Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- **Source : https://www.kaggle.com/c/malware-classification/data**
- **For every malware, we have two files**

1. .asm file (read more: https://www.reviversoft.com/file-extensions/asm)
2. .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)

- Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:
- Lots of Data for a single-box/computer.
- There are total 10,868 .bytes files and 10,868 asm files total 21,736 files
- There are 9 types of malwares (9 classes) in our give data
- Types of Malware:

1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

2.1.2. Example Data Point

.asm file

```
.text:00401000                                         assume
es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.text:00401000 56                                      push    esi
.text:00401001 8D 44 24 08                                 lea
eax, [esp+8]
.text:00401005 50                                      push    eax
.text:00401006 8B F1                                       mov
esi, ecx
.text:00401008 E8 1C 1B 00 00                                  call
??0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char
const * const &)
.text:0040100D C7 06 08 BB 42 00                               mov
dword ptr [esi], offset off_42BB08
.text:00401013 8B C6                                       mov
eax, esi
.text:00401015 5E                                      pop     esi
.text:00401016 C2 04 00                                    retn    4
.text:00401016                                 ; ------------------
------------------------------------------------------
.text:00401019 CC CC CC CC CC CC CC                            align
10h
.text:00401020 C7 01 08 BB 42 00                               mov
dword ptr [ecx], offset off_42BB08
.text:00401026 E9 26 1C 00 00                                  jmp
sub_402C51
.text:00401026                                 ; ------------------
------------------------------------------------------
.text:0040102B CC CC CC CC CC                                  align
```

```
10h
.text:00401030 56                                      push    esi
.text:00401031 8B F1                                       mov
esi, ecx
.text:00401033 C7 06 08 BB 42 00                           mov
dword ptr [esi], offset off_42BB08
.text:00401039 E8 13 1C 00 00                              call
sub_402C51
.text:0040103E F6 44 24 08 01                              test
byte ptr [esp+8], 1
.text:00401043 74 09                                       jz
short loc_40104E
.text:00401045 56                                      push    esi
.text:00401046 E8 6C 1E 00 00                              call
??3@YAXPAX@Z    ; operator delete(void *)
.text:0040104B 83 C4 04                                    add
esp, 4
.text:0040104E
.text:0040104E                                 loc_40104E:
; CODE XREF: .text:00401043 j
.text:0040104E 8B C6                                       mov
eax, esi
.text:00401050 5E                                          pop     esi
.text:00401051 C2 04 00                                    retn    4
.text:00401051                                 ; ------------------
--------------------------------------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 20 00 02 00 04
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
```

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

2.2.1. Type of Machine Learning Problem

There are nine different classes of malware that we need to classify a given a data point => Multi class classification problem

2.2.2. Performance Metric

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

Metric(s):

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

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

In [2]:
```
import dill
dill.load_session("/data/malware-classification/notebook.db")
```

3. Exploratory Data Analysis

In [23]:
```
import warnings
warnings.filterwarnings("ignore")
```

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

import scipy
from sklearn.feature_extraction.text import CountVectorizer

%matplotlib inline
```

In [2]:

```
#separating byte and asm files from train folder to separate byteFiles and as

source = 'train'
destination_1 = 'byteFiles'
destination_2 = 'asmFiles'

# we will check if the folder 'byteFiles' exists if it not there we will crea
# folder with the same name
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

# so by the end of this snippet we will separate all the .byte files and .asm
if os.path.isdir(source):
    data_files = os.listdir(source)
    for file in data_files:
        print(file)
        if (file.endswith("bytes")):
            shutil.move(source+'\\'+file,destination_1)
        if (file.endswith("asm")):
            shutil.move(source+'\\'+file,destination_2)
```

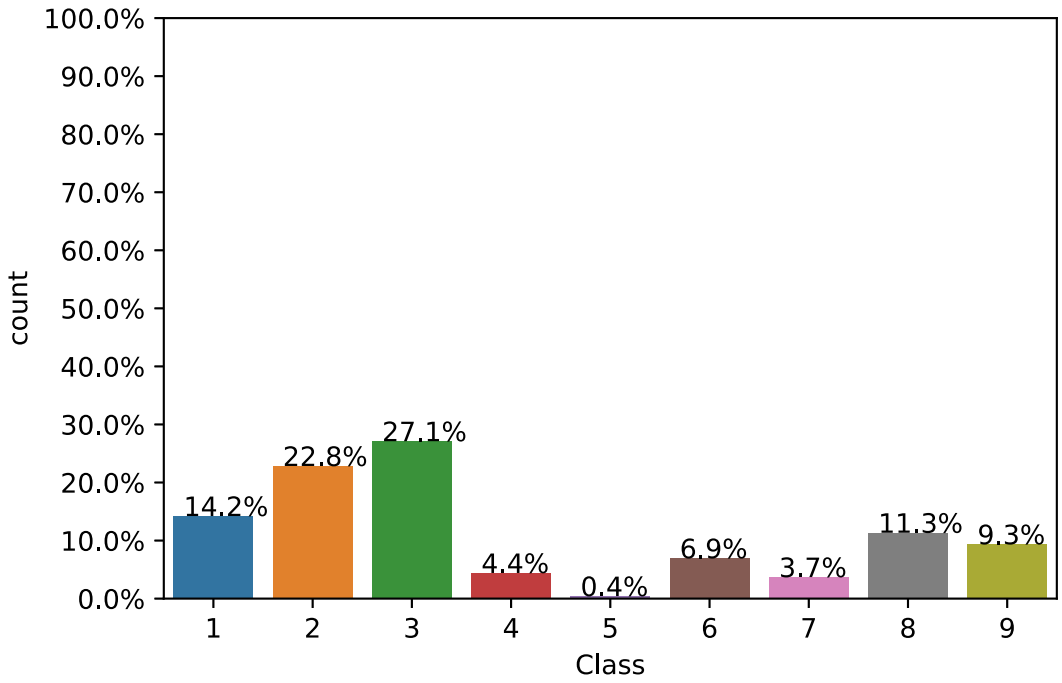
```
train
```

3.1. Distribution of malware classes in whole data set

In [2]:

```
Y=pd.read_csv("/data/malware-classification/trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
```

```
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [3]:

```
#file sizes of byte files
files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=356157170
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file n
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes
print (data_size_byte.head())
```

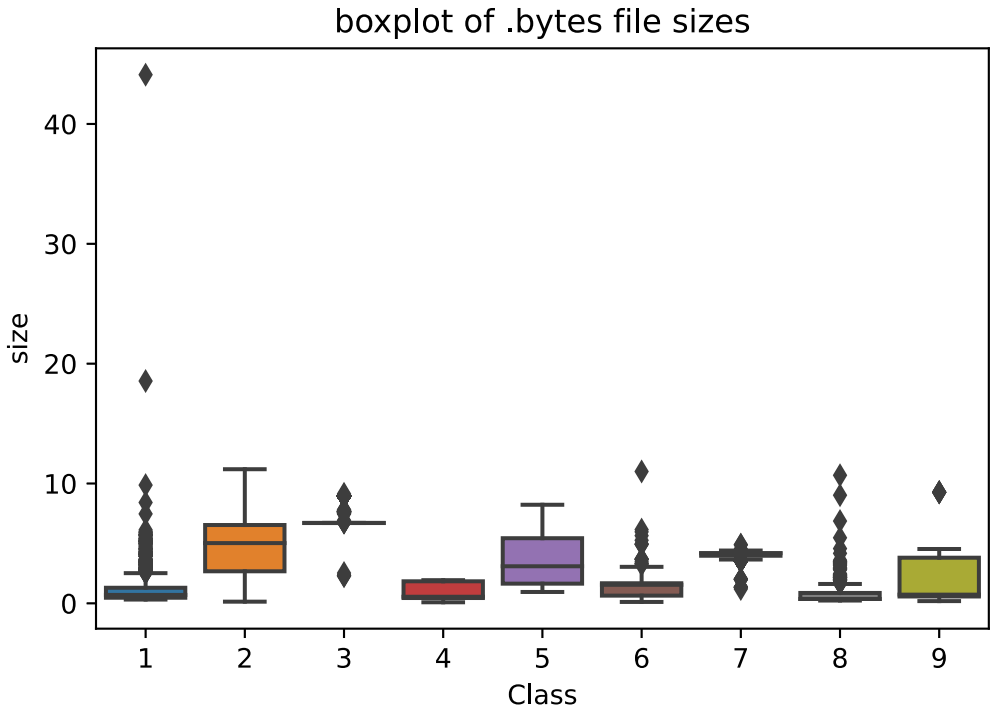
```
                      ID      size  Class
```

```
0  07iSOIG2urUvsMl9E5Rn  6.656250  2
1  07nrG1cLKUPxjOlWMFiV  8.941406  3
2  08BX5Slp2I1FraZWbc6j  0.867188  6
3  09bfacpUzuBN5W3S8KTo  6.703125  3
4  09CPNMYyQjSguFrE8UOf  1.664062  6
```

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

In [4]:

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



3.2.3 feature extraction from byte files

In [6]:

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

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

```
        text_file.close()

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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13
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 [4]:

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

Out[4]:

	ID	0	1	2	3	4	5	6	7	8
0	07iSOIG2urUvsMl9E5Rn	86939	62767	62238	68497	12864	6715	8506	7535	9278
1	07nrG1cLKUPxjOlWMFiV	13697	8357	6166	6010	5935	6211	6228	6312	6261

2 rows × 258 columns

In [5]:

```
data_size_byte.columns.values
```

Out[5]:

```
array(['ID', 'size', 'Class'], dtype=object)
```

In [6]:

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

Out[6]:

	ID	0	1	2	3	4	5	6	7	8
0	07iSOIG2urUvsMl9E5Rn	86939	62767	62238	68497	12864	6715	8506	7535	9278

	ID	0	1	2	3	4	5	6	7	8
1	07nrG1cLKUPxjOlWMFiV	13697	8357	6166	6010	5935	6211	6228	6312	6261

2 rows × 260 columns

In [7]:
```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class'
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_val
    return result1
result = normalize(byte_features_with_size)
```

In [8]:
```
result.head(2)
```

Out[8]:

	ID	0	1	2	3	4	5
0	07iSOIG2urUvsMl9E5Rn	0.03796	0.088366	0.034627	0.036951	0.007875	0.003800
1	07nrG1cLKUPxjOlWMFiV	0.00598	0.011765	0.003431	0.003242	0.003633	0.003515

2 rows × 260 columns

In [9]:
```
data_y = result['Class']
result.head()
```

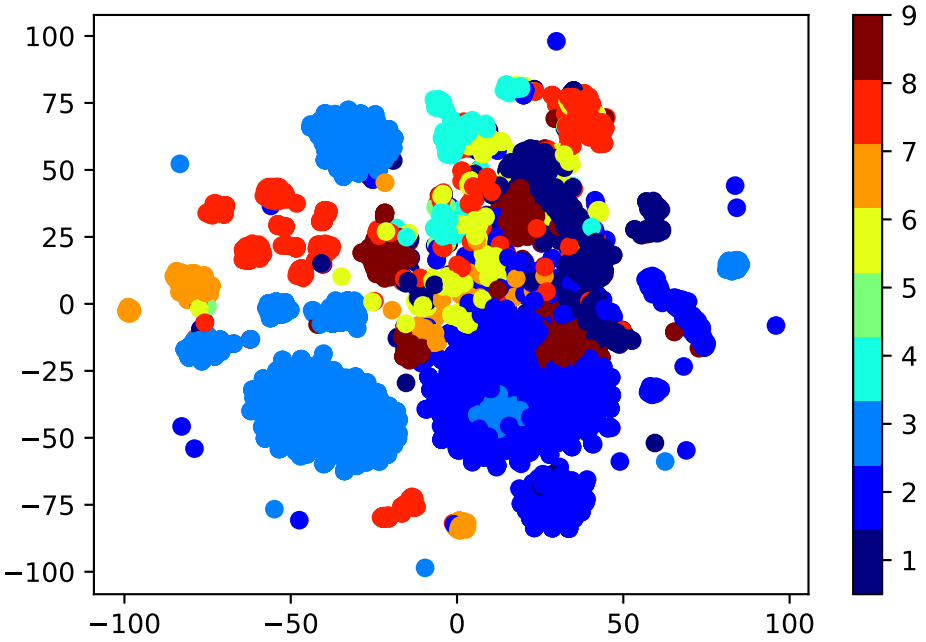
Out[9]:

	ID	0	1	2	3	4	
0	07iSOIG2urUvsMl9E5Rn	0.037960	0.088366	0.034627	0.036951	0.007875	0.00380
1	07nrG1cLKUPxjOlWMFiV	0.005980	0.011765	0.003431	0.003242	0.003633	0.00351
2	08BX5Slp2I1FraZWbc6j	0.081074	0.006632	0.000314	0.000500	0.000624	0.00022
3	09bfacpUzuBN5W3S8KTo	0.005230	0.007822	0.001832	0.001787	0.001985	0.00180
4	09CPNMYyQjSguFrE8UOf	0.029241	0.002876	0.000993	0.000966	0.002361	0.00090

5 rows × 260 columns

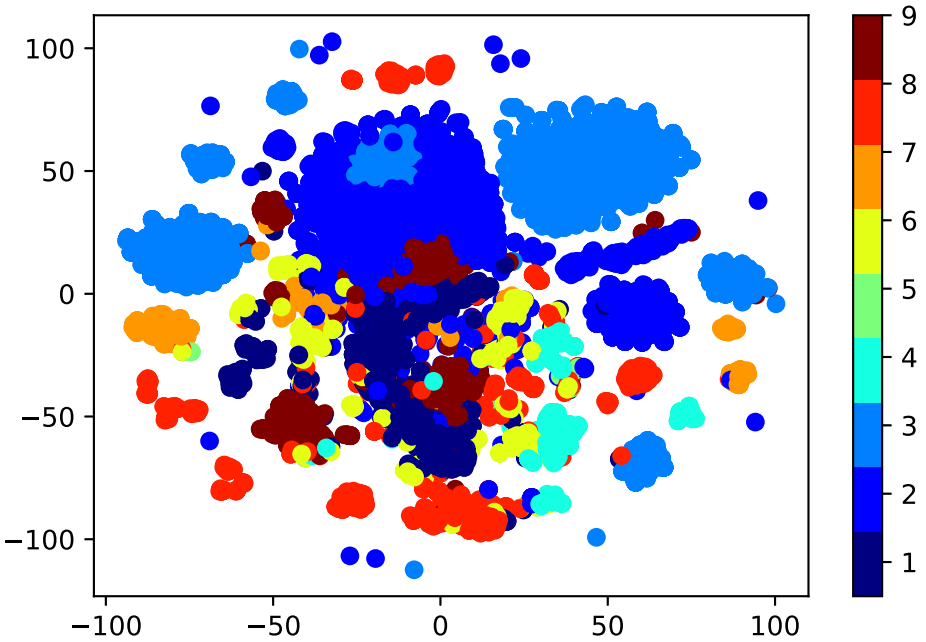
3.2.4 Multivariate Analysis

In [12]:
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



In [13]:

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



Train Test split

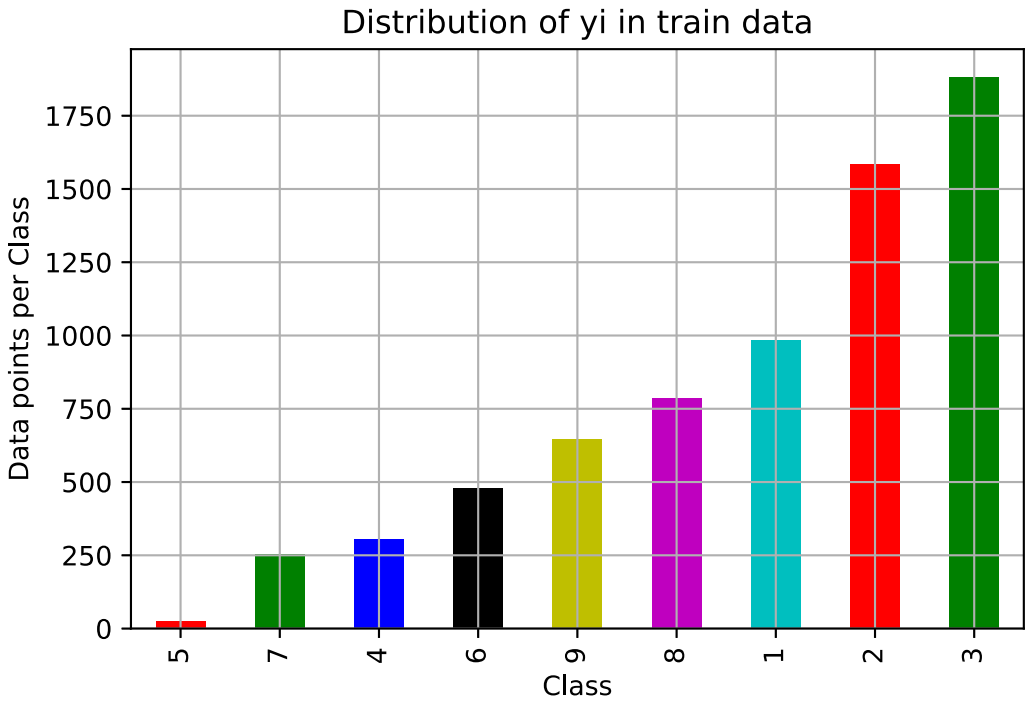
In [10]:

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of outp
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'
# split the train data into train and cross validation by maintaining same di
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_t
```

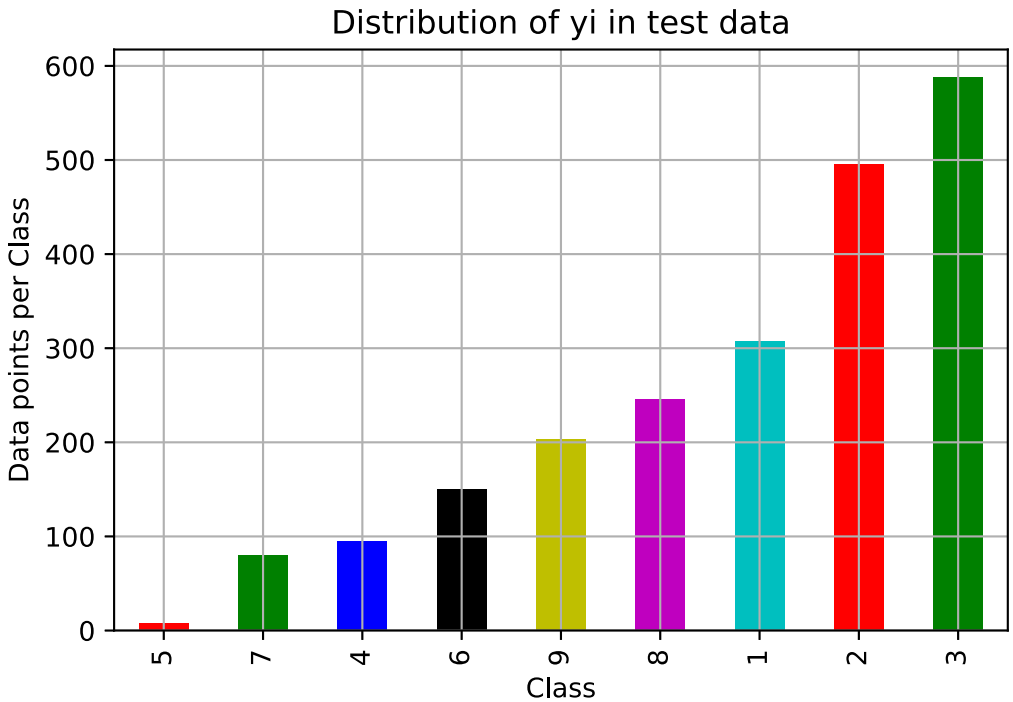
In [11]:
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

