    # in case shape is less that 800, we add zeros at the end and return it
    return np.pad(g[0], (0, 800-g[0].shape[0]), 'constant', constant_values=(0,0))
```

In [ ]:

```
files = os.listdir(asmFiles)
pixel_densities = []

for file in tqdm(files):
  file_density = pixel_density(asmFiles+file)
  density_list = file_density.tolist()
  density_list.append(file.split(".")[0])
  pixel_densities.append(density_list)

cols = ["pixel_"+str(i) for i in range(800)]
cols.append("ID")

pixel_densities_df = pd.DataFrame(pixel_densities, columns = cols)
```

```
100%|██████████| 10868/10868 [2:11:47<00:00,  1.37it/s]
```

In [ ]:

```
pixel_densities_df.head()
```

Out[ ]:

	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	pixel_9	pixel_10	p
0	46	116	101	120	116	58	48	48	52	48	49	
1	72	69	65	68	69	82	58	48	48	52	48	
2	46	116	101	120	116	58	48	48	52	48	49	
3	72	69	65	68	69	82	58	48	48	52	48	
4	72	69	65	68	69	82	58	48	48	52	48	

5 rows × 801 columns

In [ ]:

```
pixel_densities_df.to_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malwar
```

In [ ]:

```
asm_features = result_asm.merge(pixel_densities_df, on = "ID")
```

In [ ]:

```
asm_features = asm_features.merge(asm_size_byte, on = "ID")
```

In [ ]:

```
asm_features.head()
```

Out[ ]:

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

5 rows × 856 columns

In [ ]:

```
asm_features = asm_features.rename(columns={"Size_y":"Size","Class_y":"Class"})
```

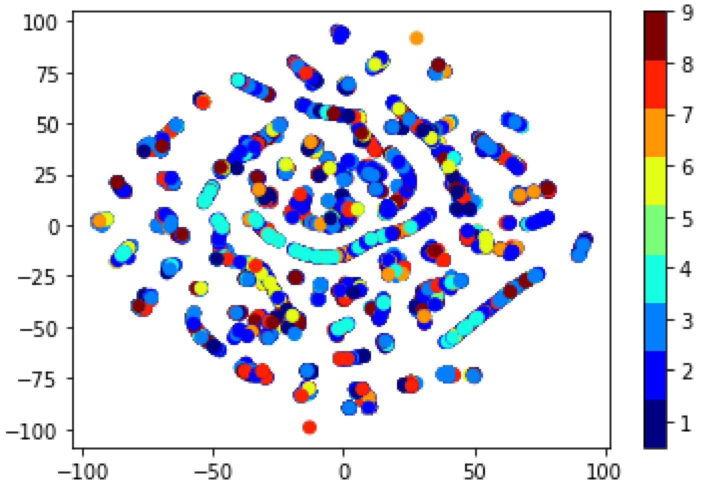
In [ ]:

```
asm_features.to_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware Dete
```

Multivariate Analysis on .asm file features

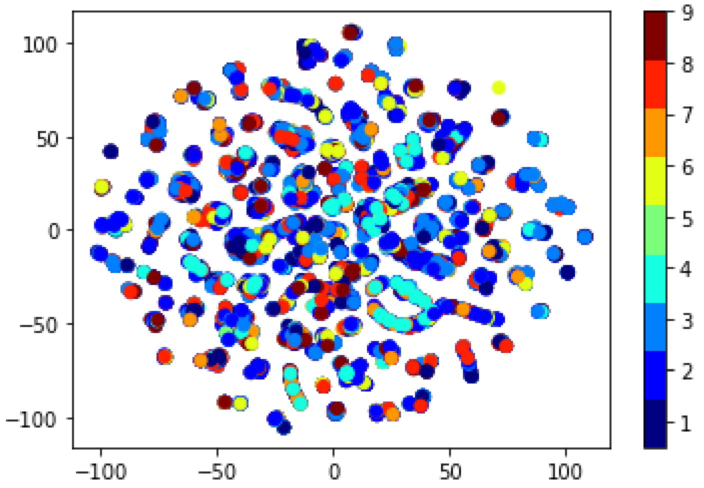
In [ ]:

```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(asm_features.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 [ ]:

```
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(asm_features.drop(['ID','Class', 'rtn', '.BSS:', '.CODE'
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 [ ]:

```
asm_features = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malwa
```

In [ ]:

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

In [ ]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,str
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train
```

In [ ]:

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

```
HEADER:       False
.text:        False
.Pav:         False
.idata:       False
.data:        False
              ...
pixel_796     False
pixel_797     False
pixel_798     False
pixel_799     False
Size          False
Length: 851, dtype: bool
```

Machine Learning Models on .asm Files

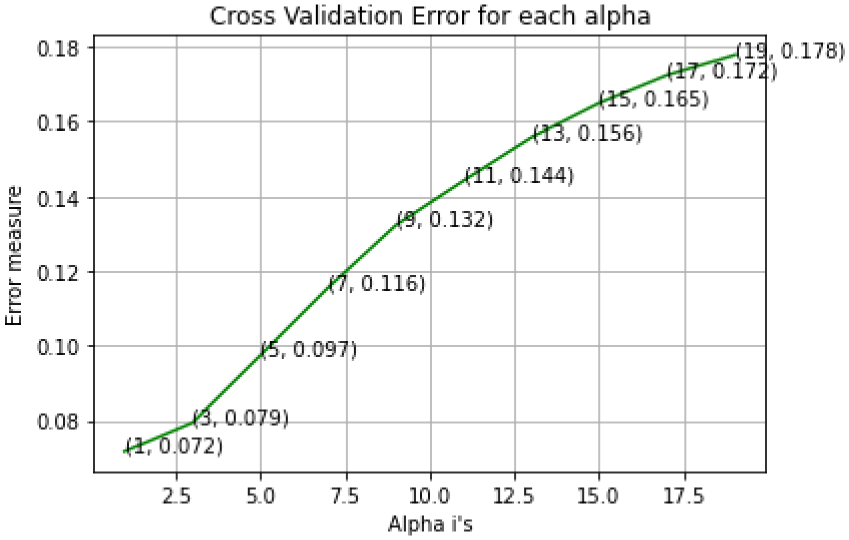
K Nearest Neighbours

In [ ]:

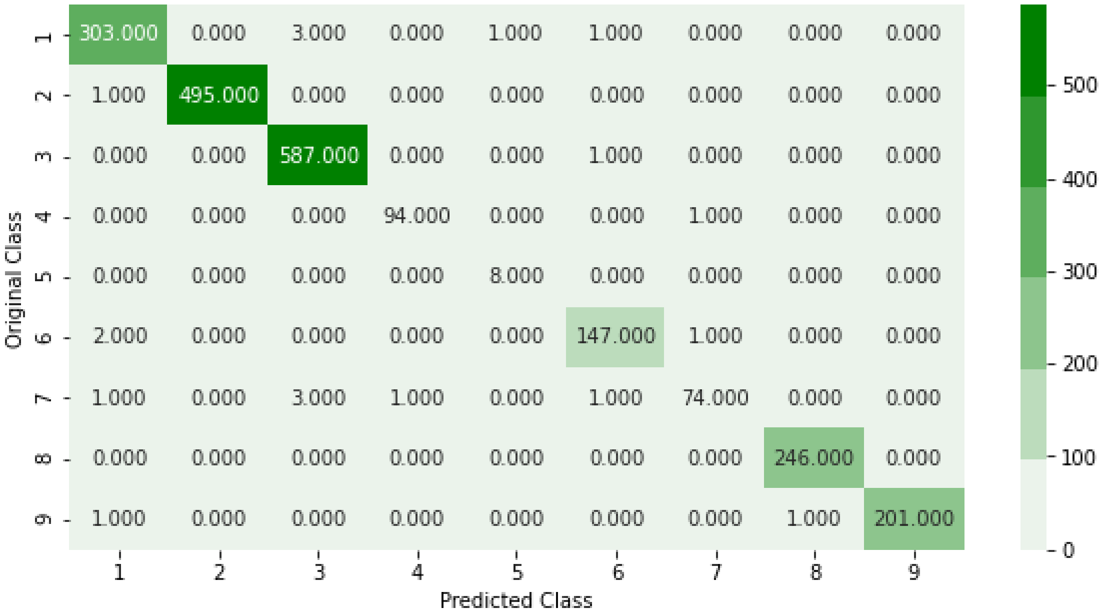
```
alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in tqdm(alpha):
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)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

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))
```

```
100%|██████████| 10/10 [04:21<00:00, 26.18s/it]
log_loss for k =  1 is 0.07178224902424325
log_loss for k =  3 is 0.07943892434927975
log_loss for k =  5 is 0.09739657140106162
log_loss for k =  7 is 0.11553184013617745
log_loss for k =  9 is 0.13227715719556218
log_loss for k =  11 is 0.14421089290814337
log_loss for k =  13 is 0.15577895981622286
log_loss for k =  15 is 0.16488814720262848
log_loss for k =  17 is 0.17232817334339184
log_loss for k =  19 is 0.1777601866820211
```