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

In [12]:
```
# it returns a dict, keys as class labels and values as the number of data po
train_class_distribution = y_train.value_counts().sort_values()
test_class_distribution = y_test.value_counts().sort_values()
cv_class_distribution = y_cv.value_counts().sort_values()

#my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.arg
# -(train_class_distribution.values): the minus sign will give us in decreasi
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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.arg
# -(train_class_distribution.values): the minus sign will give us in decreasi
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.

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

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

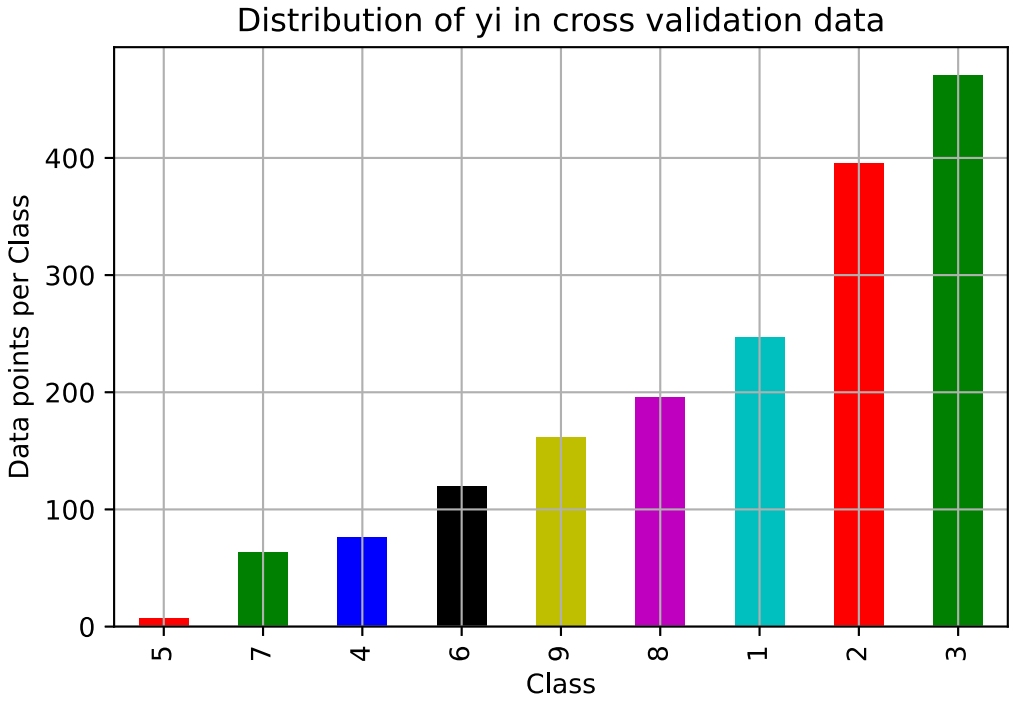



```
Number of data points in class 9 : 1883 ( 27.074 %)
Number of data points in class 8 : 1586 ( 22.804 %)
Number of data points in class 7 : 986 ( 14.177 %)
Number of data points in class 6 : 786 ( 11.301 %)
Number of data points in class 5 : 648 ( 9.317 %)
Number of data points in class 4 : 481 (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




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

```
Number of data points in class 1 : 8 ( 0.368 %)
--------------------------------------------------------------------------------
--
```




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

In [77]:

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

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

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

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

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

```
    #                       [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytic
    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, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of columns in precision matrix",B.sum(axis=0))

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

4. Machine Learning Models

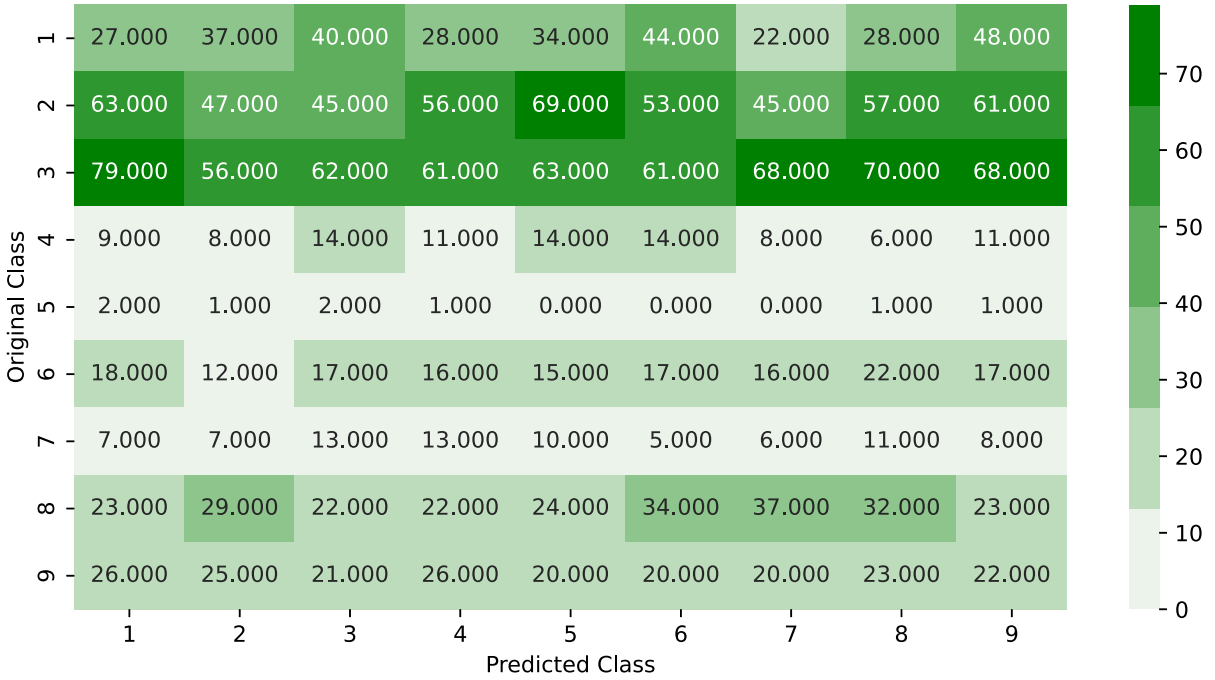
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

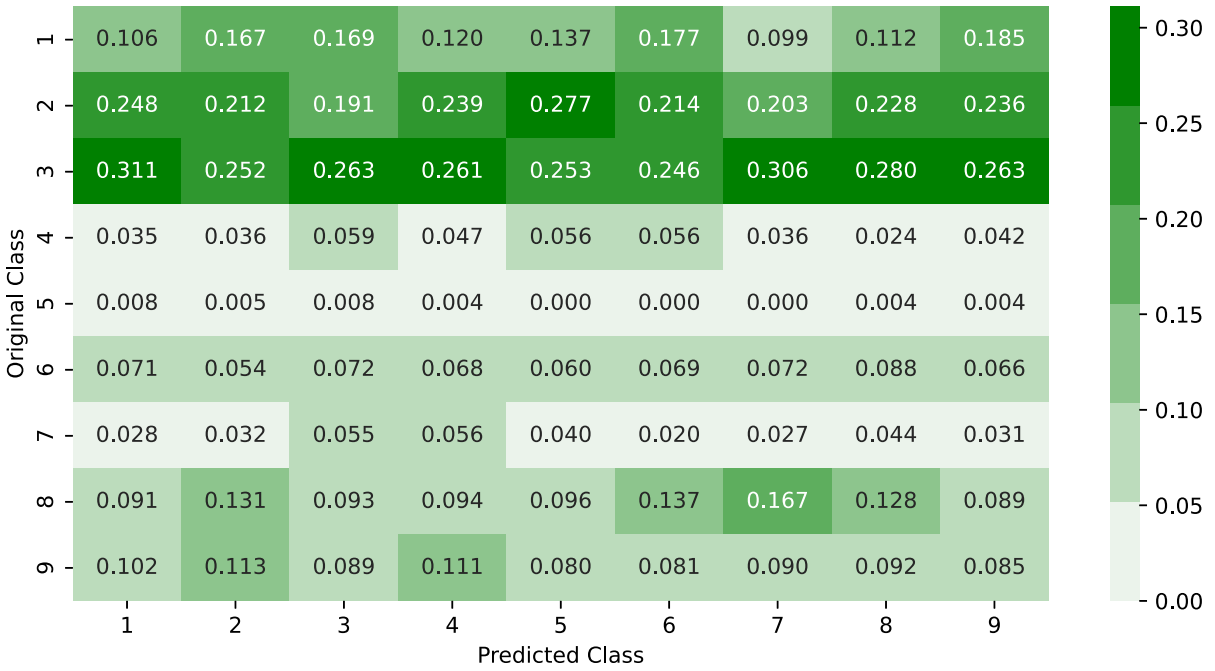
In [16]:

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

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

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

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

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

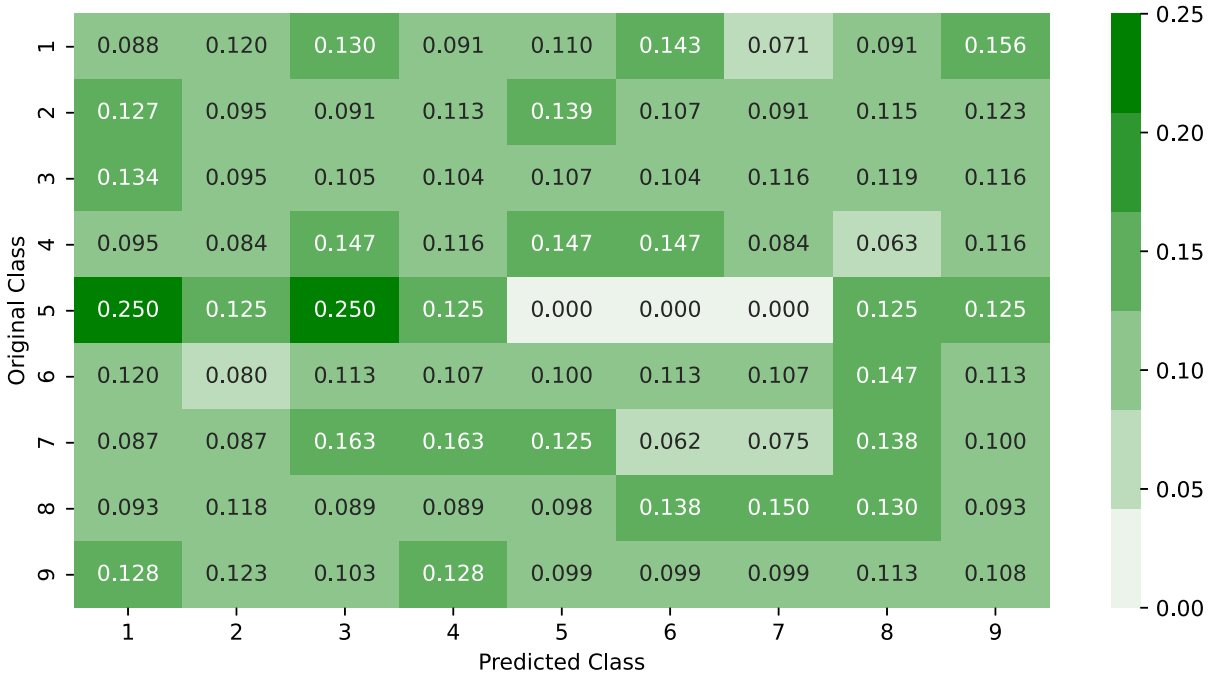
```
Log loss on Cross Validation Data using Random Model 2.4806964033309646
Log loss on Test Data using Random Model 2.4682296382568527
Number of misclassified points  89.69641214351427
---------------------------------------------------- Confusion matrix ----------
------------------------------------------
```




```
---------------------------------------------------- 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.]

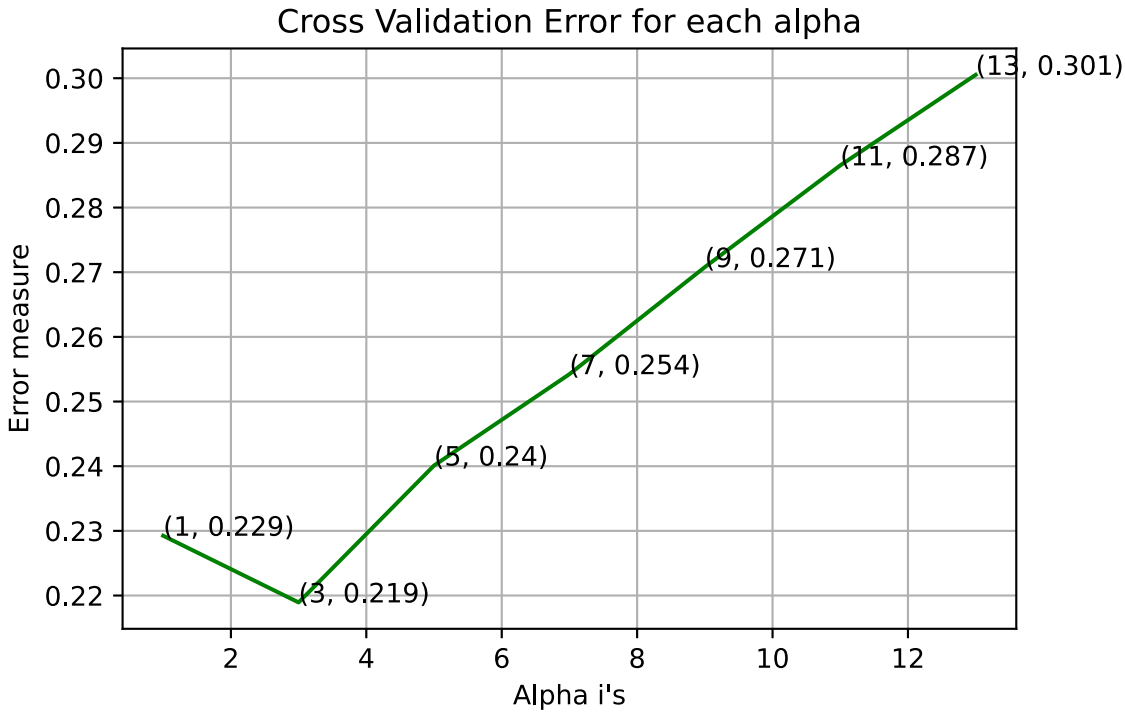
4.1.2. K Nearest Neighbour Classification

In [17]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', le
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

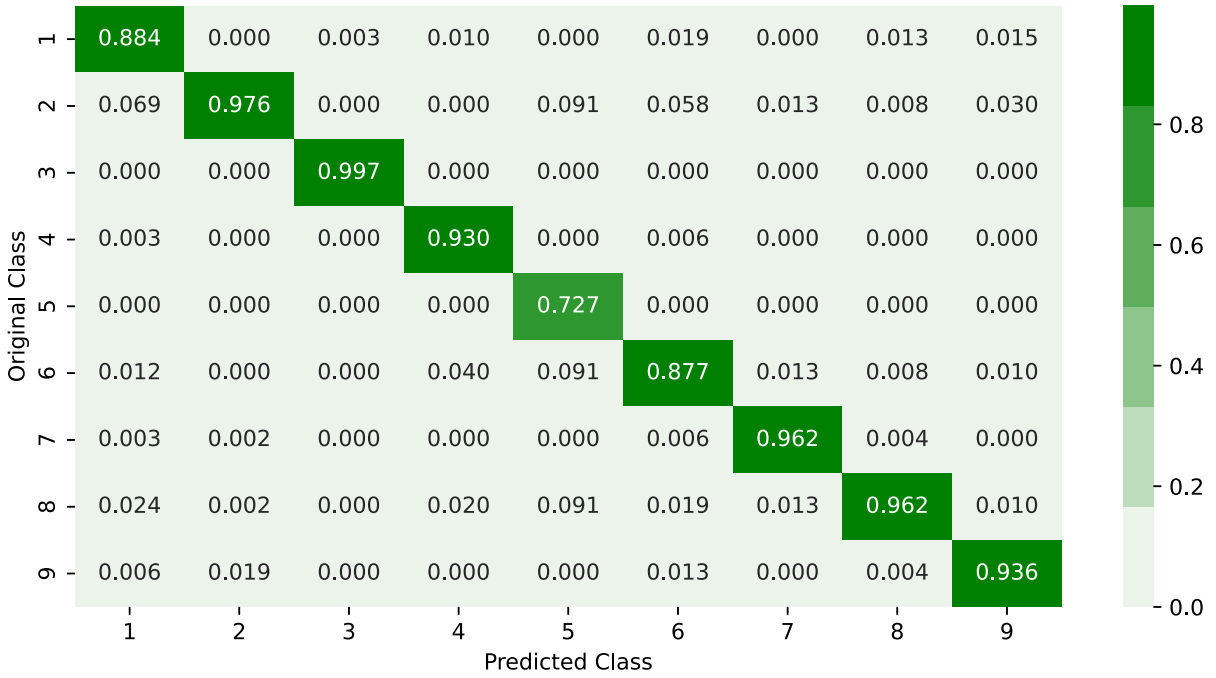
```
log_loss for k =  1 is 0.2292442421199745
log_loss for k =  3 is 0.218931734065492
log_loss for k =  5 is 0.24010052082200675
log_loss for k =  7 is 0.25423094208643443
log_loss for k =  9 is 0.2707853768993781
log_loss for k =  11 is 0.286521756071548
log_loss for k =  13 is 0.3005234028325693
```




```
For values of best alpha =  3 The train log loss is: 0.1092782805150401
For values of best alpha =  3 The cross validation log loss is: 0.218931734065
492
For values of best alpha =  3 The test log loss is: 0.20692538394443238
Number of misclassified points  4.875804967801288
```