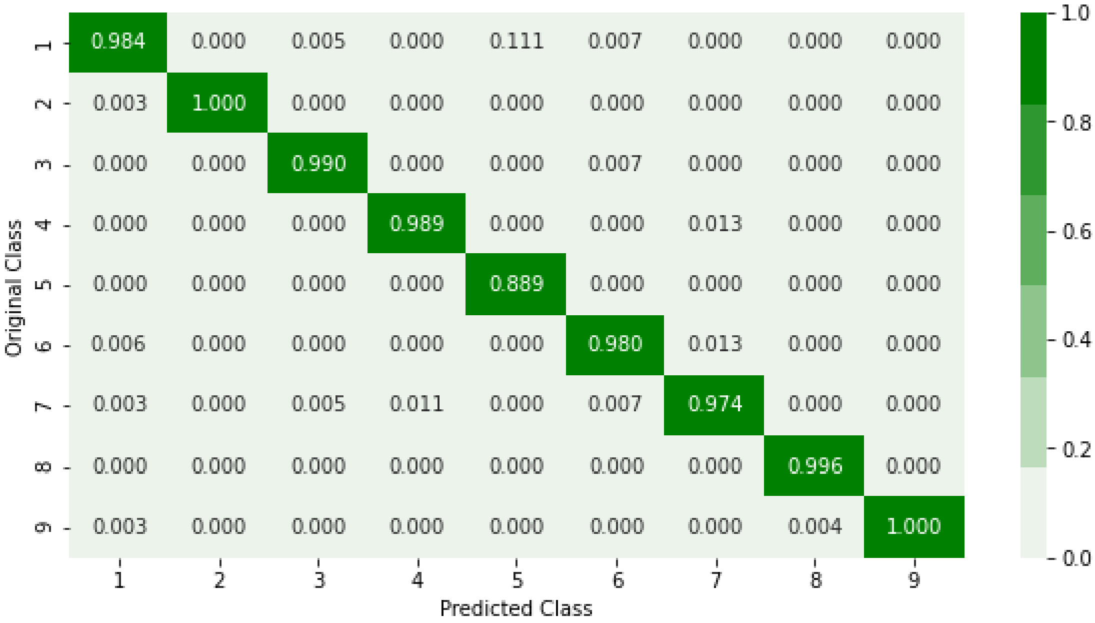


```
log loss for train data 0.021377106846947578
log loss for cv data 0.07178224902424325
log loss for test data 0.06298009640589095
Number of misclassified points: 0.8739650413983441
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```

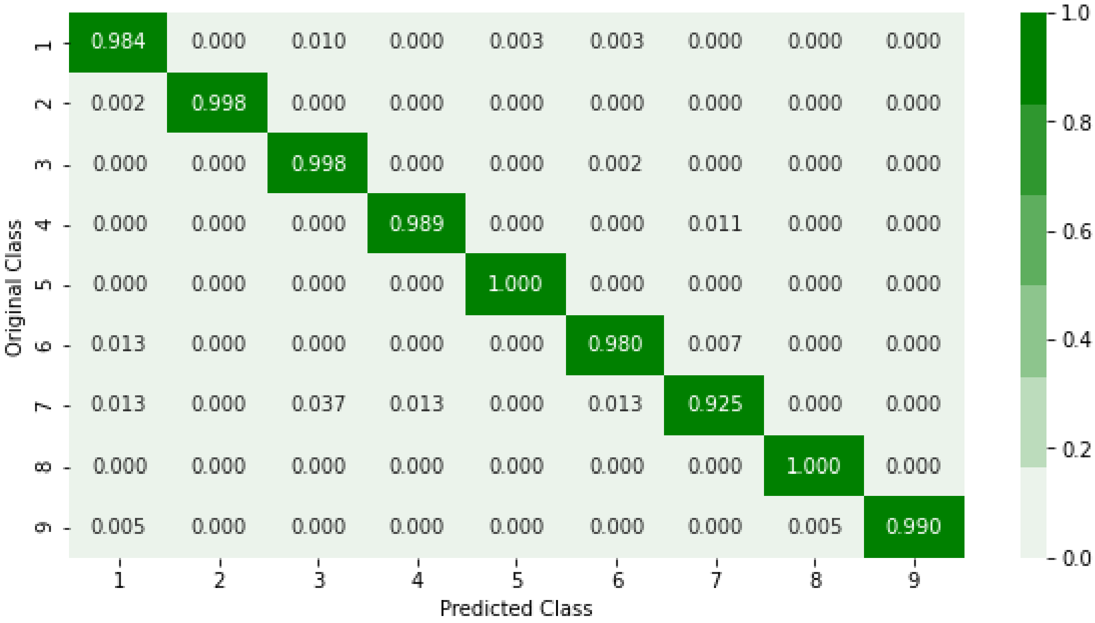


	1	2	3	4	5	6	7	8	9
1	303.000	0.000	3.000	0.000	1.000	1.000	0.000	0.000	0.000
2	1.000	495.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	587.000	0.000	0.000	1.000	0.000	0.000	0.000
4	0.000	0.000	0.000	94.000	0.000	0.000	1.000	0.000	0.000
5	0.000	0.000	0.000	0.000	8.000	0.000	0.000	0.000	0.000
6	2.000	0.000	0.000	0.000	0.000	147.000	1.000	0.000	0.000
7	1.000	0.000	3.000	1.000	0.000	1.000	74.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	246.000	0.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	201.000

```
-------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



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



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

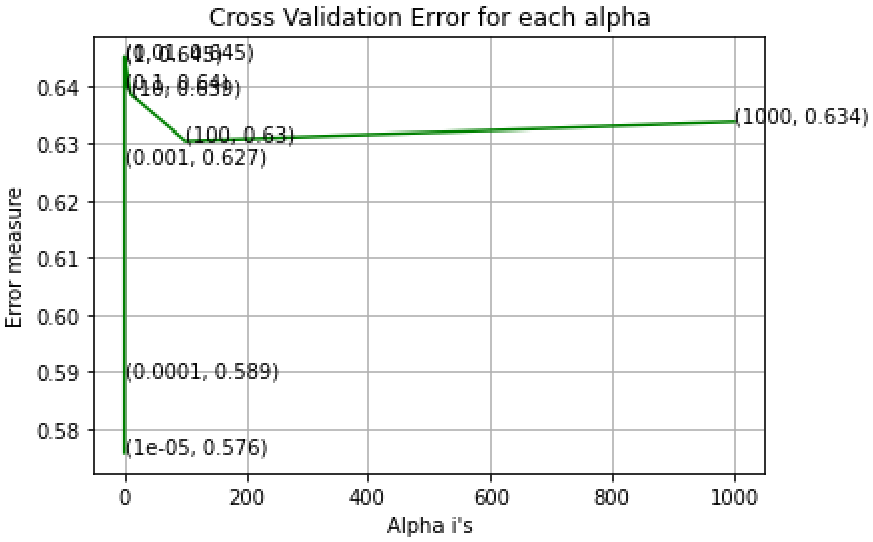
Logistic Regression

In [ ]:

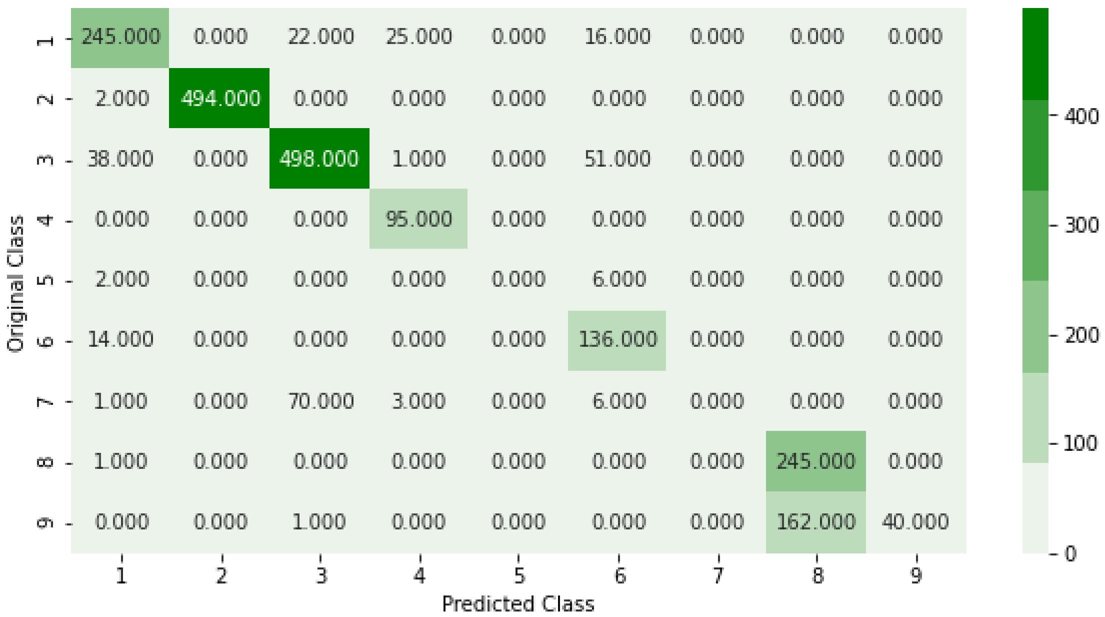
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in tqdm(alpha):
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, 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

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.
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classe
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.cl
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
100%|██████████| 9/9 [03:03<00:00, 20.41s/it]
log_loss for c =  1e-05 is 0.5757195173715258
log_loss for c =  0.0001 is 0.5891369024228066
log_loss for c =  0.001 is 0.6266026230744809
log_loss for c =  0.01 is 0.6451405393366427
log_loss for c =  0.1 is 0.6395575929305715
log_loss for c =  1 is 0.6447000836656482
log_loss for c =  10 is 0.6385718793141266
log_loss for c =  100 is 0.6304339591396597
log_loss for c =  1000 is 0.6336961757960645
```

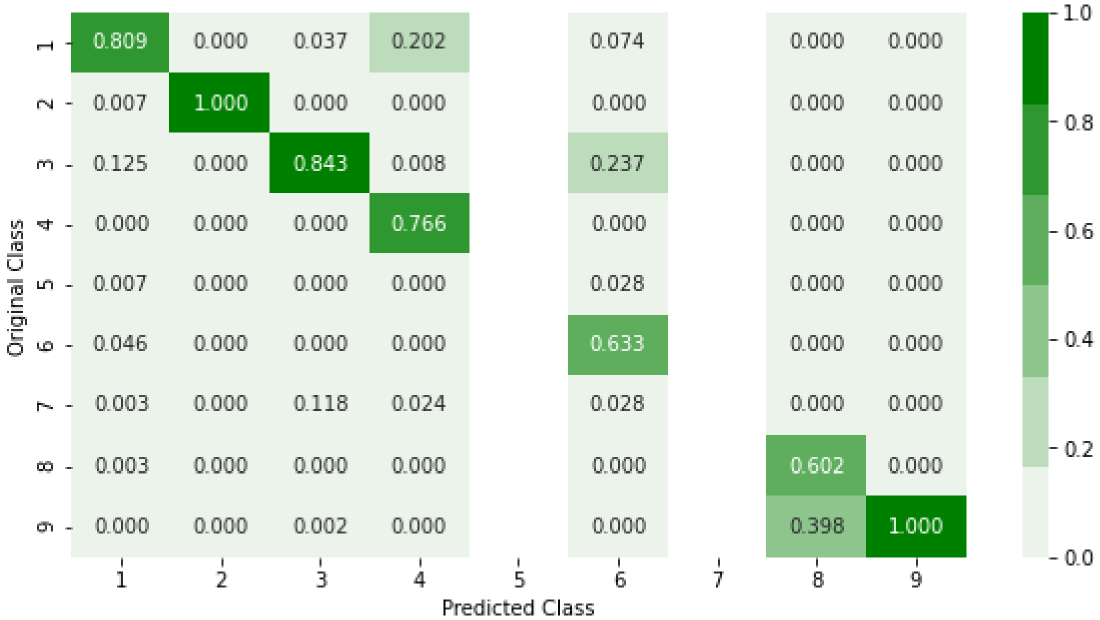


```
log loss for train data 0.582233780860647
log loss for cv data 0.5757195173715258
log loss for test data 0.5786266687749578
Number of misclassified points: 19.36522539098436
------------------------------------------------- Confusion Matrix ----------------
---------------------------------
```



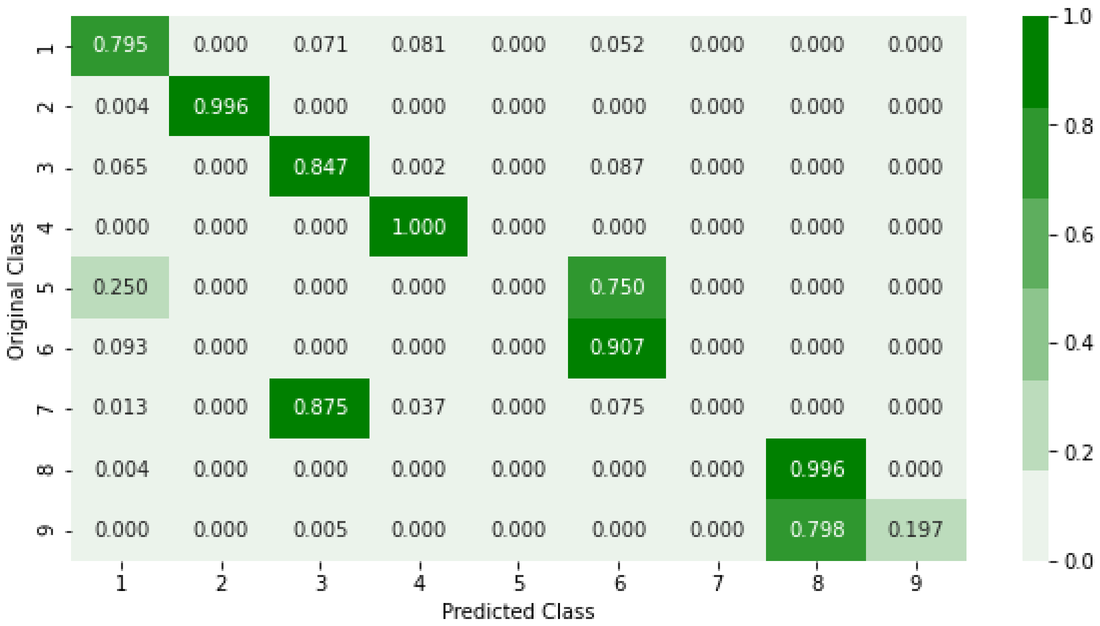
	1	2	3	4	5	6	7	8	9
1	245.000	0.000	22.000	25.000	0.000	16.000	0.000	0.000	0.000
2	2.000	494.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	38.000	0.000	498.000	1.000	0.000	51.000	0.000	0.000	0.000
4	0.000	0.000	0.000	95.000	0.000	0.000	0.000	0.000	0.000
5	2.000	0.000	0.000	0.000	0.000	6.000	0.000	0.000	0.000
6	14.000	0.000	0.000	0.000	0.000	136.000	0.000	0.000	0.000
7	1.000	0.000	70.000	3.000	0.000	6.000	0.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	0.000	245.000	0.000
9	0.000	0.000	1.000	0.000	0.000	0.000	0.000	162.000	40.000