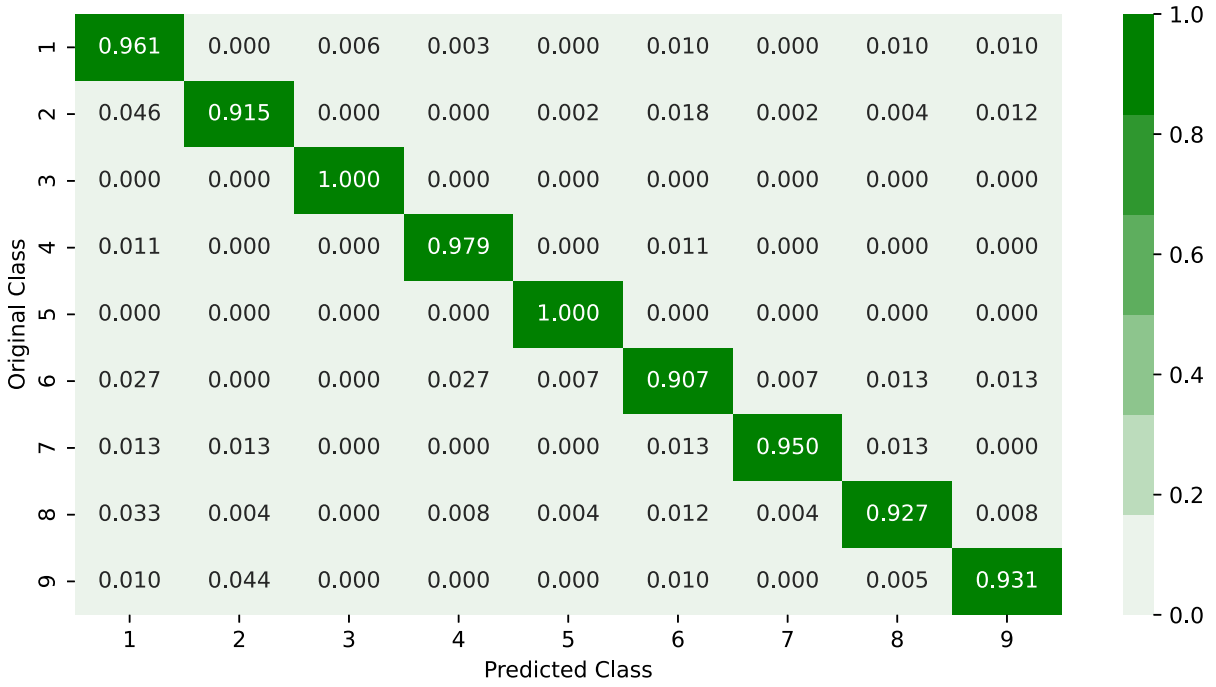
---------- Confusion matrix ----------

---------- 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.]
```

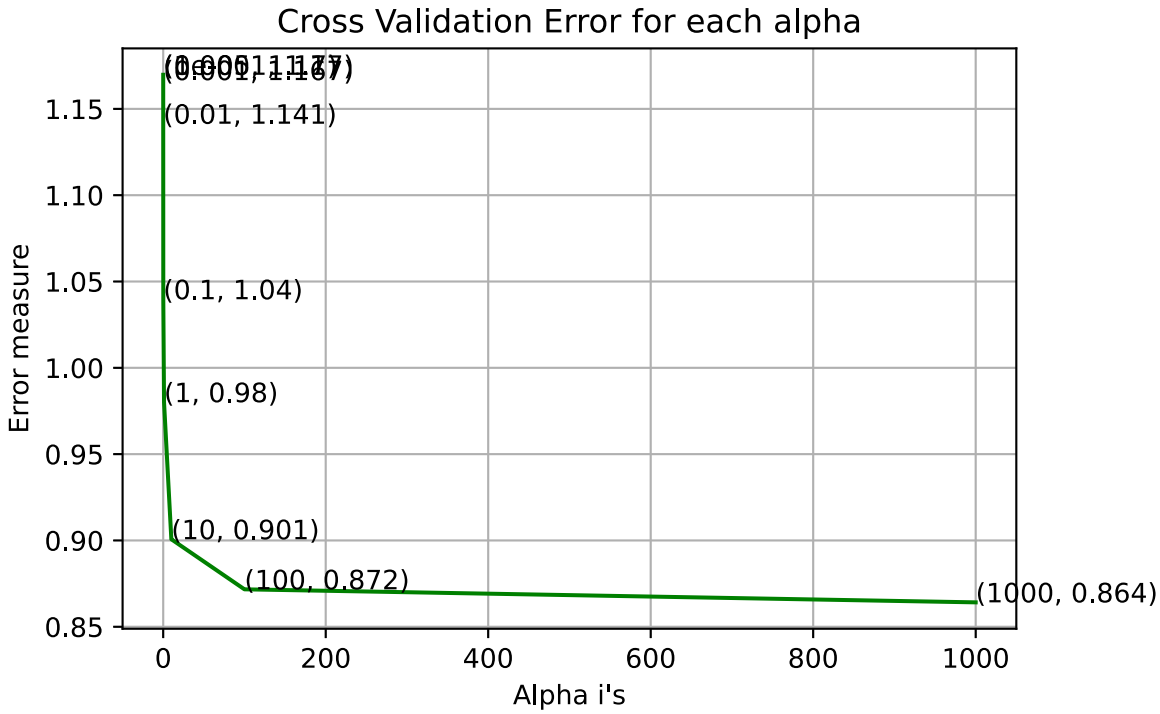
4.1.3. Logistic Regression

In [18]:

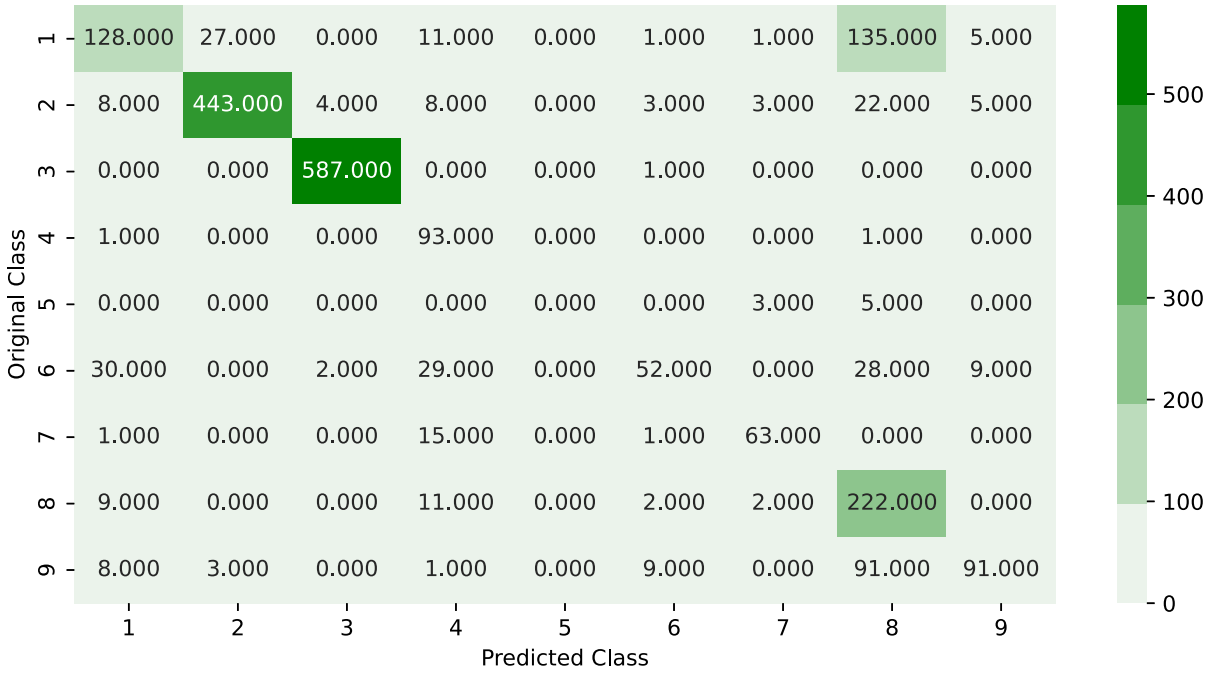
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/g
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online
#-------------------------------

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.clas

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='b
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=logistic
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.clas
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

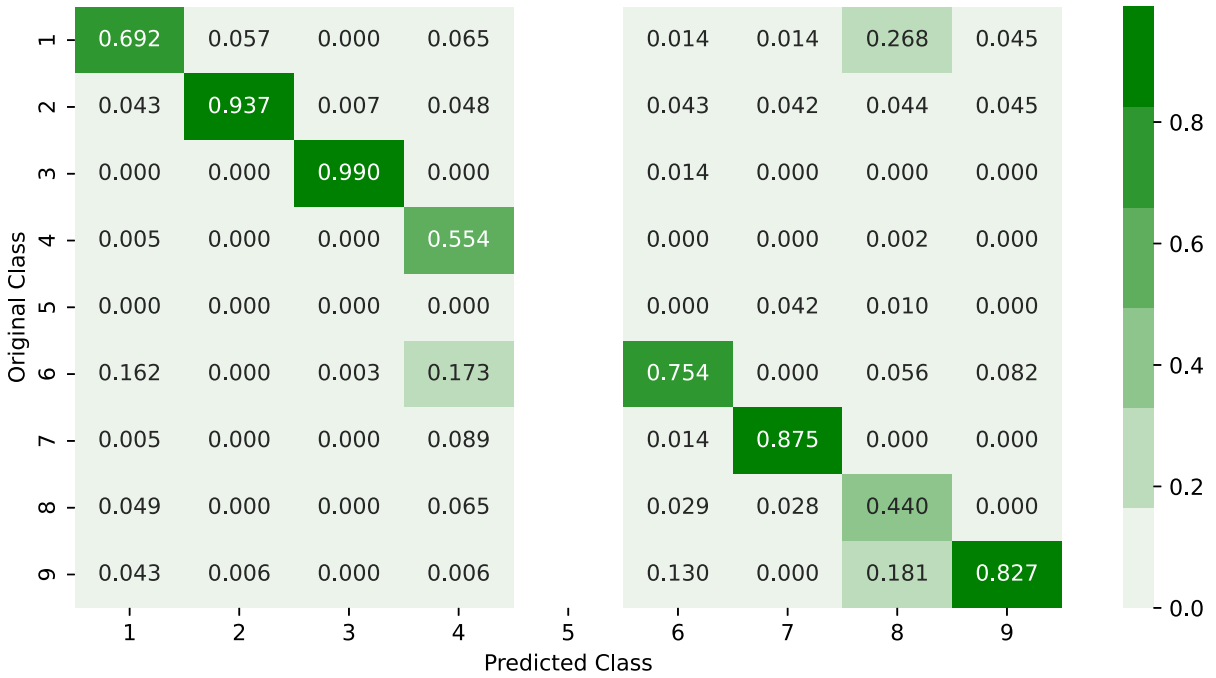
```
log_loss for c =  1e-05 is 1.1697580403855528
log_loss for c =  0.0001 is 1.169729354979023
log_loss for c =  0.001 is 1.166673594208719
log_loss for c =  0.01 is 1.1413872860160295
log_loss for c =  0.1 is 1.0397258347257718
log_loss for c =  1 is 0.980104718070747
log_loss for c =  10 is 0.900700048659595
log_loss for c =  100 is 0.8716894854472438
log_loss for c =  1000 is 0.8641380281529061
```




```
log loss for train data 0.8531564304184575
log loss for cv data 0.8641380281529061
log loss for test data 0.849541034481608
Number of misclassified points  22.769089236430542
---------------------------------------------------- Confusion matrix ----------
----------------------------------------
```

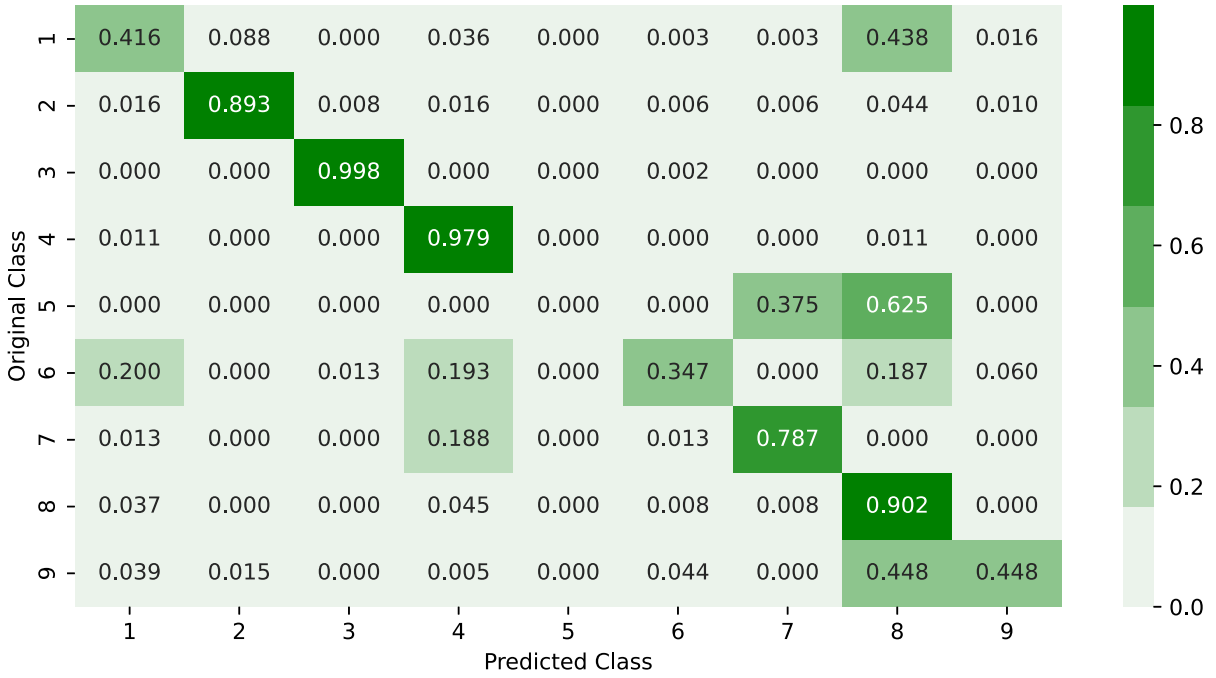


---------- **Precision matrix** ----------



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.]
```

4.1.4. Random Forest Classifier

In [ ]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', 
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online
# --------------------------------

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

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_
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

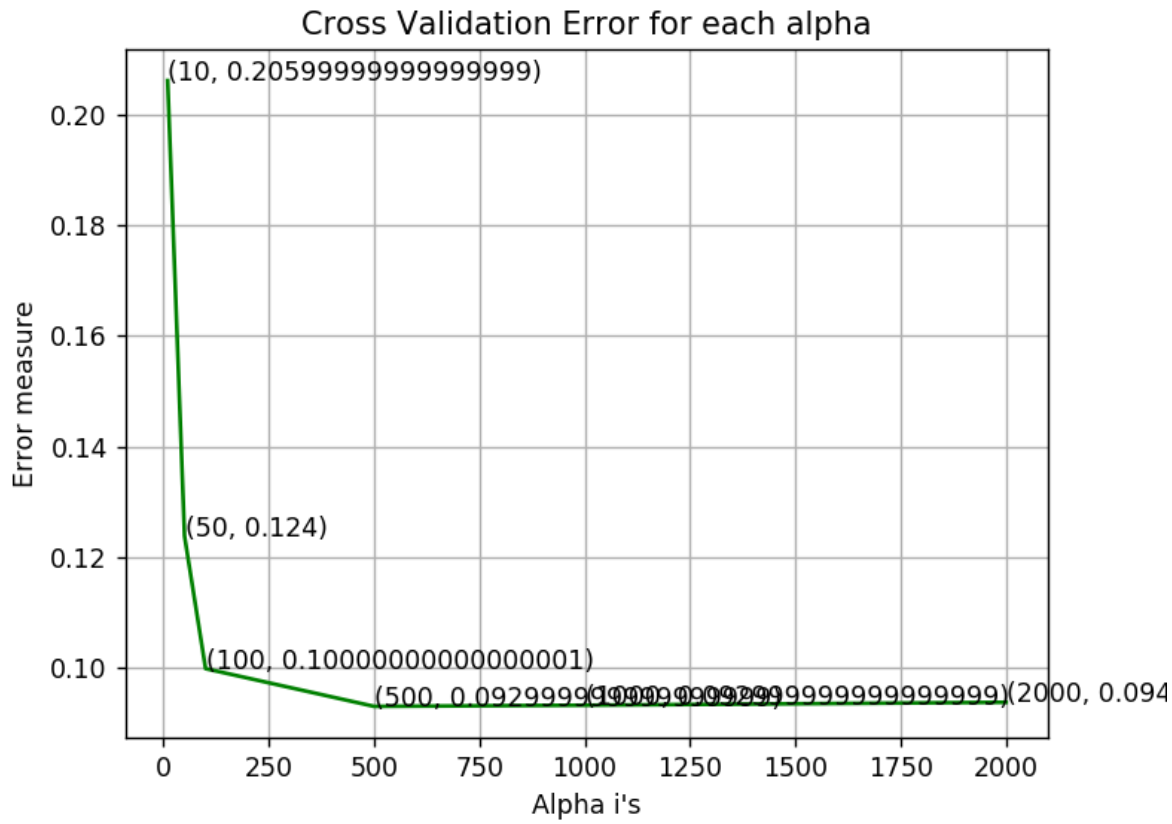
4.1.5. XgBoost Classification

In [ ]:

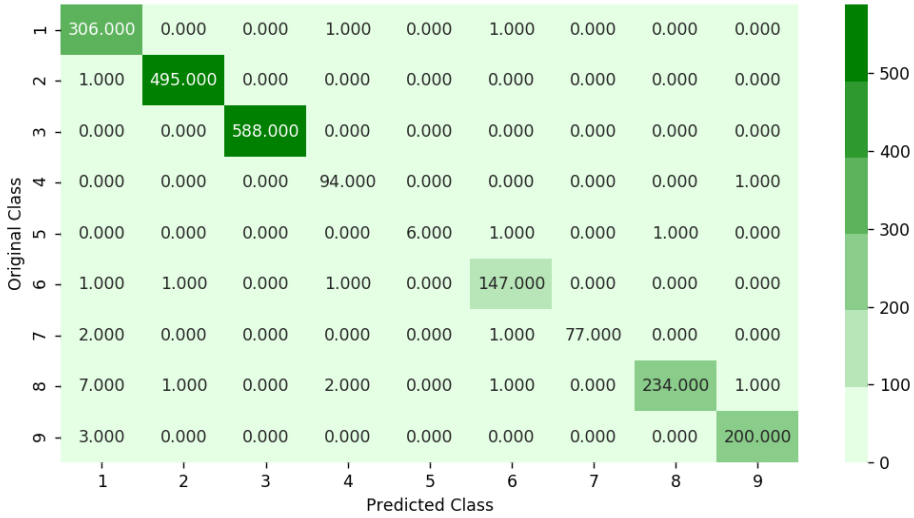
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# -----------------------

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_

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

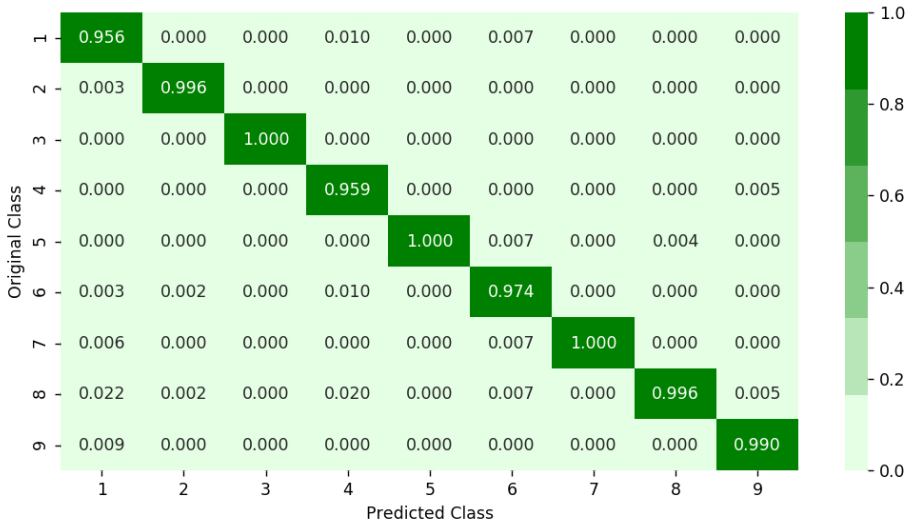
```
log_loss for c =  10 is 0.20615980494
log_loss for c =  50 is 0.123888382365
log_loss for c =  100 is 0.099919437112
log_loss for c =  500 is 0.0931035681289
log_loss for c =  1000 is 0.0933084876012
log_loss for c =  2000 is 0.0938395690309
```




```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.0931035681
289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
---------------------------------------------------- Confusion matrix ----------
----------------------------------------
```

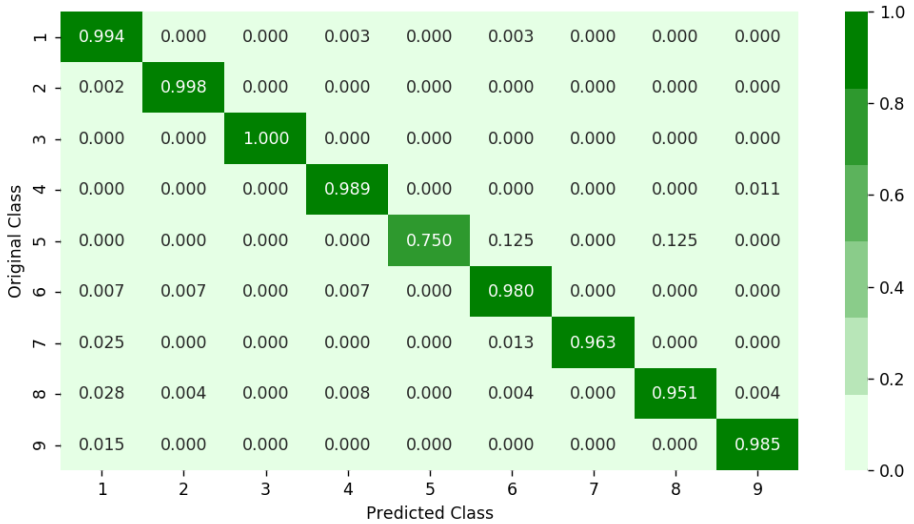


---------- Precision matrix ----------



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.]