```
--------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



	1	2	3	4	5	6	7	8	9
1	0.809	0.000	0.037	0.202		0.074		0.000	0.000
2	0.007	1.000	0.000	0.000		0.000		0.000	0.000
3	0.125	0.000	0.843	0.008		0.237		0.000	0.000
4	0.000	0.000	0.000	0.766		0.000		0.000	0.000
5	0.007	0.000	0.000	0.000		0.028		0.000	0.000
6	0.046	0.000	0.000	0.000		0.633		0.000	0.000
7	0.003	0.000	0.118	0.024		0.028		0.000	0.000
8	0.003	0.000	0.000	0.000		0.000		0.602	0.000
9	0.000	0.000	0.002	0.000		0.000		0.398	1.000

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



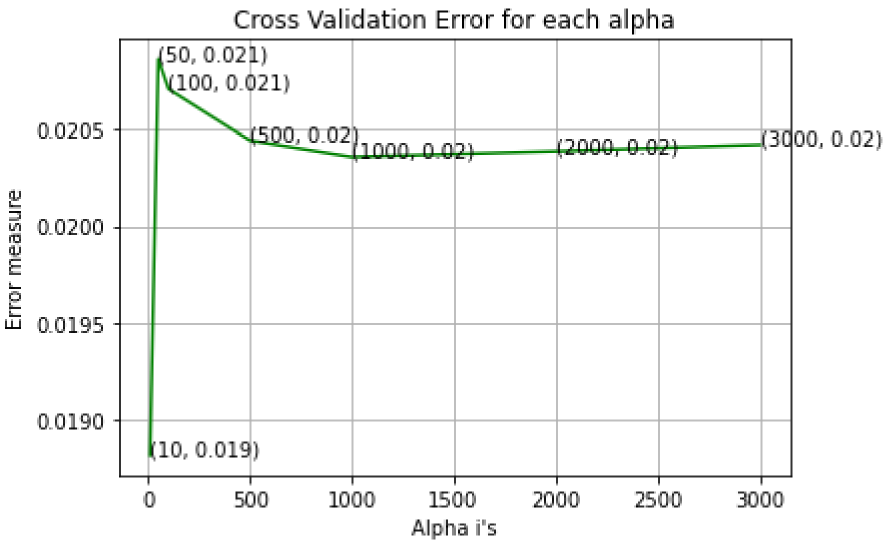
```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Random Forest Classifier

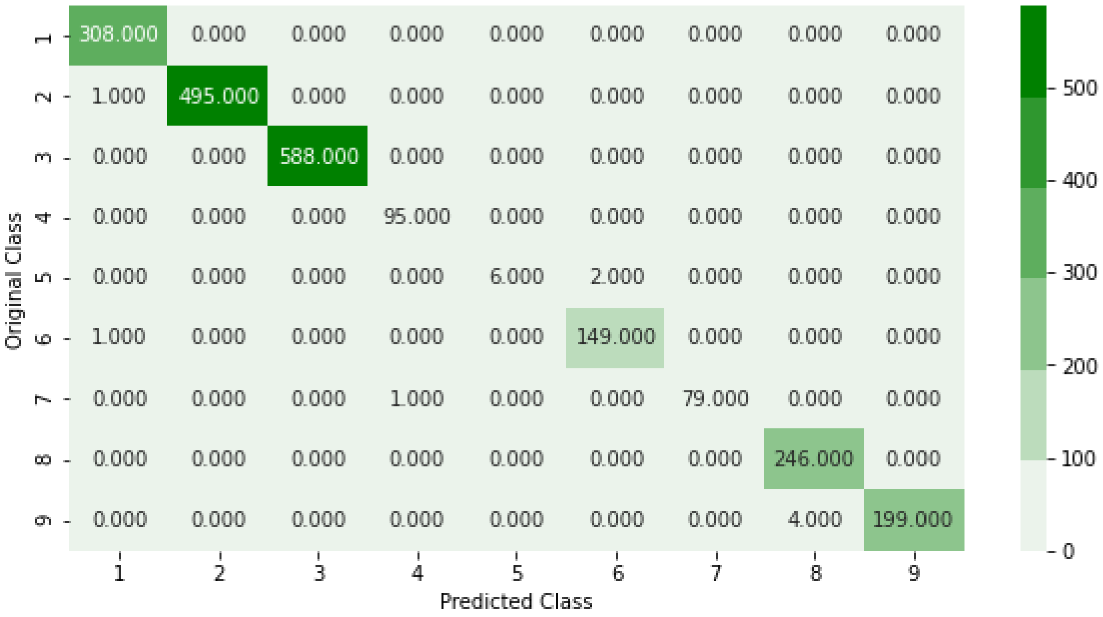
In [ ]:

```
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_, e

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=-
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.cl
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_
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.clas
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
100%|██████████| 7/7 [03:53<00:00, 33.34s/it]
log_loss for c =  10 is 0.018812624307481218
log_loss for c =  50 is 0.020864240419825455
log_loss for c =  100 is 0.020709636899567105
log_loss for c =  500 is 0.020442694972239148
log_loss for c =  1000 is 0.020357716945228893
log_loss for c =  2000 is 0.020387389510357088
log_loss for c =  3000 is 0.020420364814472517
```

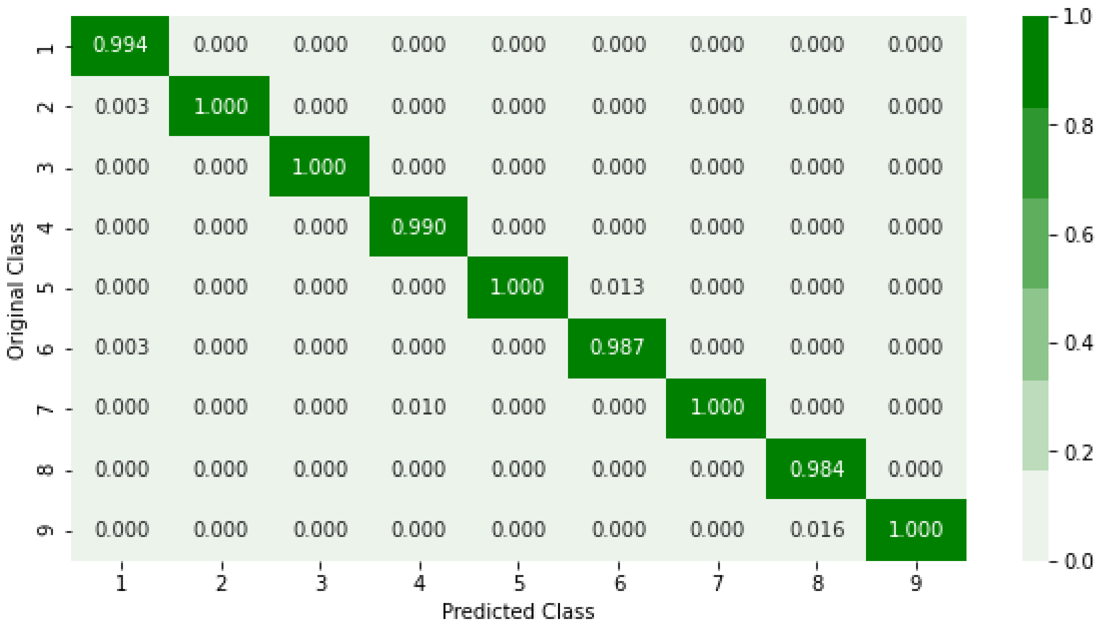