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

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

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  9.3min remaining:  5.4m
in
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 10.1min remaining:  3.1m
in
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6m
in
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 14.2min finished
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsamp
le_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15,
0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'c
olsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [ ]:
```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.05,
'colsample_bytree': 0.5}
```

In [ ]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
```

```
# ----------------------

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

```
train loss 0.022540976086
cv loss 0.0928710624158
test loss 0.0782688587098
```

4.2 Modeling with .asm files

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


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

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

4.2.1 Feature extraction from asm files

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

In [25]:

```
#intially create five folders
#first
#second
#third
#fourth
#fifth
```

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

source='asmFiles/'
files = os.listdir('asmFiles')
#ID=df['Id'].tolist()
data=range(0,10868)
data = list(data)

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 [26]:

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

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

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata
    #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', 'sul
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special register
    #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 f
```

```
            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' seg
                        if registers[i] in li and ('text' in l or 'CODE' in l
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


#same as above
def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sul
    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
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
```

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

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sul
    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 f
            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
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sul
    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
            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
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sul
    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 f
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


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

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

```
Process Process-2:
Traceback (most recent call last):
Process Process-3:
```

```
  File "/usr/lib/python3.8/multiprocessing/process.py", line 315, in _bootstra
p
    self.run()
  File "/usr/lib/python3.8/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
Traceback (most recent call last):
Process Process-4:
  File "/tmp/ipykernel_26680/5381302.py", line 30, in firstprocess
    opcodefile.write(f2+" ")
  File "/usr/lib/python3.8/multiprocessing/process.py", line 315, in _bootstra
p
    self.run()
  File "/usr/lib/python3.8/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
NameError: name 'opcodefile' is not defined
  File "/tmp/ipykernel_26680/5381302.py", line 88, in secondprocess
    opcodefile.write(f2+" ")
Traceback (most recent call last):
Process Process-5:
NameError: name 'opcodefile' is not defined
  File "/usr/lib/python3.8/multiprocessing/process.py", line 315, in _bootstra
p
    self.run()
  File "/usr/lib/python3.8/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/tmp/ipykernel_26680/5381302.py", line 136, in thirdprocess
    opcodefile.write(f2+" ")
NameError: name 'opcodefile' is not defined
Process Process-6:
Traceback (most recent call last):
  File "/usr/lib/python3.8/multiprocessing/process.py", line 315, in _bootstra
p
    self.run()
  File "/usr/lib/python3.8/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
Traceback (most recent call last):
  File "/tmp/ipykernel_26680/5381302.py", line 184, in fourthprocess
    opcodefile.write(f2+" ")
  File "/usr/lib/python3.8/multiprocessing/process.py", line 315, in _bootstra
p
    self.run()
NameError: name 'opcodefile' is not defined
  File "/usr/lib/python3.8/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/tmp/ipykernel_26680/5381302.py", line 232, in fifthprocess
    opcodefile.write(f2+" ")
NameError: name 'opcodefile' is not defined
```

In [12]:

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

Out[12]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.r
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.r
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	

5 rows × 53 columns

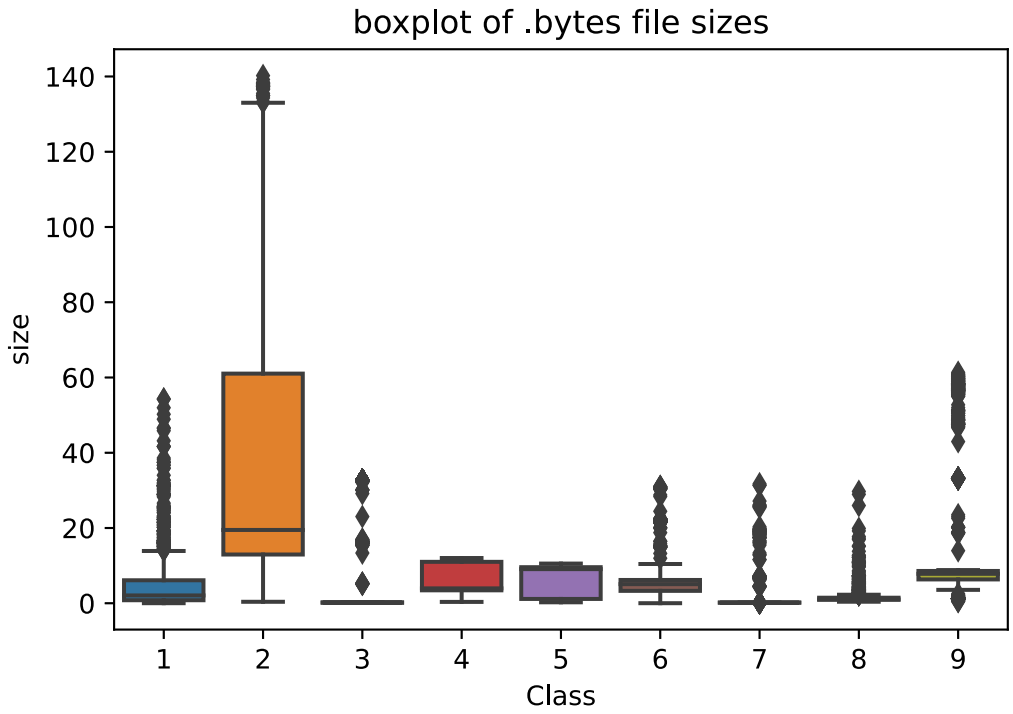
4.2.1.1 Files sizes of each .asm file

In [13]:
```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=356157170
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file n
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes}
print (asm_size_byte.head())
```

```
                     ID       size  Class
0  2QJ6Xc5LRmWZkdnVBtoM  71.258157      2
1  2HqSerDI4yG1WRxTbl8w   7.780184      9
2  0yBQ5dAc2gN4qx9YWuiS  10.672950      1
3  2xmv5zSqBlakorbUw9MJ   0.171156      3
4  0iZTHuQ5KMb4RtAlrz6D   6.012874      6
```

4.2.1.2 Distribution of .asm file sizes

In [22]:
```
#boxplot of asm files
ax = sns.boxplot(x="Class", y="size", data=asm_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



In [14]:
```
# add the file size feature to previous extracted features
print(result_asm.shape)
print(asm_size_byte.shape)
result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='I
result_asm.head()
```

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

Out[14]:

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

5 rows × 54 columns

In [15]:
```
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

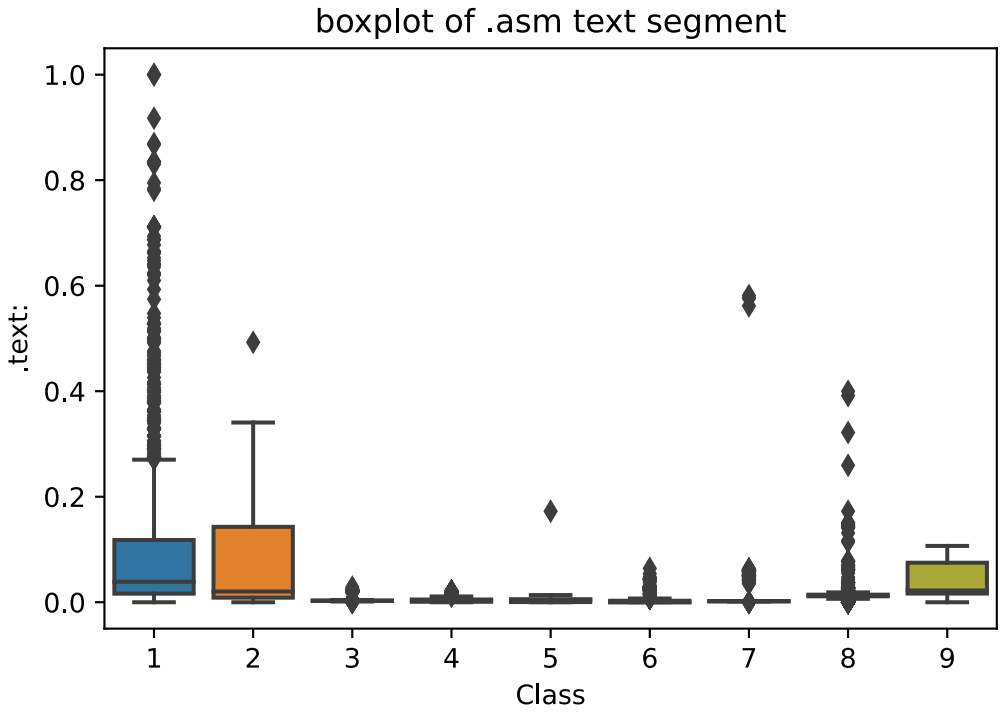
Out[15]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rd
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000

5 rows × 54 columns

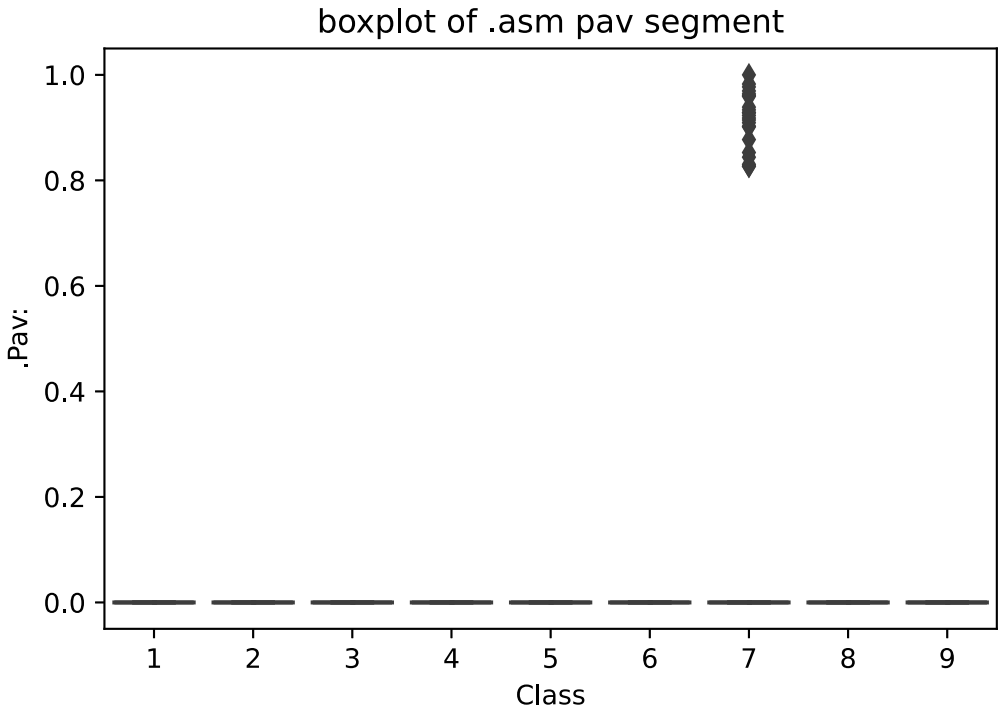
4.2.2 Univariate analysis on asm file features

In [25]:
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```

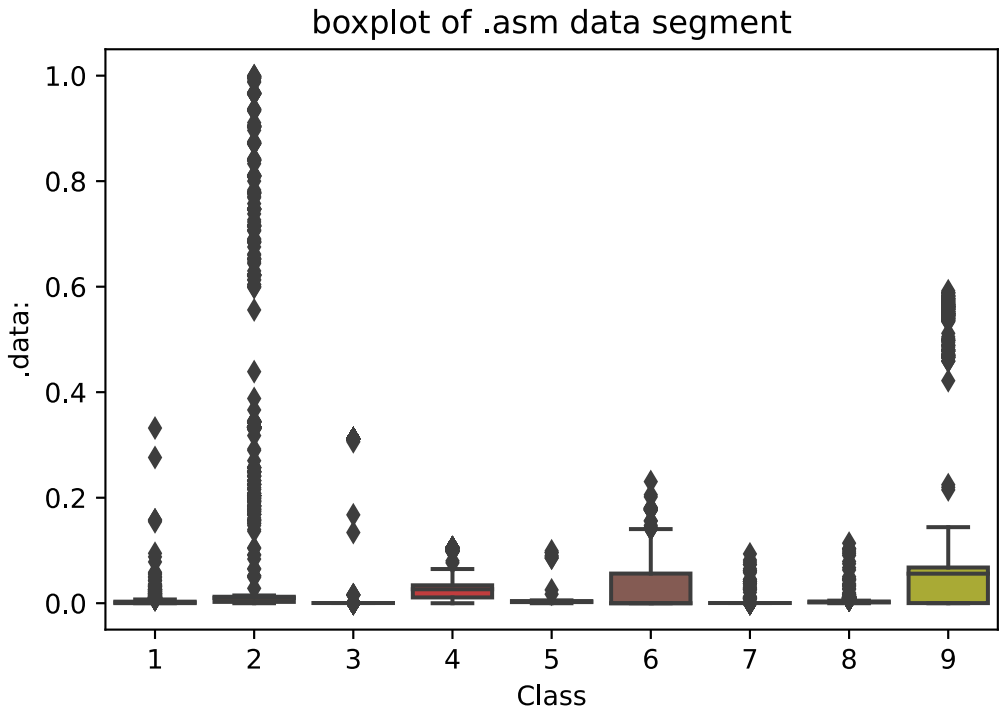


The plot is between Text and class
Class 1,2 and 9 can be easly separated

In [26]:
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```

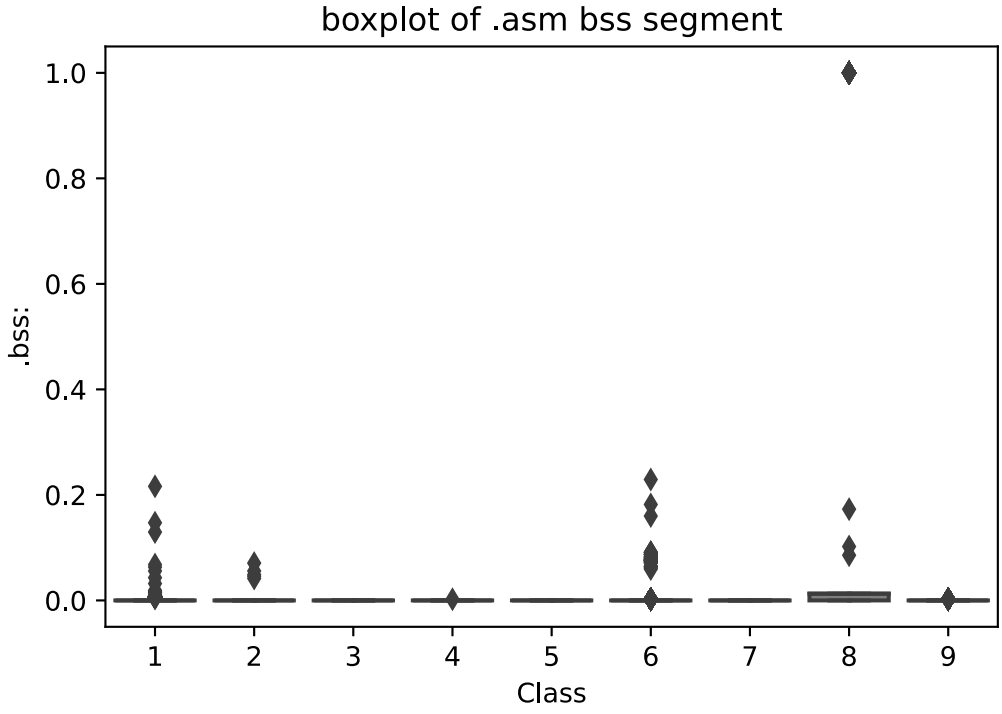


In [27]:
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



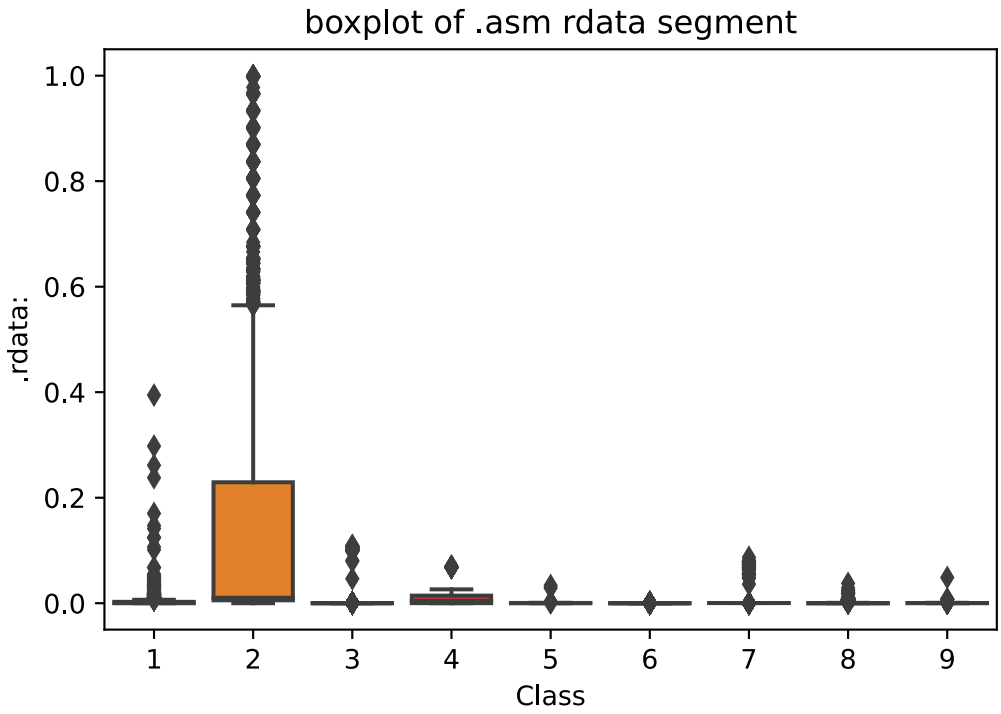
The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [28]:
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



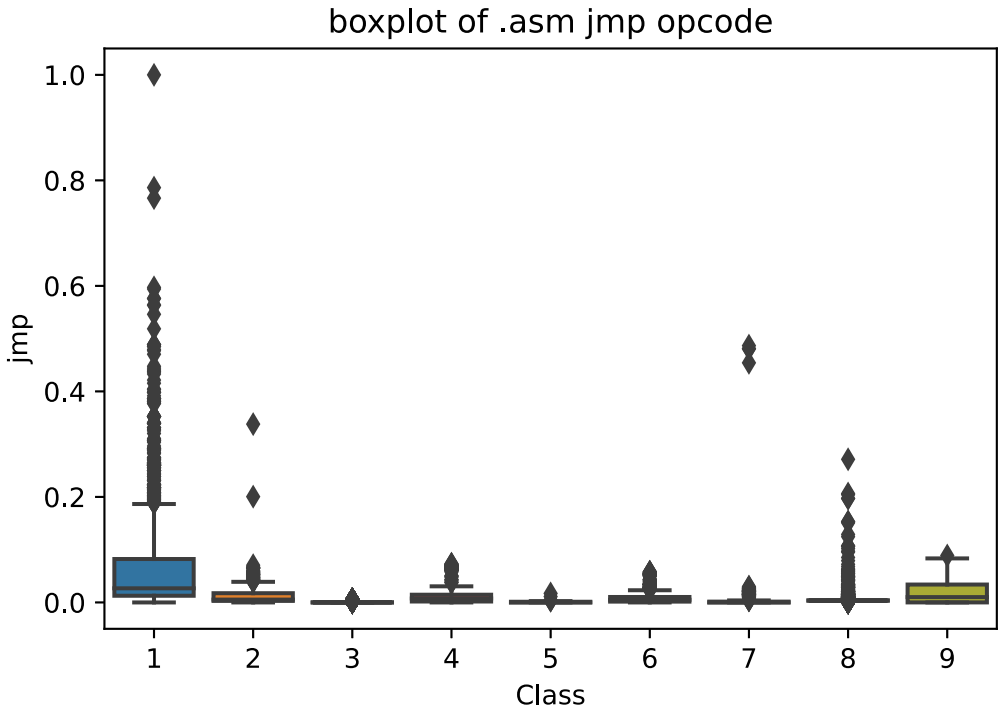
plot between bss segment and class label
very less number of files are having bss segment

In [29]:
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

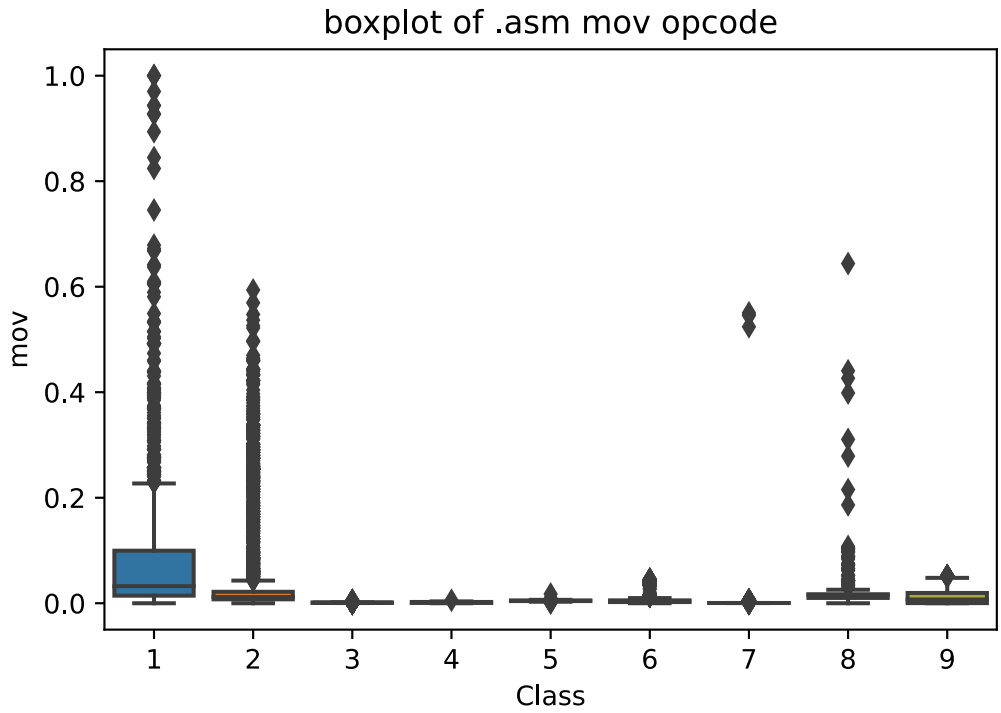
In [30]:
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



plot between jmp and Class label

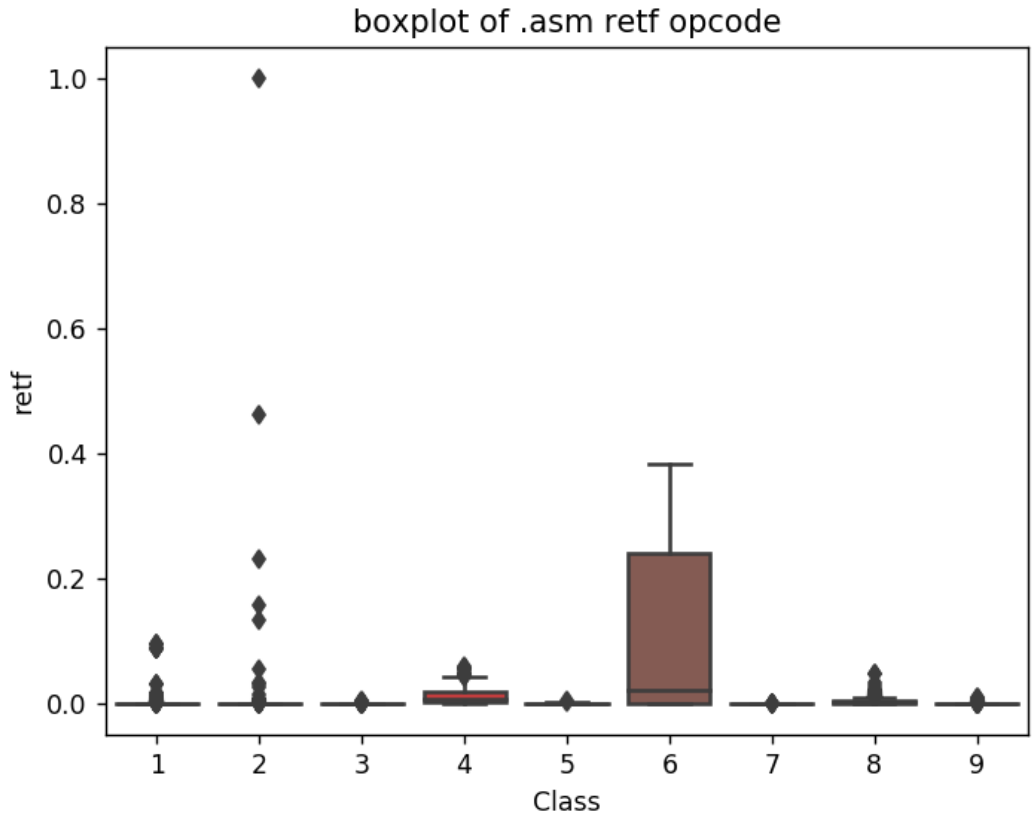
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [31]:
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



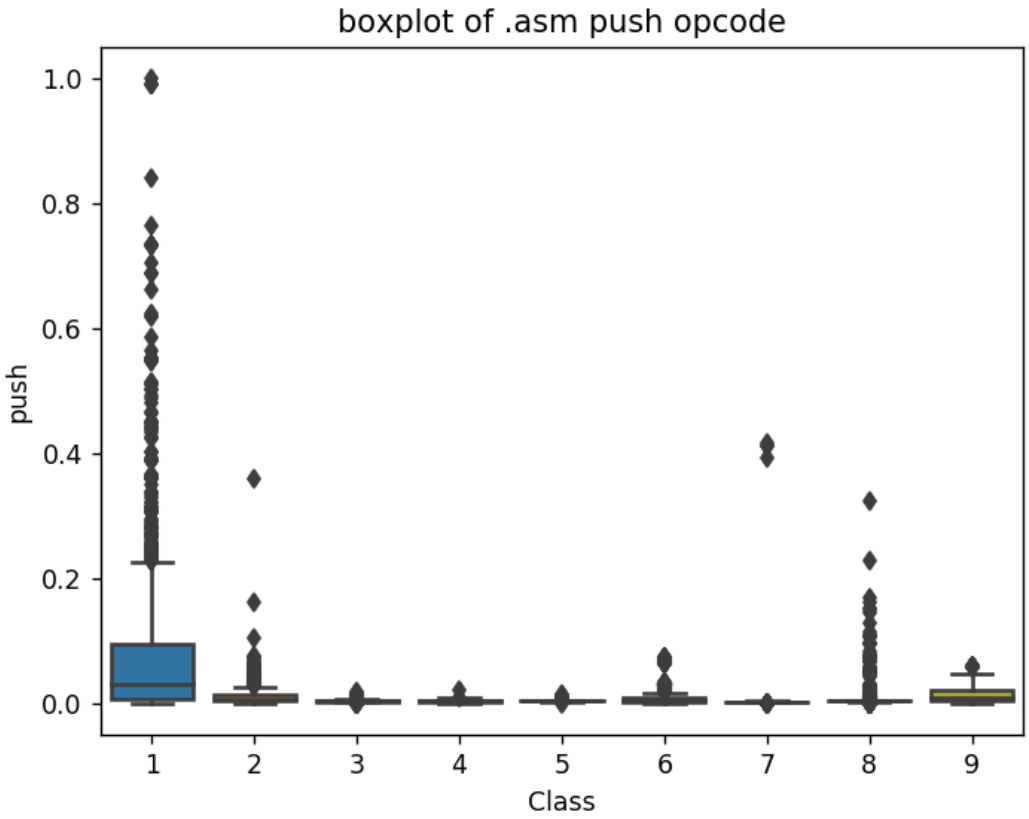
plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

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



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

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

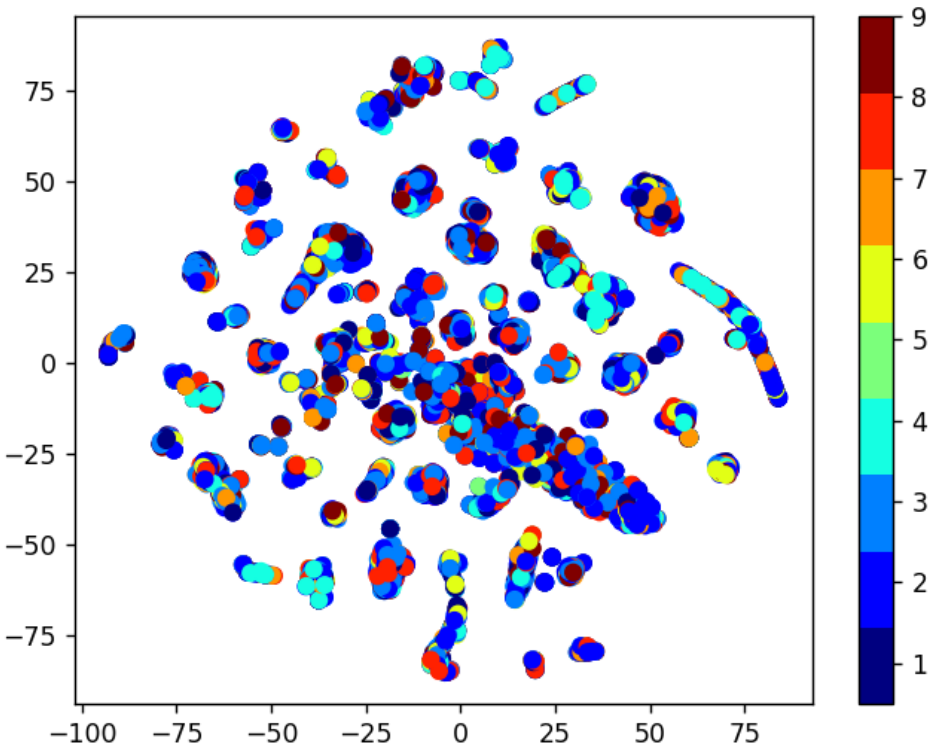


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

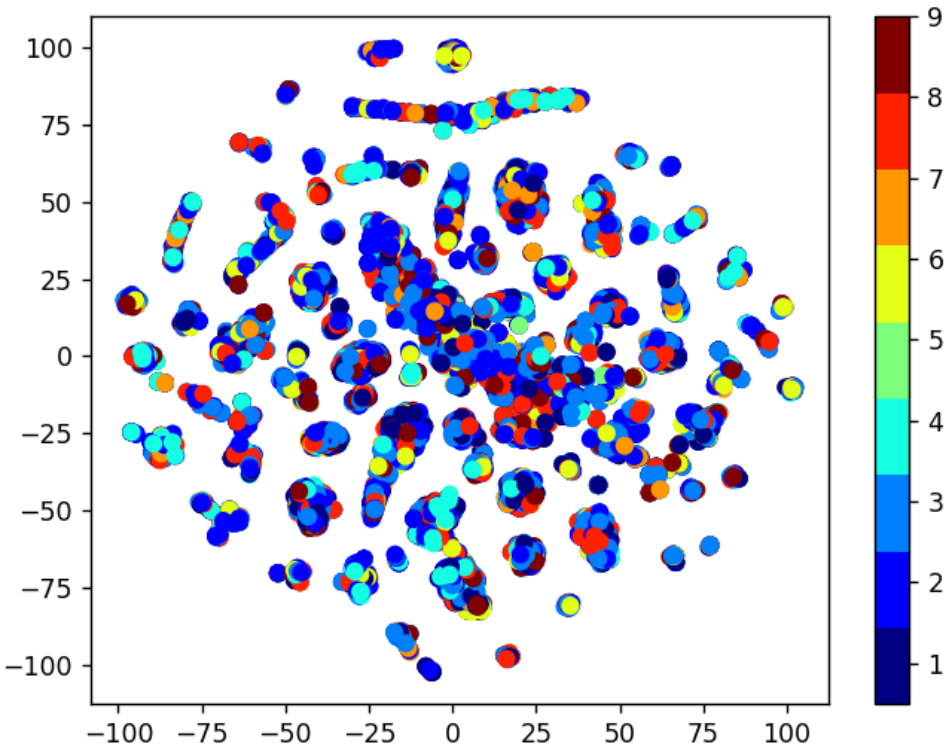
In [ ]:

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

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



In [ ]:

```
# by univariate analysis on the .asm file features we are getting very neglig
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analys
# the plot looks very messy

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



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [16]:
```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

In [17]:
```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm,
```

In [20]:
```
print( X_cv_asm.isnull().all())
```

```
HEADER:    False
.text:     False
.Pav:      False
.idata:    False
```

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

4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

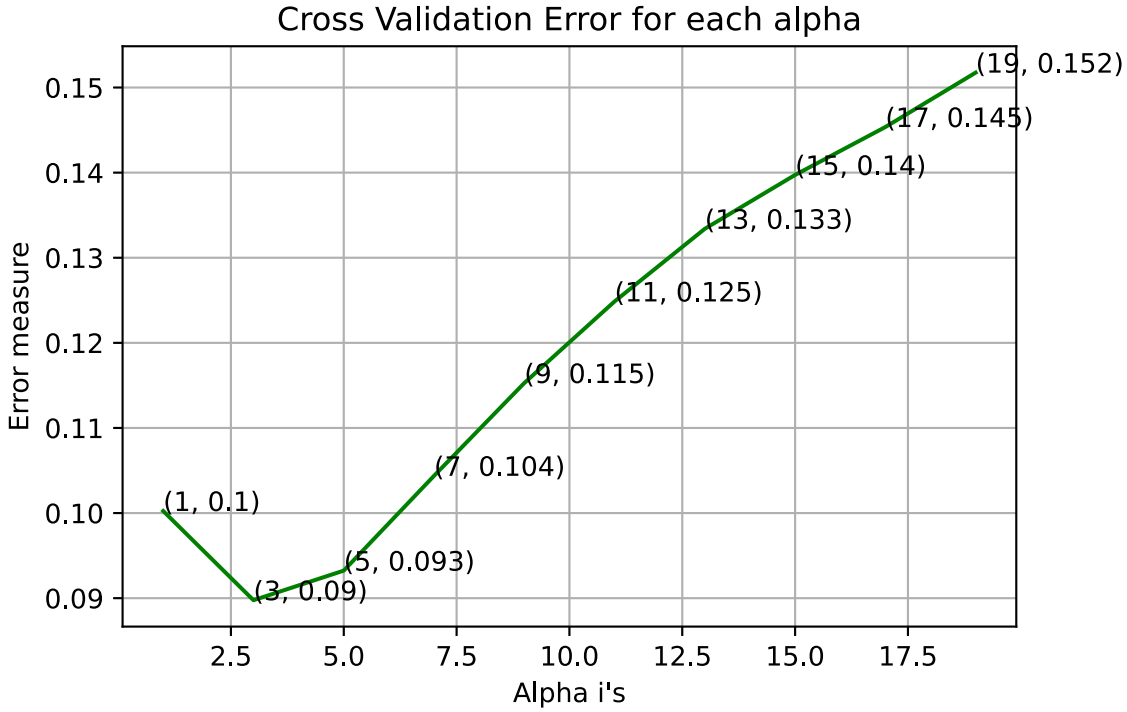
In [35]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', le
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online
```

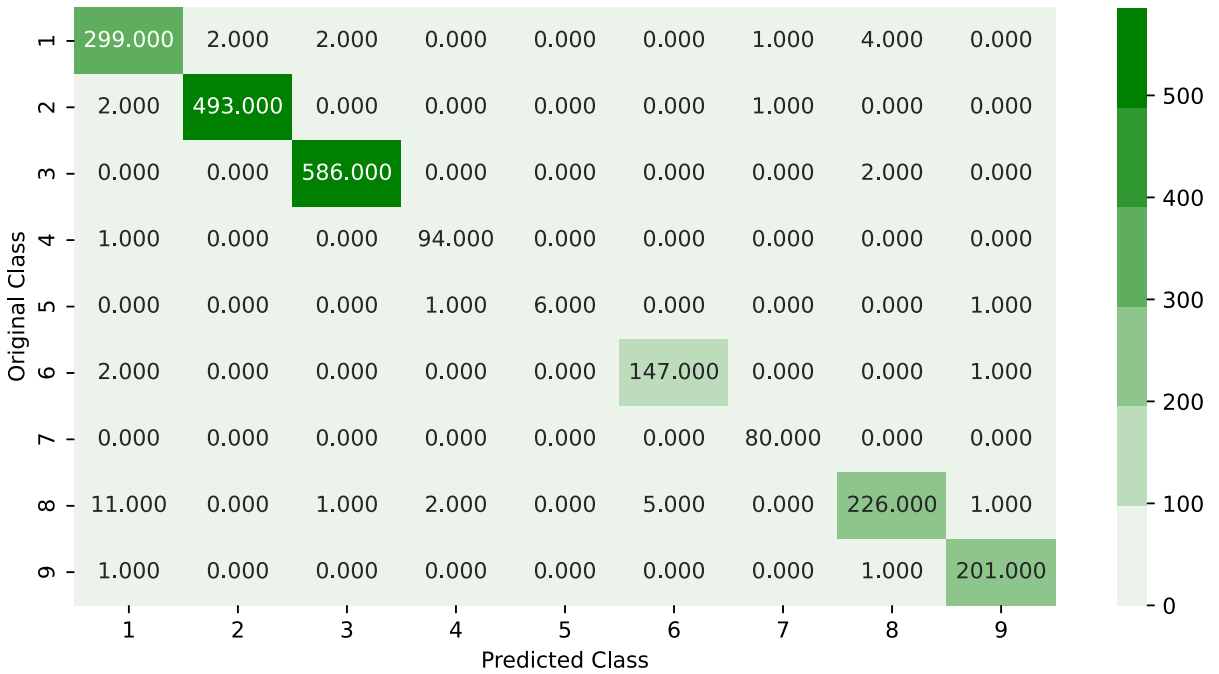
```
#--------------------------------------


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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