```
log loss for train data 0.009574816488648094
log loss for cv data 0.018812624307481218
log loss for test data 0.022103676831235815
Number of misclassified points: 0.41398344066237347
-------------------------------------------------- Confusion Matrix ----------------
---------------------------------
```

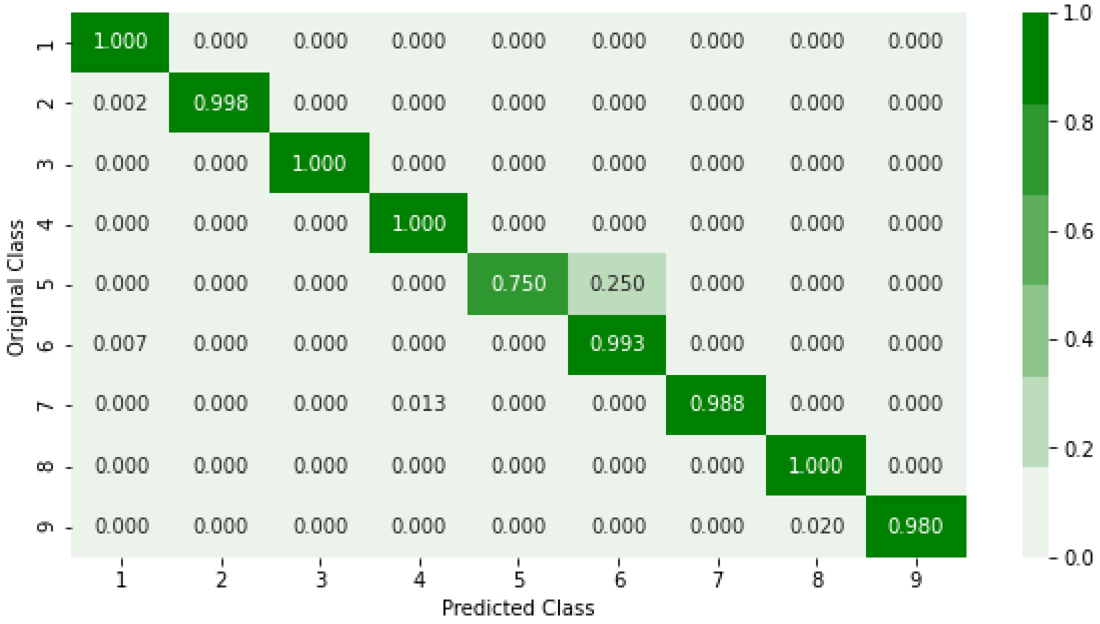


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	308.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	1.000	495.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	95.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	6.000	2.000	0.000	0.000	0.000
6	1.000	0.000	0.000	0.000	0.000	149.000	0.000	0.000	0.000
7	0.000	0.000	0.000	1.000	0.000	0.000	79.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	246.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.000	199.000

```
-------------------------------------------------- Precision Matrix ----------------
---------------------------------
```



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



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

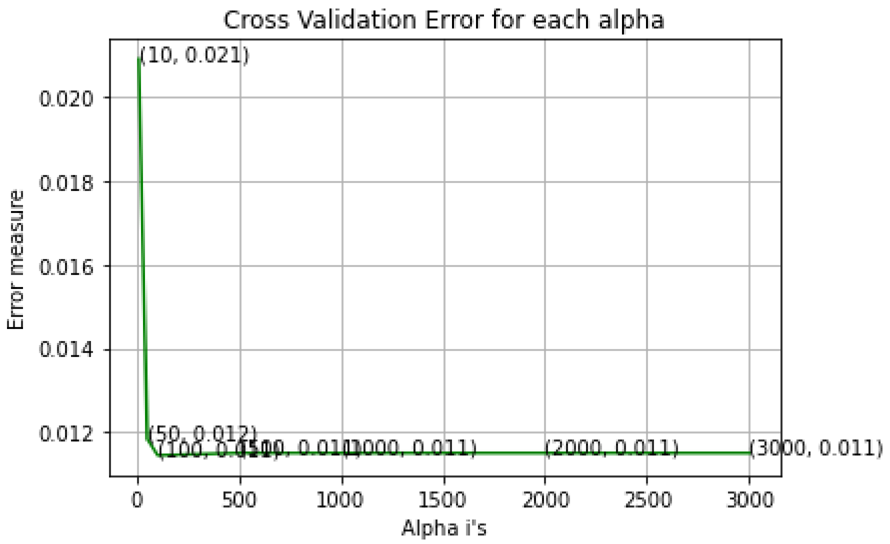
XGBoost Classifier

In [ ]:

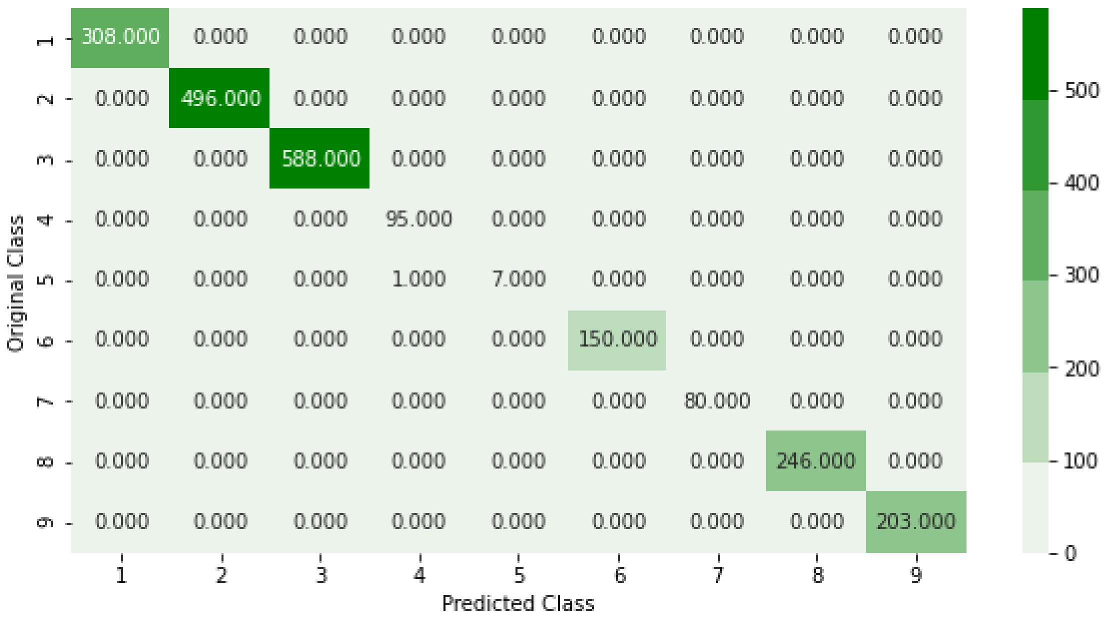
```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)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