```
log_loss for k =  1 is 0.10024350446210807
log_loss for k =  3 is 0.08976087516713856
log_loss for k =  5 is 0.0932384998888478
log_loss for k =  7 is 0.10437622296424091
log_loss for k =  9 is 0.11527999945519254
log_loss for k =  11 is 0.12486380602083774
log_loss for k =  13 is 0.13340095595265572
log_loss for k =  15 is 0.13972882394255248
```

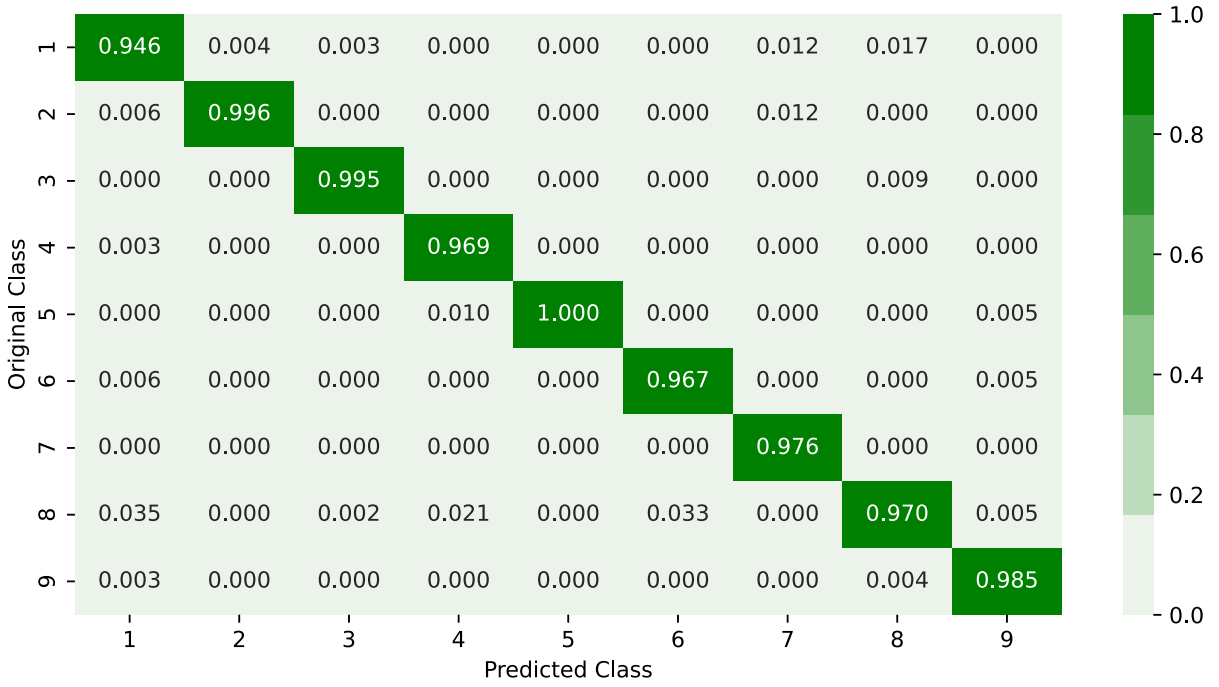
```
log_loss for k =  17 is 0.1453580006159046
log_loss for k =  19 is 0.15174842729901064
```




```
log loss for train data 0.04506703652550648
log loss for cv data 0.08976087516713856
log loss for test data 0.09767573047985882
Number of misclassified points  1.9319227230910765
---------------------------------------------------- Confusion matrix ----------
------------------------------------------
```

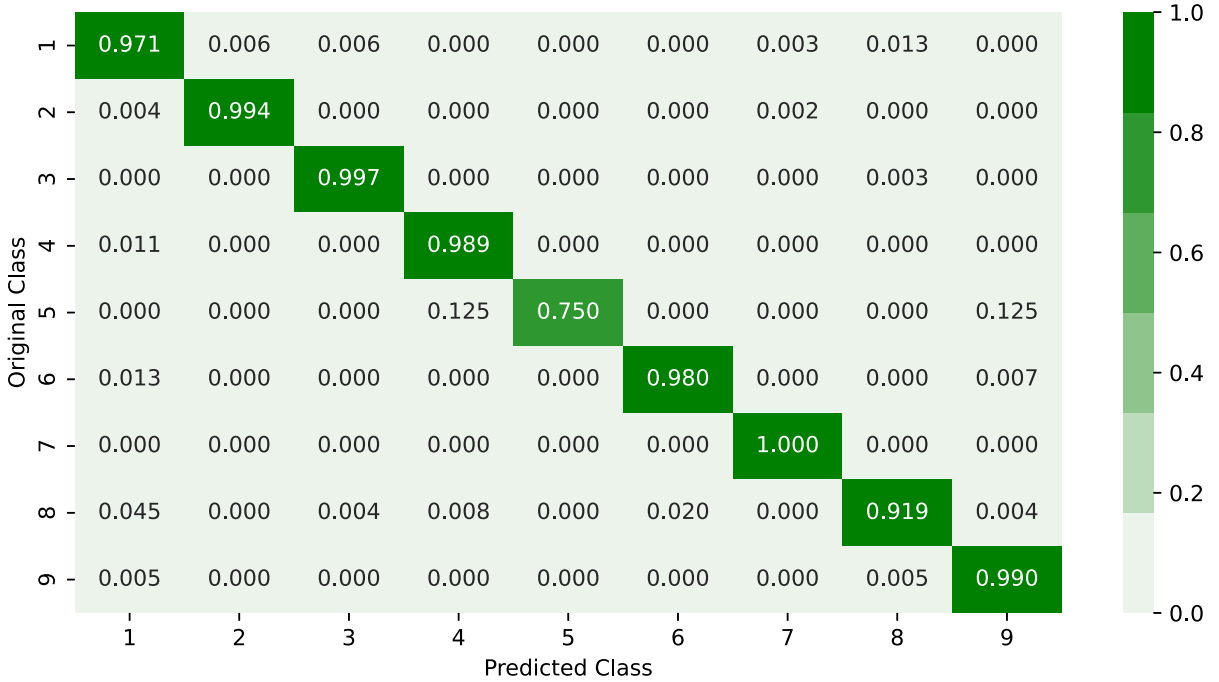



```
---------------------------------------------------- Precision matrix ----------
------------------------------------------
```



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.]

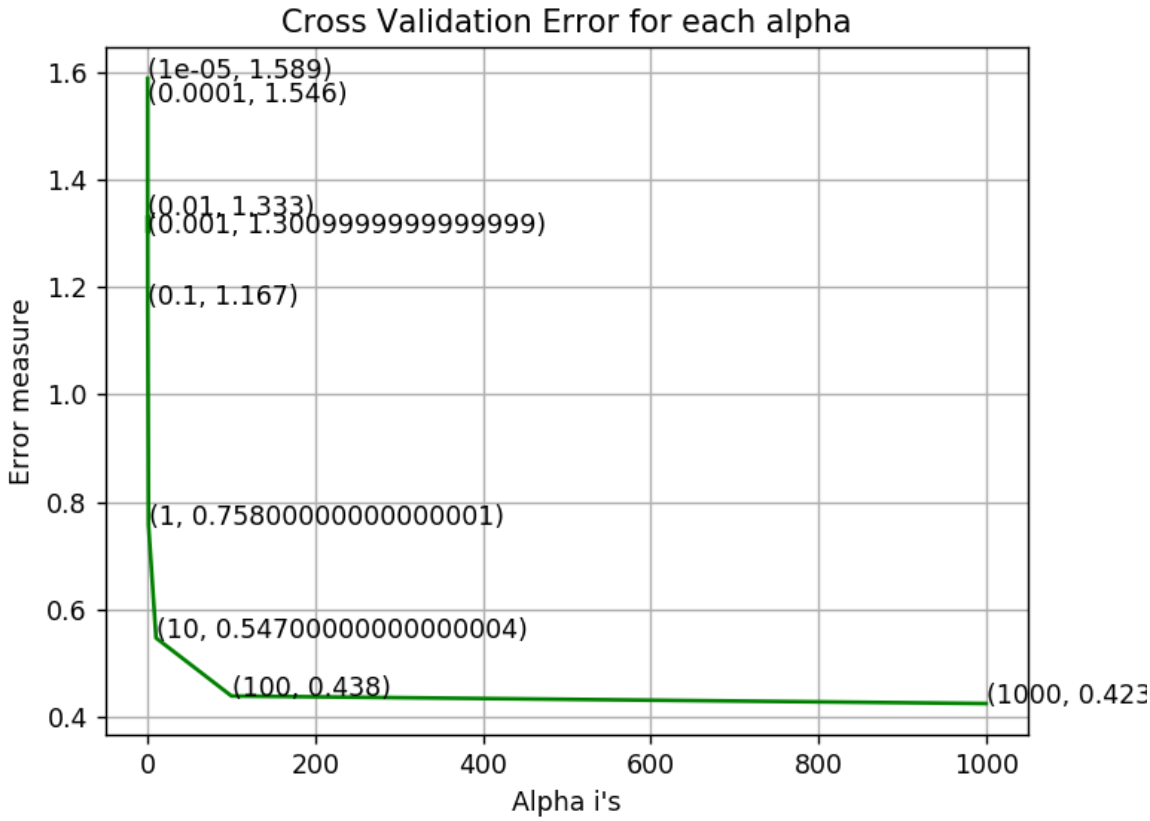
4.4.2 Logistic Regression

In [ ]:

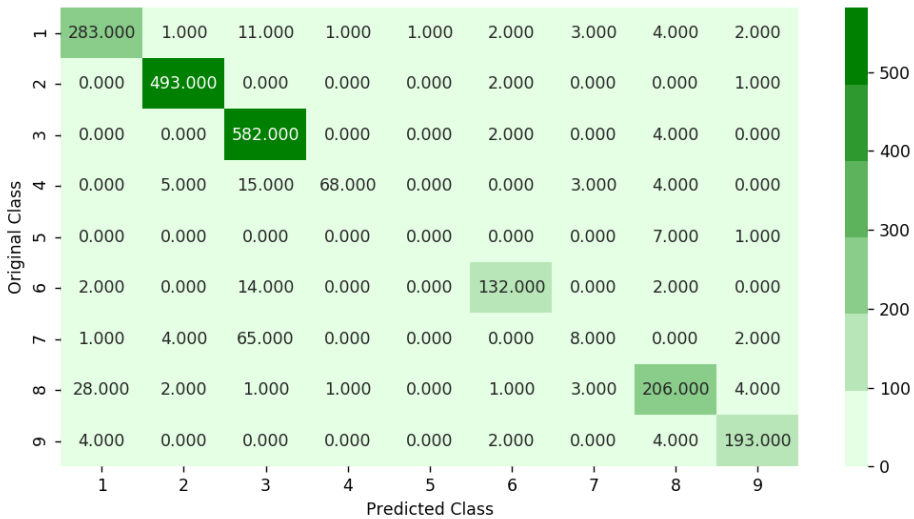
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/g
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#------------------------------
```

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online
#------------------------------


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

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

best_alpha = np.argmin(cv_log_error_array)

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

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='ba
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=log
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logis
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

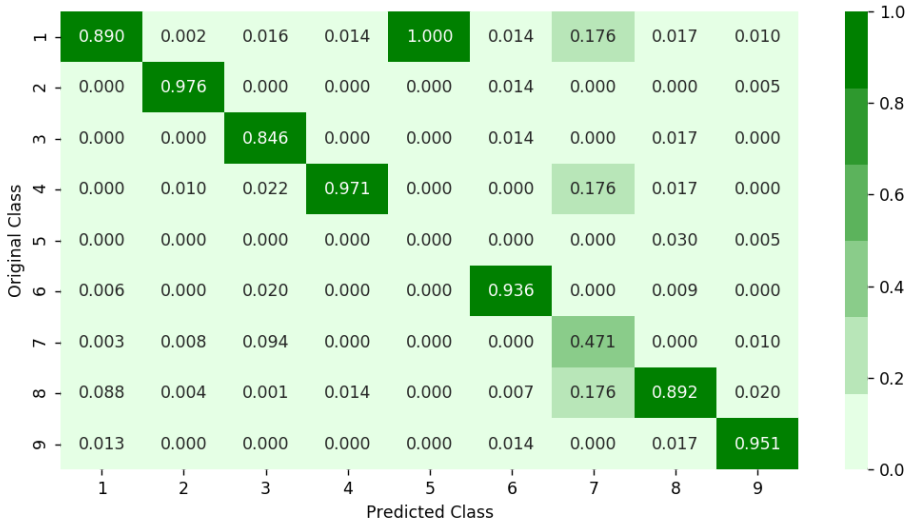
```
log_loss for c =  1e-05 is 1.58867274165
log_loss for c =  0.0001 is 1.54560797884
log_loss for c =  0.001 is 1.30137786807
log_loss for c =  0.01 is 1.33317456931
log_loss for c =  0.1 is 1.16705751378
log_loss for c =  1 is 0.757667807779
log_loss for c =  10 is 0.546533939819
log_loss for c =  100 is 0.438414998062
log_loss for c =  1000 is 0.424423536526
```



log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points 9.61361545538
---------- Confusion matrix ----------

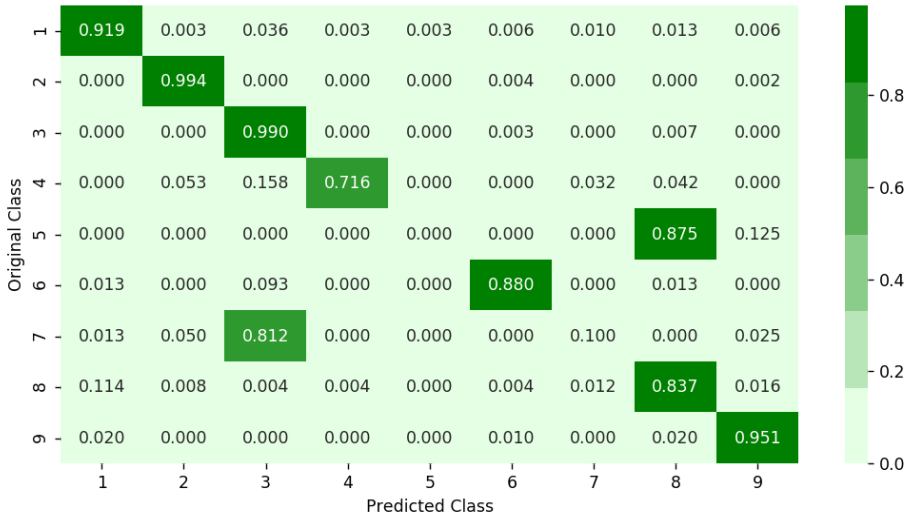


---------- Precision matrix ----------



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.]

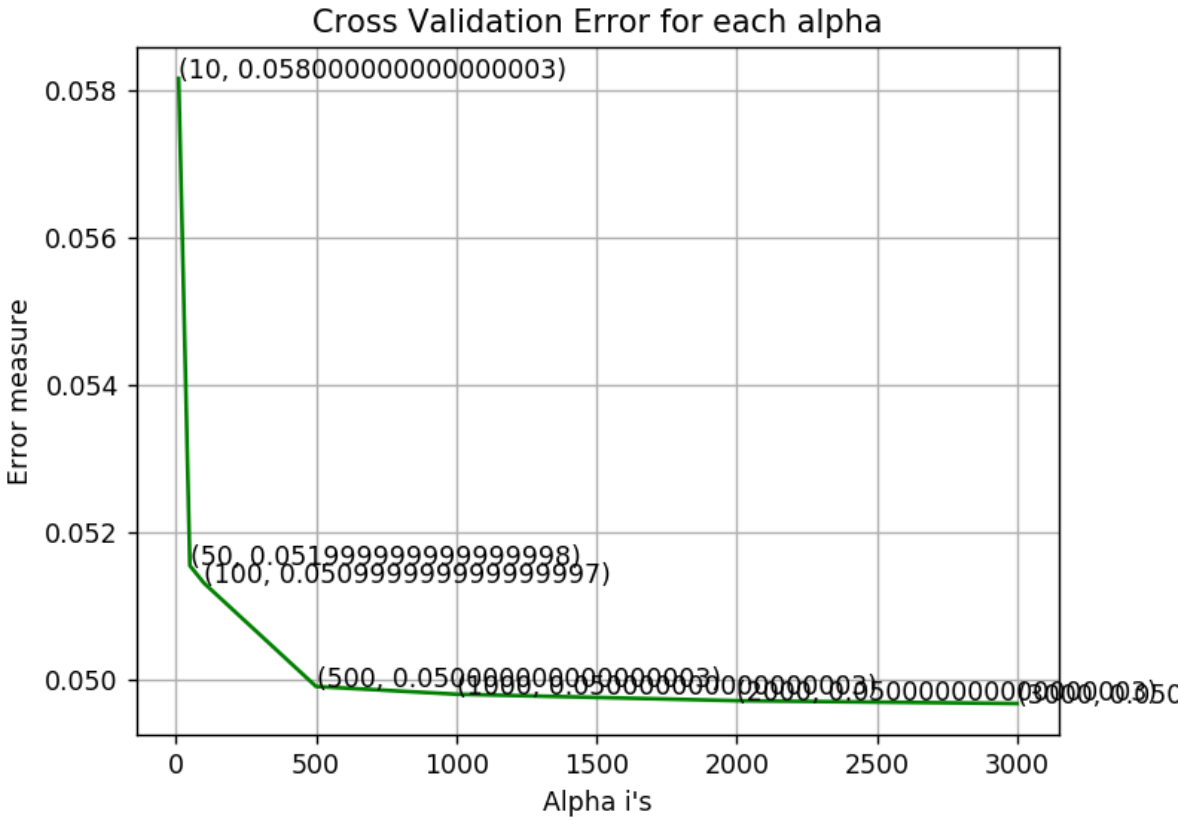
4.4.3 Random Forest Classifier

In [ ]:

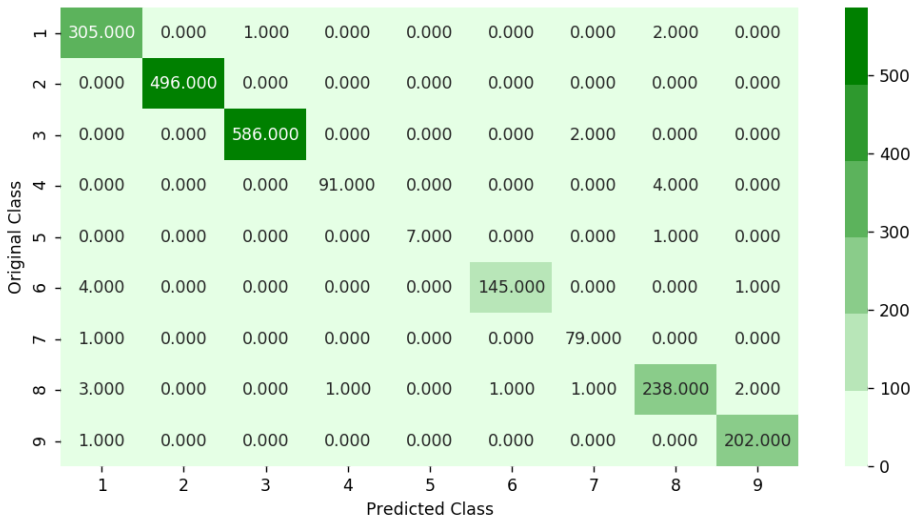
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', 
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_
# class_weight=None)

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

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

# -------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online
```

```
# --------------------------------

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

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
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
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.c
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_c
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