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
```

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

```
100%|██████████| 7/7 [1:28:31<00:00, 758.83s/it]
log_loss for c =  10 is 0.02091811130153723
log_loss for c =  50 is 0.011828122279235423
log_loss for c =  100 is 0.01143920152456805
log_loss for c =  500 is 0.011495897765358895
log_loss for c =  1000 is 0.011495895216076047
log_loss for c =  2000 is 0.011495850744131052
log_loss for c =  3000 is 0.011495702802326369
```

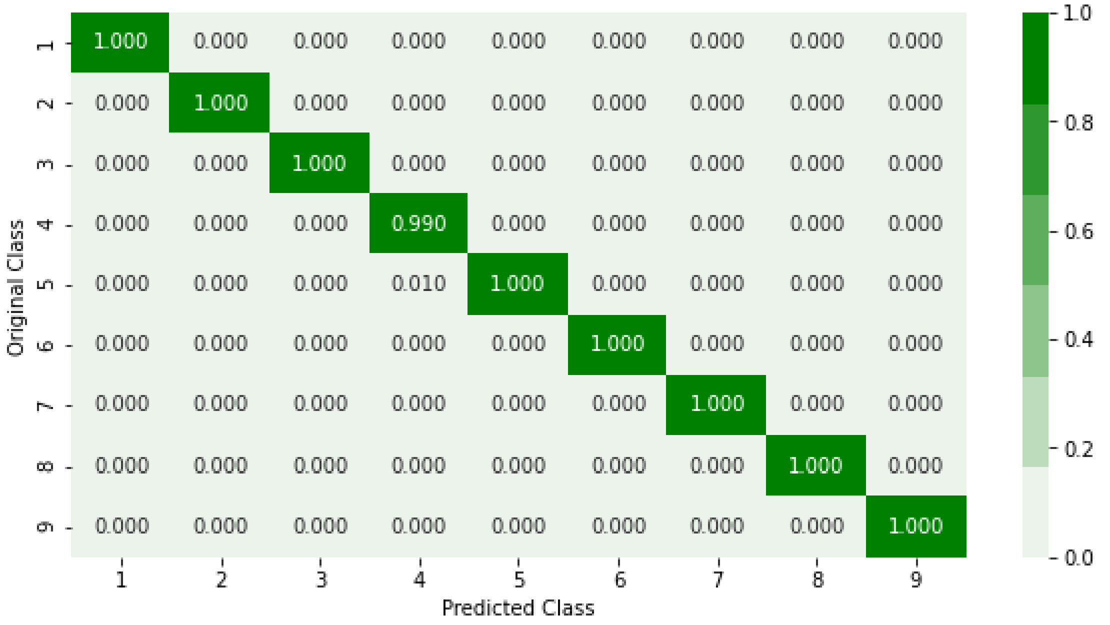


```
For values of best alpha =  100 The train log loss is: 0.007476422524261056
For values of best alpha =  100 The cross validation log loss is: 0.0114392015245680
5
For values of best alpha =  100 The test log loss is: 0.00931880890298055
Number of misclassified points: 0.045998160073597055
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```



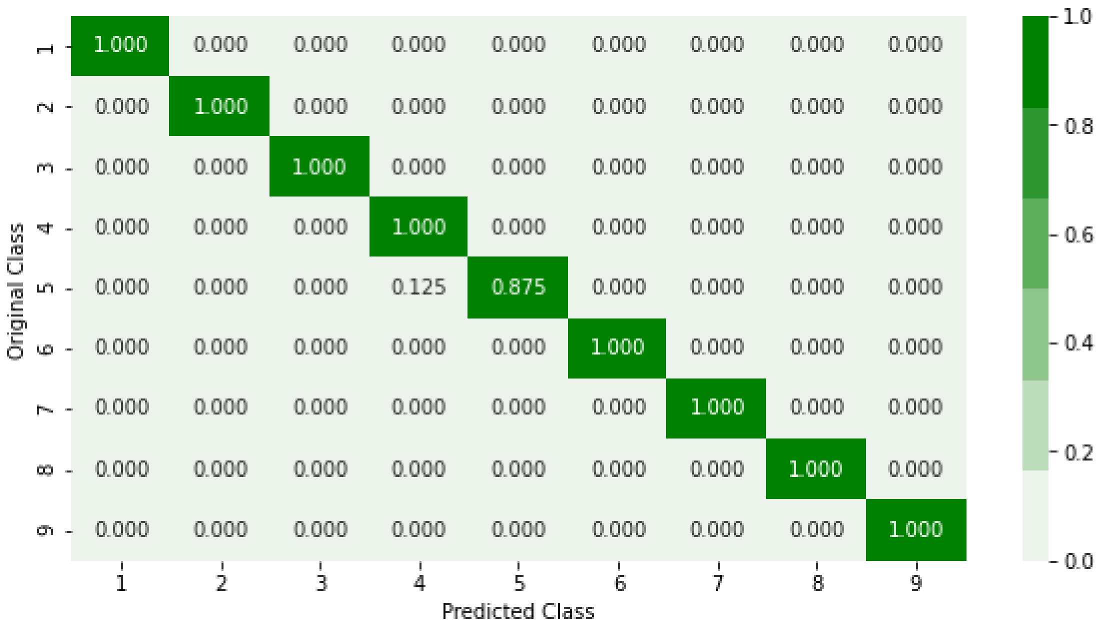
	1	2	3	4	5	6	7	8	9
1	308.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	496.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	95.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	1.000	7.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	150.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	80.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	246.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	203.000

```
-------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



	1	2	3	4	5	6	7	8	9
1	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.990	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.010	1.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

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



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

In [ ]:

```
pickle.dump(x_cfl, open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware
pickle.dump(sig_clf, open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malwar
```

Combining byte and asm features

Importing Data

In [ ]:

```
bigrams = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware De
byte_features = bigrams.merge(result, on = "ID")
```

In [ ]:

```
byte_features_no_id = byte_features.drop(["ID", "Class"], axis = 1)
byte_features_no_id_norm = normalize(byte_features_no_id)
bigram_column_names = byte_features_no_id.columns
byte_features_norm = pd.DataFrame(data=byte_features_no_id_norm, columns = bigram_co
byte_features_ids = byte_features["ID"]
byte_features_norm.insert(loc = 0, column = "ID", value = byte_features_ids)
byte_features_norm.drop("Unnamed: 0_x", axis = 1, inplace =True)
byte_features_norm.head()
```

Out[ ]:

	ID	00_00	00_01	00_02	00_03	00_04	00_05	00_06	
0	01IsoiSMh5gxyDYTl4CB	0.448546	0.016245	0.001446	0.000972	0.003593	0.000226	0.000136	0.0
1	01SuzwMJEIXsK7A8dQbl	0.865108	0.003282	0.001132	0.006225	0.000453	0.000622	0.000170	0.0
2	01azqd4InC7m9JpocGv5	0.996383	0.003656	0.002923	0.004269	0.003441	0.003065	0.004105	0.0
3	01jsnpXSAlgw6aPeDxrU	0.587520	0.021695	0.005754	0.005277	0.018653	0.021622	0.020192	0.0
4	01kcPWA9K2BOxQeS5Rju	0.597321	0.012305	0.003559	0.004162	0.006213	0.002051	0.001146	0.0

5 rows × 2760 columns

In [ ]:

```
byte_features_norm = byte_features_norm.rename(columns={"Size":"Byte_Size"})
```

In [ ]:

```
asm_features = asm_features.rename(columns={"Size":"asm_Size"})
```

In [ ]:

```
final_df = byte_features_norm.merge(asm_features, on = "ID")
```

In [ ]:

```
final_df.head()
```

Out[ ]:

	ID	00_00	00_01	00_02	00_03	00_04	00_05	00_06	(
0	01IsoiSMh5gxyDYTl4CB	0.448546	0.016245	0.001446	0.000972	0.003593	0.000226	0.000136	0.0(
1	01SuzwMJEIXsK7A8dQbl	0.865108	0.003282	0.001132	0.006225	0.000453	0.000622	0.000170	0.0(
2	01azqd4InC7m9JpocGv5	0.996383	0.003656	0.002923	0.004269	0.003441	0.003065	0.004105	0.0(
3	01jsnpXSAlgw6aPeDxrU	0.587520	0.021695	0.005754	0.005277	0.018653	0.021622	0.020192	0.0(
4	01kcPWA9K2BOxQeS5Rju	0.597321	0.012305	0.003559	0.004162	0.006213	0.002051	0.001146	0.0(

5 rows × 3615 columns

In [ ]:

```
final_df.to_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware Detectio
```

In [4]:

```
final_df = pd.read_csv("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware D
```

Train Test Split

In [5]:

```
final_y = final_df['Class']
final_x = final_df.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

Modelling on Byte and asm Features

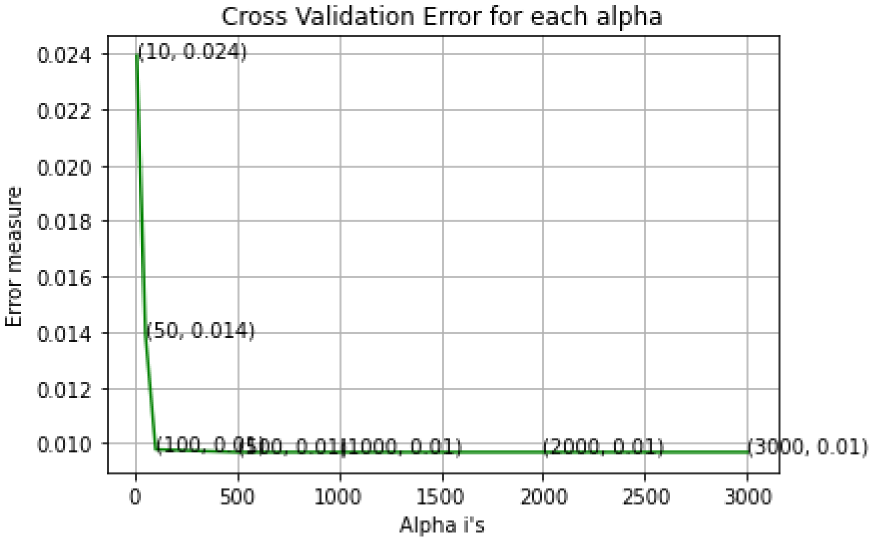
In [6]:

```
X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(final_x,
X_train_final, X_cv_final, y_train_final, y_cv_final = train_test_split(X_train_fina
```

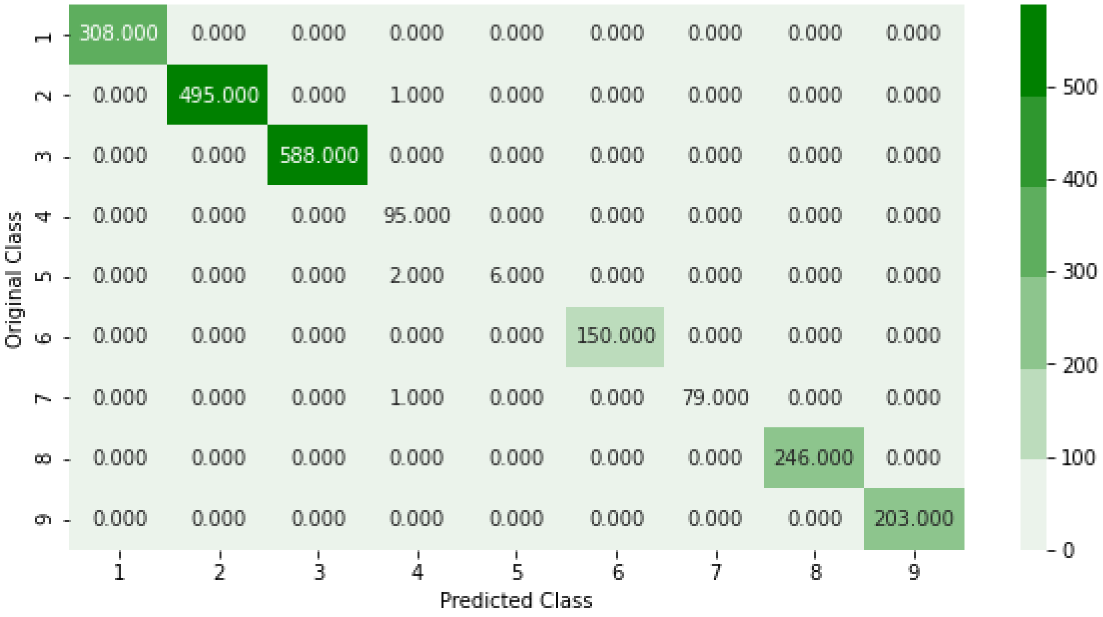
In [7]:

```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_final,y_train_final)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_final, y_train_final)
    predict_y = sig_clf.predict_proba(X_cv_final)
    temp_log_loss = log_loss(y_cv_final, predict_y, labels=x_cfl.classes_, eps=1e-15
    cv_log_error_array.append(temp_log_loss)
    print(f"\nLog Loss for {i} trees is {temp_log_loss}")


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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_final)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",lo
predict_y = sig_clf.predict_proba(X_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
predict_y = sig_clf.predict_proba(X_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
plot_confusion_matrix(y_test_final,sig_clf.predict(X_test_final))
```

```
 14%|█▍        | 1/7 [03:31<21:06, 211.07s/it]
Log Loss for 10 trees is 0.02392770130400327
 29%|██▊       | 2/7 [19:52<55:21, 664.27s/it]
Log Loss for 50 trees is 0.013894133530606546
 43%|████▎     | 3/7 [45:58<1:11:44, 1076.00s/it]
Log Loss for 100 trees is 0.00977380646700361
 57%|█████▋    | 4/7 [1:41:12<1:37:58, 1959.49s/it]
Log Loss for 500 trees is 0.00967474963599015
 71%|███████▏  | 5/7 [3:13:16<1:48:09, 3244.85s/it]
Log Loss for 1000 trees is 0.009674823624372478
 86%|████████▌ | 6/7 [5:57:24<1:31:30, 5490.13s/it]
Log Loss for 2000 trees is 0.009674625159195115
100%|██████████| 7/7 [9:49:31<00:00, 5053.13s/it]
Log Loss for 3000 trees is 0.009674656085950483
log_loss for c =  10 is 0.02392770130400327
log_loss for c =  50 is 0.013894133530606546
log_loss for c =  100 is 0.00977380646700361
log_loss for c =  500 is 0.00967474963599015
log_loss for c =  1000 is 0.009674823624372478
log_loss for c =  2000 is 0.009674625159195115
log_loss for c =  3000 is 0.009674656085950483
```