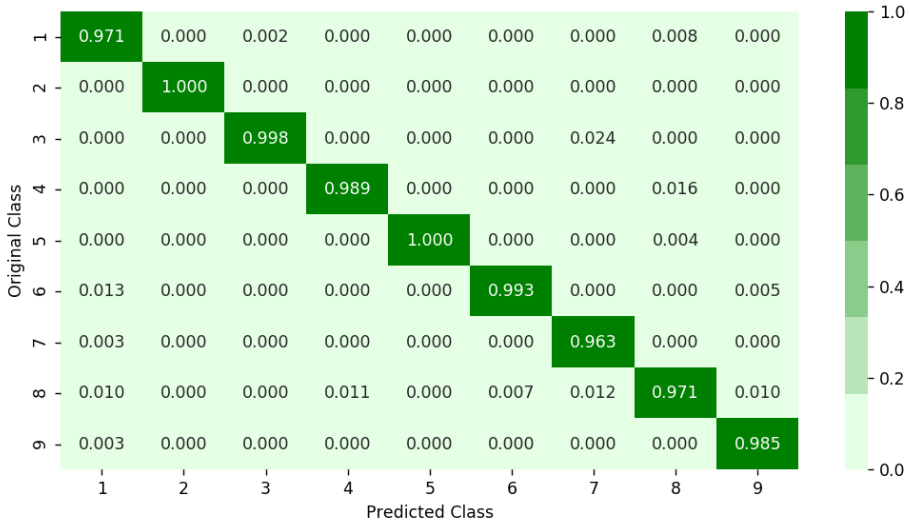
```
log_loss for c =  10 is 0.0581657906023
log_loss for c =  50 is 0.0515443148419
log_loss for c =  100 is 0.0513084973231
log_loss for c =  500 is 0.0499021761479
log_loss for c =  1000 is 0.0497972474298
log_loss for c =  2000 is 0.0497091690815
log_loss for c =  3000 is 0.0496706817633
```



log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points 1.14995400184
---------- Confusion matrix ----------

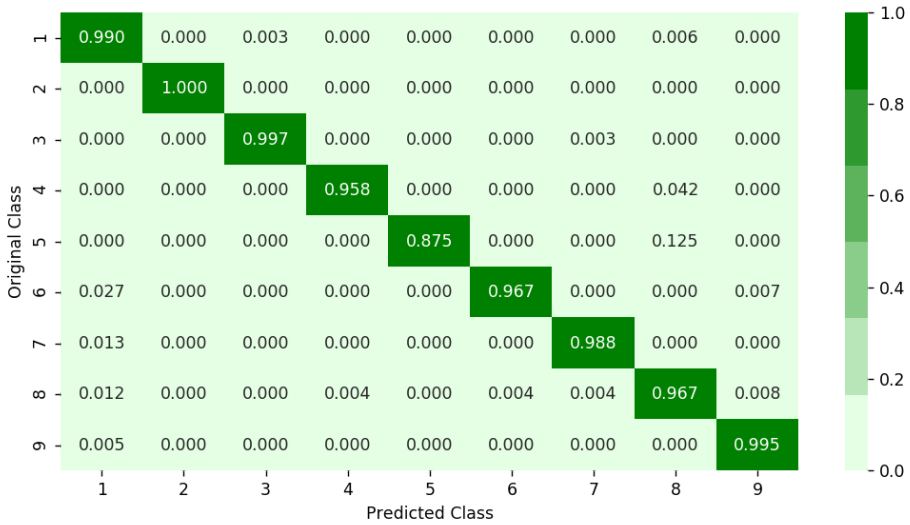


---------- Precision matrix ----------



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.]

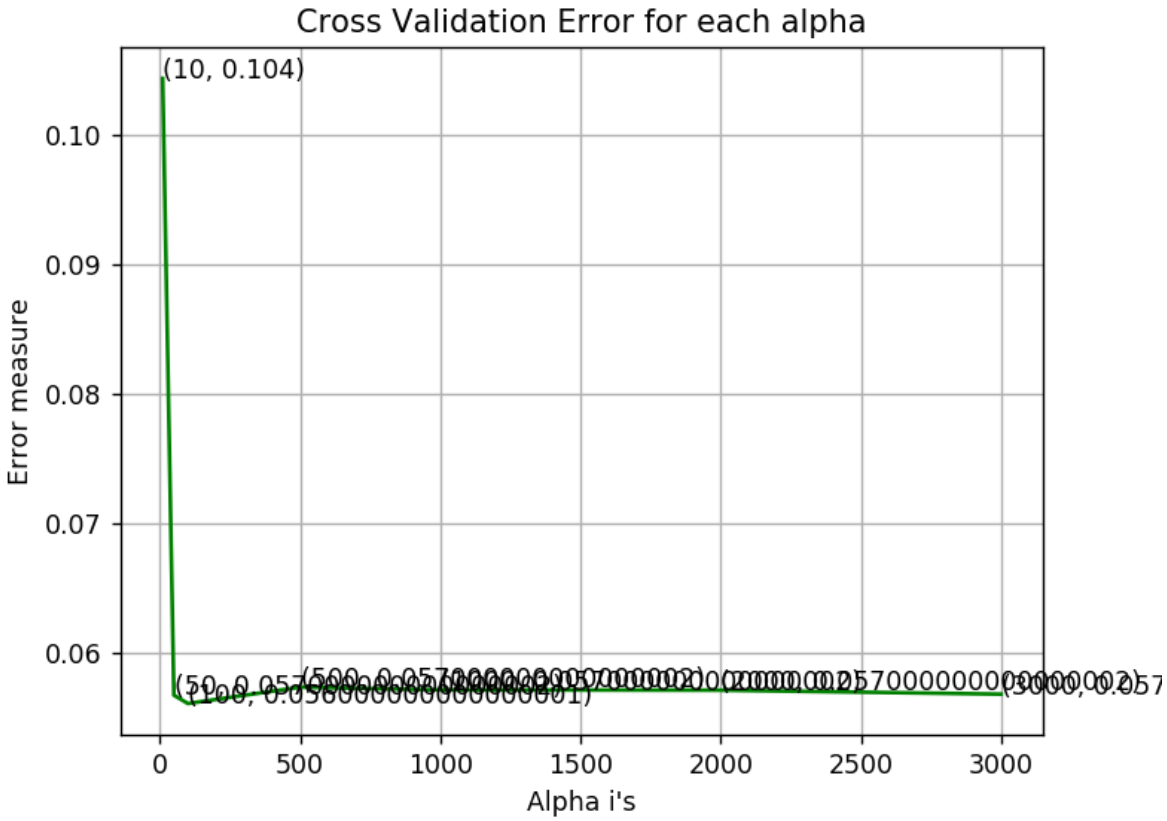
4.4.4 XgBoost Classifier

In [ ]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# -----------------------
```

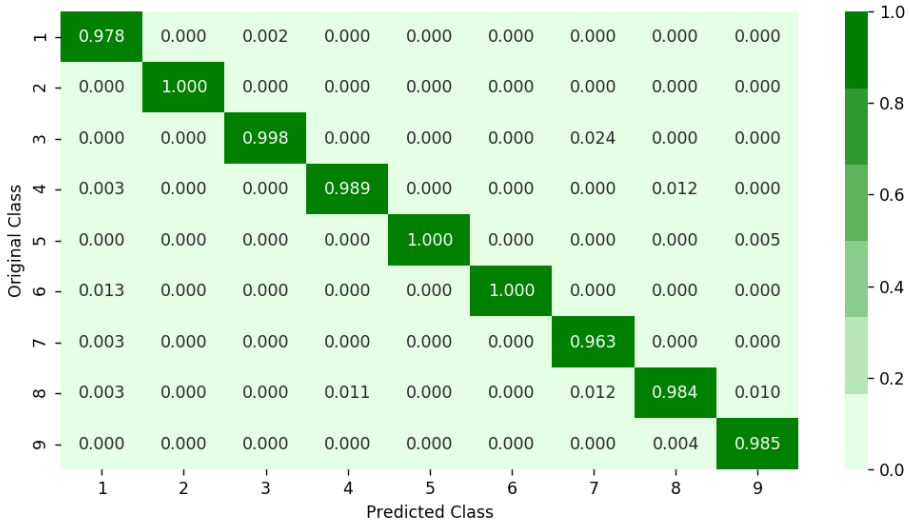
```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)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

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

```
log_loss for c =  10 is 0.104344888454
log_loss for c =  50 is 0.0567190635611
log_loss for c =  100 is 0.056075038646
log_loss for c =  500 is 0.057336051683
log_loss for c =  1000 is 0.0571265109903
log_loss for c =  2000 is 0.057103406781
log_loss for c =  3000 is 0.0567993215778
```

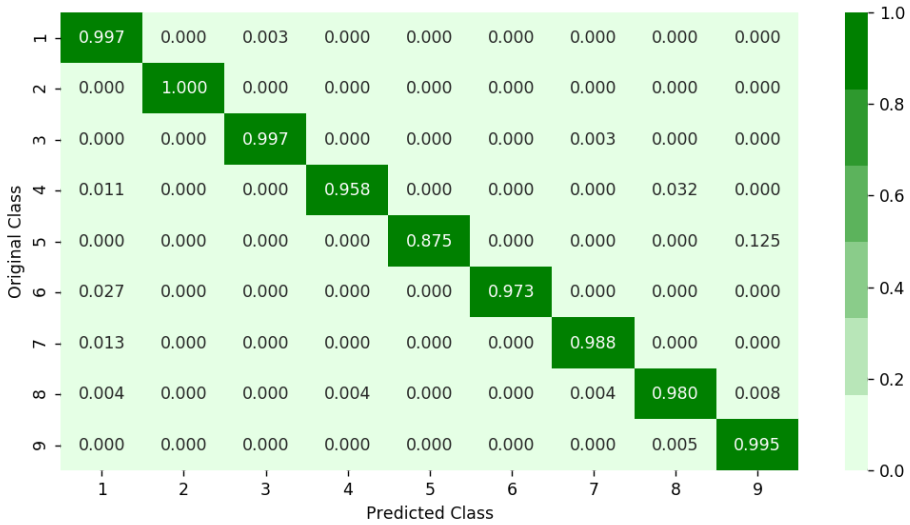


For values of best alpha = 100 The train log loss is: 0.0117883742574
For values of best alpha = 100 The cross validation log loss is: 0.056075038646
For values of best alpha = 100 The test log loss is: 0.0491647763845
Number of misclassified points 0.873965041398
---------- Confusion matrix ----------

---------- 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.]

4.4.5 Xgboost Classifier with best hyperparameters

In [ ]:

```
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  1.1min remaining:   39.
3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.3min remaining:   23.
```

```
0s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  1.4min remaining:    9.
2s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.3min finished
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsamp
le_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15,
0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'c
olsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

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

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.15,
'colsample_bytree': 0.5}
```

In [ ]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# -----------------------

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsamp
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

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

```
train loss 0.0102661325822
cv loss 0.0501201796687
test loss 0.0483908764397
```

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

4.5.1. Merging both asm and byte file features

In [18]:
```
result.head()
```

Out[18]:

	ID	0	1	2	3	4	
0	07iSOIG2urUvsMl9E5Rn	0.037960	0.088366	0.034627	0.036951	0.007875	0.00380
1	07nrG1cLKUPxjOlWMFiV	0.005980	0.011765	0.003431	0.003242	0.003633	0.00351
2	08BX5Slp2I1FraZWbc6j	0.081074	0.006632	0.000314	0.000500	0.000624	0.00022
3	09bfacpUzuBN5W3S8KTo	0.005230	0.007822	0.001832	0.001787	0.001985	0.00180
4	09CPNMYyQjSguFrE8UOf	0.029241	0.002876	0.000993	0.000966	0.002361	0.00090

5 rows × 260 columns

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

Out[2]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rd
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000

5 rows × 54 columns

In [23]:
```
print(result.shape)
print(result_asm.shape)
```

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

In [20]:
```
result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='l
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[20]:

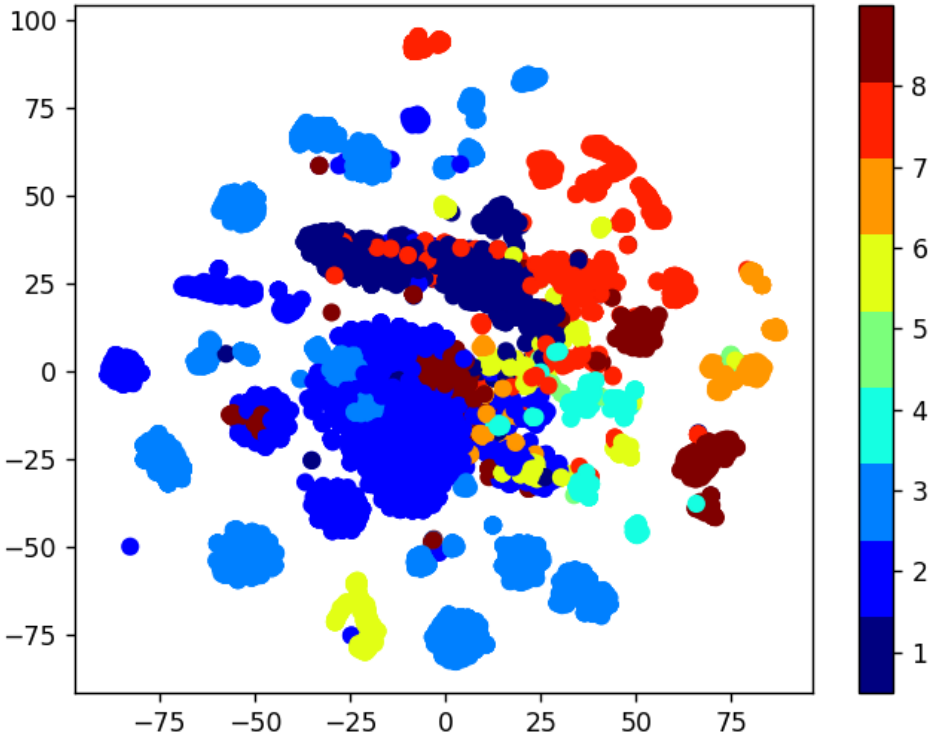
	0	1	2	3	4	5	6	7	
0	0.037960	0.088366	0.034627	0.036951	0.007875	0.003800	0.004795	0.006934	(
1	0.005980	0.011765	0.003431	0.003242	0.003633	0.003515	0.003511	0.005809	
2	0.081074	0.006632	0.000314	0.000500	0.000624	0.000225	0.000232	0.000686	
3	0.005230	0.007822	0.001832	0.001787	0.001985	0.001802	0.001841	0.003045	(
4	0.029241	0.002876	0.000993	0.000966	0.002361	0.000909	0.000953	0.001517	

5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

In [ ]:
```
xtsne=TSNE(perplexity=50)
```

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



4.5.3. Train and Test split

In [21]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, res
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_tra
```

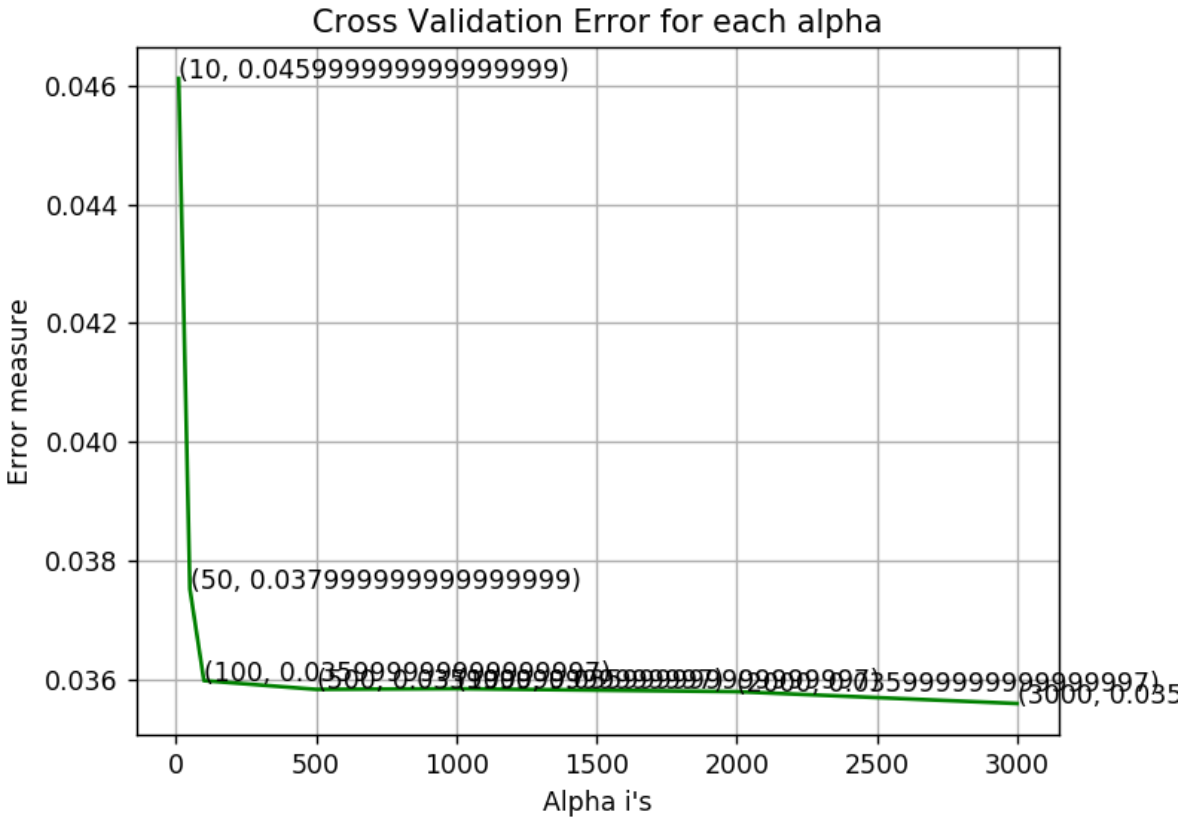
4.5.4. Random Forest Classifier on final features

In [ ]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_
# class_weight=None)

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

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

# -------------------------------
```

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online
# ------------------------------

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

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_
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
log_loss for c =  10 is 0.0461221662017
log_loss for c =  50 is 0.0375229563452
log_loss for c =  100 is 0.0359765822455
log_loss for c =  500 is 0.0358291883873
log_loss for c =  1000 is 0.0358403093496
log_loss for c =  2000 is 0.0357908022178
log_loss for c =  3000 is 0.0355909487962
```




```
For values of best alpha =  3000 The train log loss is: 0.0166267614753
For values of best alpha =  3000 The cross validation log loss is: 0.035590948
7962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

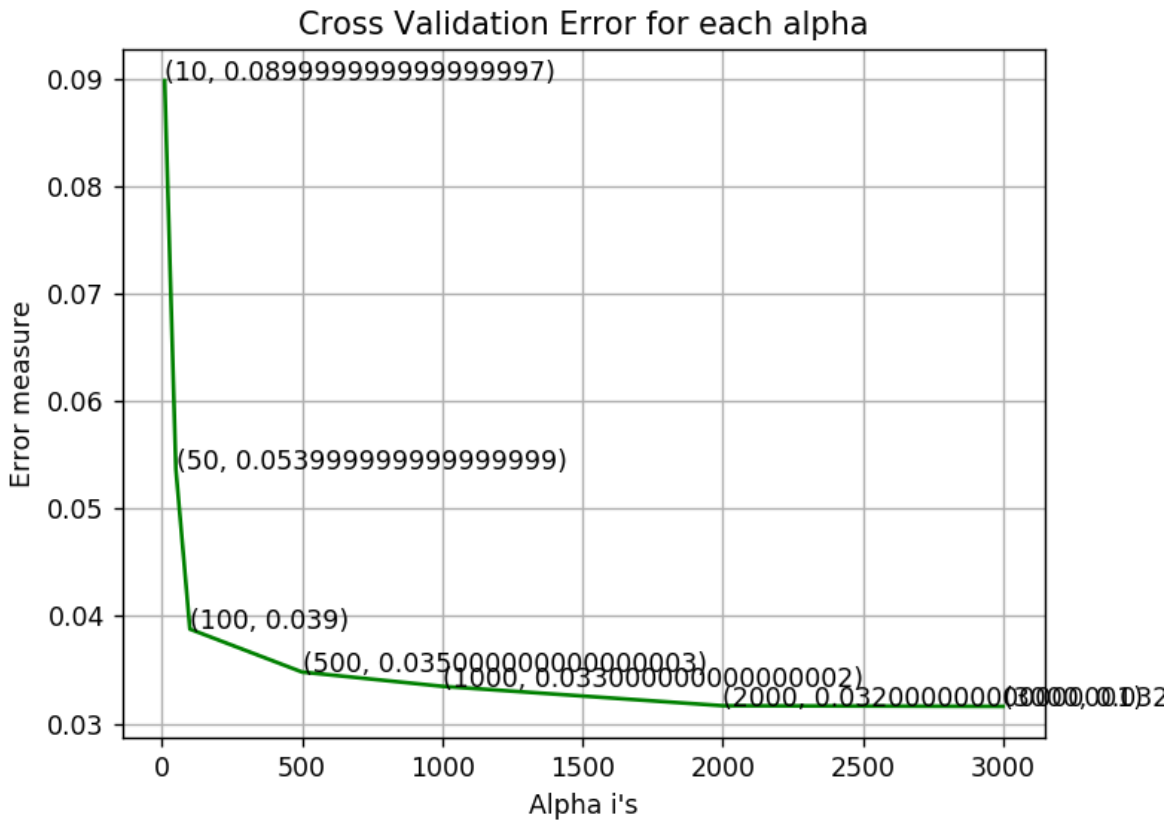
4.5.5. XgBoost Classifier on final features

In [ ]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  10 is 0.0898979446265
log_loss for c =  50 is 0.0536946658041
log_loss for c =  100 is 0.0387968186177
log_loss for c =  500 is 0.0347960327293
log_loss for c =  1000 is 0.0334668083237
log_loss for c =  2000 is 0.0316569078846
log_loss for c =  3000 is 0.0315972694477
```




```
For values of best alpha =  3000 The train log loss is: 0.0111918809342
```

```
For values of best alpha =  3000 The cross validation log loss is: 0.031597269
4477
For values of best alpha =  3000 The test log loss is: 0.0323978515915
```