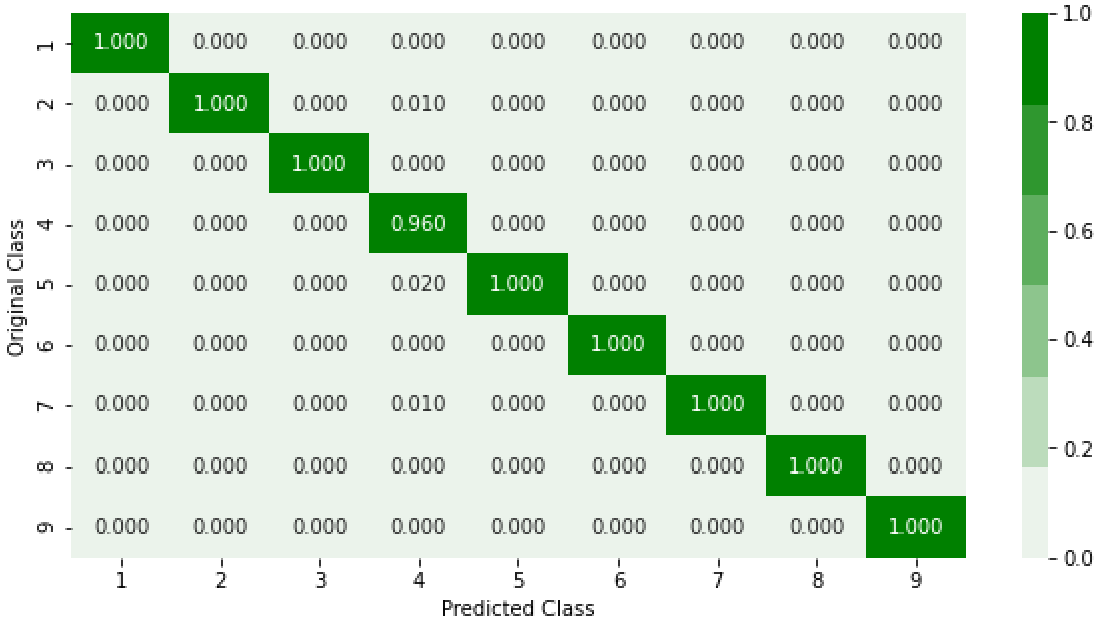


```
For values of best alpha =  2000 The train log loss is: 0.008128682789492898
For values of best alpha =  2000 The cross validation log loss is: 0.009674625159195
115
For values of best alpha =  2000 The test log loss is: 0.014297763213690582
Number of misclassified points: 0.18399264029438822
-------------------------------------------------- Confusion Matrix ----------------
----------------------------------
```

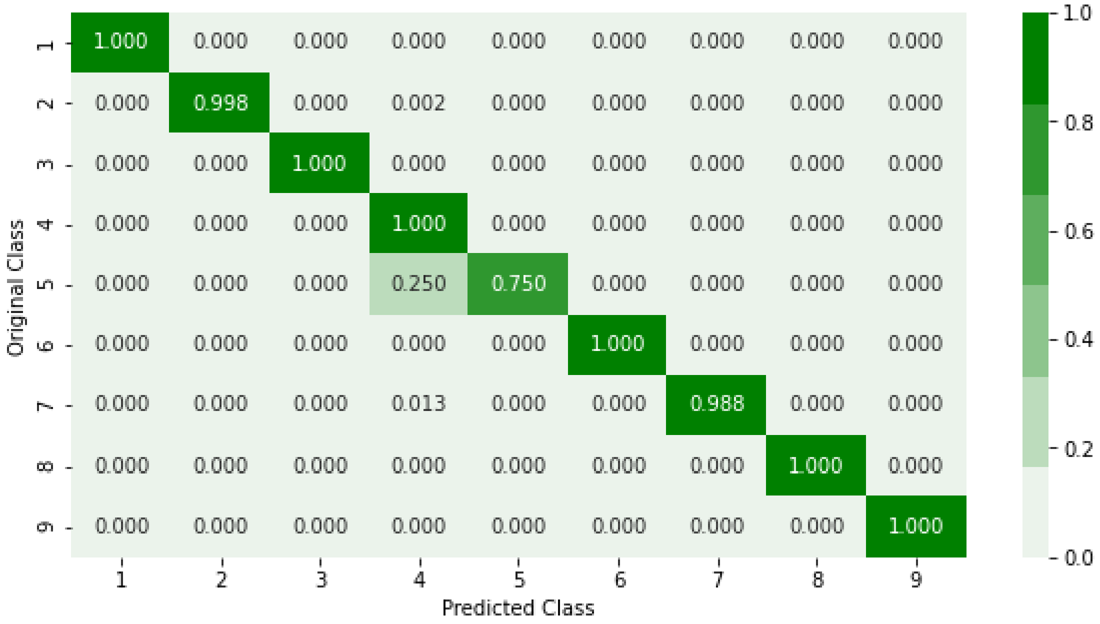


	1	2	3	4	5	6	7	8	9
1	308.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	495.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	95.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	2.000	6.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	150.000	0.000	0.000	0.000
7	0.000	0.000	0.000	1.000	0.000	0.000	79.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	246.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	203.000

```
-------------------------------------------------- Precision Matrix ----------------
----------------------------------
```



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



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

In [8]:

```
pickle.dump(x_cfl, open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malware 
pickle.dump(sig_clf, open("/content/drive/MyDrive/AAIC/Case Studies/Microsoft Malwar
```

Results Table

In [30]:

```
train_results = [2.452, 0.072, 0.498, 0.026, 0.022, 2.492, 0.098, 0.884, 0.018, 0.04
cv_results =    [2.483, 0.225, 0.549, 0.087, 0.093, 2.439, 0.289, 0.866, 0.034, 0.09
test_results =  [2.485, 0.241, 0.528, 0.085, 0.079, 2.872, 0.327, 0.887, 0.029, 0.08
dataset_names = ['Unigrams + Byte Size', 'Unigrams + Byte Size', 'Unigrams + Byte Si
model_names = ['Random Model', 'KNN', 'Logistic Regression', 'Random Forest', 'XGBoo
results_data = {'Dataset': dataset_names, 'Model': model_names, 'Train Log Loss': tr
results_df = pd.DataFrame(results_data)
```

In [31]:

```
display(results_df)
```

	Dataset	Model	Train Log Loss	CV Log Loss	Test Log Loss
0	Unigrams + Byte Size	Random Model	2.452	2.483	2.485
1	Unigrams + Byte Size	KNN	0.072	0.225	0.241
2	Unigrams + Byte Size	Logistic Regression	0.498	0.549	0.528
3	Unigrams + Byte Size	Random Forest	0.026	0.087	0.085
4	Unigrams + Byte Size	XGBoost	0.022	0.093	0.079
5	Unigrams + Bigrams + Byte Size	Random Model	2.492	2.439	2.872
6	Unigrams + Bigrams + Byte Size	KNN	0.098	0.289	0.327
7	Unigrams + Bigrams + Byte Size	Logistic Regression	0.884	0.866	0.887
8	Unigrams + Bigrams + Byte Size	XGBoost	0.018	0.034	0.029
9	Op Code + asm Size	KNN	0.047	0.095	0.089
10	Op Code + asm Size	Logistic Regression	0.396	0.424	0.415
11	Op Code + asm Size	Random Forest	0.011	0.049	0.057
12	Op Code + asm Size	XGBoost	0.011	0.056	0.046
13	Op Code + asm Size + Pixel Density	KNN	0.021	0.071	0.062
14	Op Code + asm Size + Pixel Density	Logistic Regression	0.582	0.575	0.578
15	Op Code + asm Size + Pixel Density	Random Forest	0.009	0.188	0.022
16	Op Code + asm Size + Pixel Density	XGBoost	0.007	0.011	0.009
17	All Byte + asm Features	XGBoost	0.008	0.009	0.014