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

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

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  4.5min remaining:  2.6m
in
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remaining:  1.8m
in
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remaining:   44.
5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsamp
le_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15,
0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'c
olsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

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

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 10, 'learning_rate': 0.15,
'colsample_bytree': 0.3}
```

In [ ]:
```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
```

```
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# ----------------------

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsamp
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
For values of best alpha =  3000 The train log loss is: 0.0121922832297
For values of best alpha =  3000 The cross validation log loss is: 0.034495548
7471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video) and include pixel intensity features to improve the logloss

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

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

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

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

Byte Features

In [1]:

```
byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,
,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,3
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
d,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,
,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,8
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
f,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,
,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,d
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
1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [3]:

```
len(byte_vocab.split(','))
```

Out[3]:
```
257
```

In [2]:
```
def byte_bigram():
    byte_bigram_vocab = []
    for i, v in enumerate(byte_vocab.split(',')):
        for j in range(0, len(byte_vocab.split(','))):
            byte_bigram_vocab.append(v + ' ' +byte_vocab.split(',')[j])
    return byte_bigram_vocab
```

In [3]:
```
byte_bigram_vocab = byte_bigram()
len(byte_bigram_vocab)
```

Out[3]:
```
66049
```

In [25]:
```
byte_bigram_vocab[:5]
```

Out[25]:
```
['00 00', '00 01', '00 02', '00 03', '00 04']
```

In [5]:
```
def byte_trigram():
    byte_trigram_vocab = []
    for i, v in enumerate(byte_vocab.split(',')):
        for j in range(0, len(byte_vocab.split(','))):
            for k in range(0, len(byte_vocab.split(','))):
                byte_trigram_vocab.append(v + ' ' +byte_vocab.split(',')[j]+'

    return byte_trigram_vocab
```

In [6]:
```
byte_trigram_vocab = byte_trigram()
len(byte_trigram_vocab)
```

Out[6]:
```
16974593
```

In [32]:
```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import scipy

vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_b
bytebigram_vect = scipy.sparse.csr_matrix((10868, 66049))

for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
    f = open('byteFiles/' + file)
    bytebigram_vect[i, :]+= scipy.sparse.csr_matrix(vector.fit_transform([f.r
    f.close()
```

```
5640it [5:30:08,  6.02s/it]
```

In [ ]:
```
import dill
dill.dump_session('/data/malware-classification/notebook.db')
```

In [7]:
```
import dill
dill.load_session("/data/malware-classification/notebook.db")
```

In [ ]:
```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import scipy
from multiprocessing import Pool, Lock, Process
from functools import partial
import dill
import time


def worker(fileTuple, vector):
    fileName = fileTuple[1]
    i = fileTuple[0]
    if i % 100 == 0:
        print("Processing file number ", i)
    f = open('byteFiles/' + fileName)
    processed = scipy.sparse.csr_matrix(vector.fit_transform([f.read().replac
    return (i, processed)

def sorting_func(tup):
    return tup[0]

vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_b
#bytebigram_vect = scipy.sparse.csr_matrix((10868, 66049))
bytebigram_vect = scipy.sparse.lil_matrix((10868, 66049))
pool = multiprocessing.Pool(processes = 25)
files_with_indexes = []
func = partial(worker, vector=vector)
for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
    files_with_indexes.append((i, file))

res = pool.map(func, files_with_indexes)
print ("Pool completed")
```

In [3]:
```
import pickle
res = sorted(res, key=sorting_func)
f = open("/data/malware-classification/res.pickle", "wb")
f.write(pickle.dumps(res))
f.close()
```

In [ ]:
```
start = time.time()

for res_tuple in res:
    index = res_tuple[0]
    processed = res_tuple[1]
    bytebigram_vect[index, :] = processed
    if index % 100 == 0:
        print ("Done with the index ", index)
bytebigram_vect = bytebigram_vect.tocsr()
end = time.time()
print("bytegram time: ", (end-start))

print ("bytegram_vec constructed")
```

In [36]:
```
import pickle
f = open("/data/malware-classification/bytebigram_vect_new.pickle", "wb")
pickle.dump(bytebigram_vect, f)
f.close()
```

In [8]:
```
f = open("/data/malware-classification/bytebigram_vect_new.pickle", "rb")
bytebigram_vect = pickle.load(f)
```

```
f.close()
```

In [9]:
```
bytebigram_vect.toarray()
```

Out[9]:
```
array([[1.79260e+04, 1.39000e+03, 8.96000e+02, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       [5.70100e+03, 4.90000e+01, 5.20000e+01, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       [1.77378e+05, 9.20000e+01, 4.00000e+01, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       ...,
       [4.25100e+03, 2.40000e+01, 1.90000e+01, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       [8.93100e+03, 2.05000e+02, 1.20000e+01, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       [2.73719e+05, 1.34400e+03, 1.02400e+03, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00]])
```

In [6]:
```
scipy.sparse.save_npz('bytebigram.npz', bytebigram_vect)
```

In [10]:
```
from sklearn.preprocessing import normalize
import scipy
byte_bigram_vect = normalize(scipy.sparse.load_npz('bytebigram.npz'), axis = (
```

N-Gram(2-Gram, 3-Gram, 4-Gram) Opcode Vectorization

In [11]:
```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub',
'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol',
'jz','rtn','lea','movzx']
```

In [12]:
```
def asmopcodebigram():
    asmopcodebigram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            asmopcodebigram.append(v + ' ' + opcodes[j])
    return asmopcodebigram
```

In [13]:
```
asmopcodebigram = asmopcodebigram()
```

In [14]:
```
def asmopcodetrigram():
    asmopcodetrigram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            for k in range(0, len(opcodes)):
                asmopcodetrigram.append(v + ' ' + opcodes[j] + ' ' + opcodes[
    return asmopcodetrigram
```

In [15]:
```
asmopcodetrigram = asmopcodetrigram()
```

In [16]:
```
def asmopcodetetragram():
    asmopcodetetragram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            for k in range(0, len(opcodes)):
```

```
                for l in range(0, len(opcodes)):
                    asmopcodetetragram.append(v + ' ' + opcodes[j] + ' ' + op
    return asmopcodetetragram
```

In [31]:

```
asmopcodetetragram = asmopcodetetragram()
```

In [6]:

```
from tqdm import tqdm
from multiprocessing import Pool, Lock, Process
from functools import partial
import time

def worker(fileTuple, vector):
    asmfile = fileTuple[1]
    index = fileTuple[0]
    if index % 100 == 0:
        print("Processing file number ", index)

    with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='repla
        raw_opcode = []
        for lines in fli:
            line = lines.rstrip().split()
            for li in line:
                if li in opcodes:
                    raw_opcode.append(li)
        opcode_str = " ".join(raw_opcode)
        processed = scipy.sparse.csr_matrix(vect.transform([opcode_str]))

    return (index, processed)

def sorting_func(tup):
    return tup[0]


vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asmopcodebigram)
opcodebivect = scipy.sparse.lil_matrix((10868, len(asmopcodebigram)))

pool = multiprocessing.Pool(processes = 25)
files_with_indexes = []
func = partial(worker, vector=vect)
for i, file in tqdm(enumerate(os.listdir('asmFiles'))):
    files_with_indexes.append((i, file))

res = pool.map(func, files_with_indexes)
print ("Pool completed")
```

```
10868it [00:00, 1117767.98it/s]
Processing file number  0
Processing file number  1200

Processing file number  2400
Processing file number  1000
Processing file number  1100
Processing file number  2300
Processing file number  2200
Processing file number  900
Processing file number  2100
Processing file number  2000
Processing file number  1900
Processing file number  700
Processing file number  800
Processing file number  1800
Processing file number  1700
```

```
Processing file number  600
Processing file number  1600
Processing file number  400
Processing file number  1500
Processing file number  500
Processing file number  1400
Processing file number  200
Processing file number  300
Processing file number  2600
Processing file number  1300
Processing file number  2700
Processing file number  2500
Processing file number  3600
Processing file number  3500
Processing file number  100
Processing file number  3400
Processing file number  4800
Processing file number  3300
Processing file number  3200
Processing file number  4600
Processing file number  3100
Processing file number  4500
Processing file number  2900
Processing file number  4300
Processing file number  2800
Processing file number  4700
Processing file number  4400
Processing file number  3000
Processing file number  4200
Processing file number  4100
Processing file number  3900
Processing file number  5300
Processing file number  5400
Processing file number  4000
Processing file number  5200
Processing file number  5800
Processing file number  3700
Processing file number  5100
Processing file number  3800
Processing file number  6000
Processing file number  5500
Processing file number  5900
Processing file number  5600
Processing file number  7200
Processing file number  4900
Processing file number  7000
Processing file number  5700
Processing file number  6900
Processing file number  7100
Processing file number  5000
Processing file number  6700
Processing file number  6800
Processing file number  6600
Processing file number  7800
Processing file number  8100
Processing file number  7900
Processing file number  8000
Processing file number  6500
Processing file number  6400
Processing file number  8400
Processing file number  6200
Processing file number  6100
Processing file number  8200
Processing file number  8300
Processing file number  6300
```

```
Processing file number  7700
Processing file number  7600
Processing file number  7300
Processing file number  7400
Processing file number  7500
Processing file number  9600
Processing file number  9400
Processing file number  8800
Processing file number  9100
Processing file number  9500
Processing file number  9000
Processing file number  9200
Processing file number  9300
Processing file number  10800
Processing file number  10700
Processing file number  10400
Processing file number  8500
Processing file number  8700
Processing file number  8900
Processing file number  10600
Processing file number  10500
Processing file number  8600
Processing file number  10100
Processing file number  10200
Processing file number  9800
Processing file number  10000
Processing file number  10300
Processing file number  9900
Processing file number  9700
Pool completed
```

In [7]:
```
res = sorted(res, key=sorting_func)
```

In [8]:
```
res
```

Out[8]:
```
[(0,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 192 stored elements in Compressed Sparse Row format>),
 (1,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 217 stored elements in Compressed Sparse Row format>),
 (2,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 367 stored elements in Compressed Sparse Row format>),
 (3,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 73 stored elements in Compressed Sparse Row format>),
 (4,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 100 stored elements in Compressed Sparse Row format>),
 (5,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 249 stored elements in Compressed Sparse Row format>),
 (6,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 69 stored elements in Compressed Sparse Row format>),
 (7,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 211 stored elements in Compressed Sparse Row format>),
 (8,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 319 stored elements in Compressed Sparse Row format>),
 (9,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 36 stored elements in Compressed Sparse Row format>),
(10,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(11,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 223 stored elements in Compressed Sparse Row format>),
(12,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 352 stored elements in Compressed Sparse Row format>),
(13,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 292 stored elements in Compressed Sparse Row format>),
(14,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(15,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 266 stored elements in Compressed Sparse Row format>),
(16,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(17,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(18,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 267 stored elements in Compressed Sparse Row format>),
(19,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(20,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 277 stored elements in Compressed Sparse Row format>),
(21,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 205 stored elements in Compressed Sparse Row format>),
(22,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 35 stored elements in Compressed Sparse Row format>),
(23,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 115 stored elements in Compressed Sparse Row format>),
(24,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 337 stored elements in Compressed Sparse Row format>),
(25,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 210 stored elements in Compressed Sparse Row format>),
(26,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 149 stored elements in Compressed Sparse Row format>),
(27,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 252 stored elements in Compressed Sparse Row format>),
(28,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(29,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 75 stored elements in Compressed Sparse Row format>),
(30,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 233 stored elements in Compressed Sparse Row format>),
```

```
(31,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(32,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(33,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(34,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 222 stored elements in Compressed Sparse Row format>),
(35,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 230 stored elements in Compressed Sparse Row format>),
(36,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 277 stored elements in Compressed Sparse Row format>),
(37,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 114 stored elements in Compressed Sparse Row format>),
(38,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 200 stored elements in Compressed Sparse Row format>),
(39,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(40,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 210 stored elements in Compressed Sparse Row format>),
(41,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(42,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(43,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 185 stored elements in Compressed Sparse Row format>),
(44,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 100 stored elements in Compressed Sparse Row format>),
(45,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(46,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 209 stored elements in Compressed Sparse Row format>),
(47,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 337 stored elements in Compressed Sparse Row format>),
(48,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 353 stored elements in Compressed Sparse Row format>),
(49,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 170 stored elements in Compressed Sparse Row format>),
(50,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(51,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 110 stored elements in Compressed Sparse Row format>),
(52,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
        with 203 stored elements in Compressed Sparse Row format>),
 (53,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 212 stored elements in Compressed Sparse Row format>),
 (54,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 229 stored elements in Compressed Sparse Row format>),
 (55,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 215 stored elements in Compressed Sparse Row format>),
 (56,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 184 stored elements in Compressed Sparse Row format>),
 (57,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 71 stored elements in Compressed Sparse Row format>),
 (58,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 245 stored elements in Compressed Sparse Row format>),
 (59,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 98 stored elements in Compressed Sparse Row format>),
 (60,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 60 stored elements in Compressed Sparse Row format>),
 (61,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 256 stored elements in Compressed Sparse Row format>),
 (62,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 60 stored elements in Compressed Sparse Row format>),
 (63,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 70 stored elements in Compressed Sparse Row format>),
 (64,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 108 stored elements in Compressed Sparse Row format>),
 (65,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 220 stored elements in Compressed Sparse Row format>),
 (66,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 64 stored elements in Compressed Sparse Row format>),
 (67,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 111 stored elements in Compressed Sparse Row format>),
 (68,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 231 stored elements in Compressed Sparse Row format>),
 (69,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 311 stored elements in Compressed Sparse Row format>),
 (70,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 313 stored elements in Compressed Sparse Row format>),
 (71,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 68 stored elements in Compressed Sparse Row format>),
 (72,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 70 stored elements in Compressed Sparse Row format>),
 (73,
  <1x676 sparse matrix of type '<class 'numpy.int64'>'
        with 341 stored elements in Compressed Sparse Row format>),
 (74,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 34 stored elements in Compressed Sparse Row format>),
(75,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(76,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(77,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 170 stored elements in Compressed Sparse Row format>),
(78,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 219 stored elements in Compressed Sparse Row format>),
(79,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 210 stored elements in Compressed Sparse Row format>),
(80,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 211 stored elements in Compressed Sparse Row format>),
(81,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(82,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(83,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 378 stored elements in Compressed Sparse Row format>),
(84,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 350 stored elements in Compressed Sparse Row format>),
(85,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(86,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 225 stored elements in Compressed Sparse Row format>),
(87,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 185 stored elements in Compressed Sparse Row format>),
(88,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 142 stored elements in Compressed Sparse Row format>),
(89,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 220 stored elements in Compressed Sparse Row format>),
(90,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 59 stored elements in Compressed Sparse Row format>),
(91,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(92,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 261 stored elements in Compressed Sparse Row format>),
(93,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 228 stored elements in Compressed Sparse Row format>),
(94,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 149 stored elements in Compressed Sparse Row format>),
(95,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 348 stored elements in Compressed Sparse Row format>),
```

```
(96,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(97,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(98,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 276 stored elements in Compressed Sparse Row format>),
(99,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(100,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(101,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 132 stored elements in Compressed Sparse Row format>),
(102,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 198 stored elements in Compressed Sparse Row format>),
(103,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 239 stored elements in Compressed Sparse Row format>),
(104,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 126 stored elements in Compressed Sparse Row format>),
(105,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 211 stored elements in Compressed Sparse Row format>),
(106,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 100 stored elements in Compressed Sparse Row format>),
(107,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 209 stored elements in Compressed Sparse Row format>),
(108,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 217 stored elements in Compressed Sparse Row format>),
(109,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(110,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 155 stored elements in Compressed Sparse Row format>),
(111,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 35 stored elements in Compressed Sparse Row format>),
(112,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 376 stored elements in Compressed Sparse Row format>),
(113,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 188 stored elements in Compressed Sparse Row format>),
(114,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 205 stored elements in Compressed Sparse Row format>),
(115,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 261 stored elements in Compressed Sparse Row format>),
(116,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 240 stored elements in Compressed Sparse Row format>),
(117,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 74 stored elements in Compressed Sparse Row format>),
(118,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 187 stored elements in Compressed Sparse Row format>),
(119,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 196 stored elements in Compressed Sparse Row format>),
(120,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(121,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 68 stored elements in Compressed Sparse Row format>),
(122,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 133 stored elements in Compressed Sparse Row format>),
(123,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 219 stored elements in Compressed Sparse Row format>),
(124,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(125,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 266 stored elements in Compressed Sparse Row format>),
(126,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 34 stored elements in Compressed Sparse Row format>),
(127,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 348 stored elements in Compressed Sparse Row format>),
(128,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 191 stored elements in Compressed Sparse Row format>),
(129,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(130,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 288 stored elements in Compressed Sparse Row format>),
(131,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 247 stored elements in Compressed Sparse Row format>),
(132,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 387 stored elements in Compressed Sparse Row format>),
(133,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 63 stored elements in Compressed Sparse Row format>),
(134,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 64 stored elements in Compressed Sparse Row format>),
(135,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 355 stored elements in Compressed Sparse Row format>),
(136,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(137,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 36 stored elements in Compressed Sparse Row format>),
(138,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 154 stored elements in Compressed Sparse Row format>),
(139,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 260 stored elements in Compressed Sparse Row format>),
(140,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 178 stored elements in Compressed Sparse Row format>),
(141,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 238 stored elements in Compressed Sparse Row format>),
(142,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 225 stored elements in Compressed Sparse Row format>),
(143,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(144,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 234 stored elements in Compressed Sparse Row format>),
(145,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(146,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 227 stored elements in Compressed Sparse Row format>),
(147,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 128 stored elements in Compressed Sparse Row format>),
(148,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 221 stored elements in Compressed Sparse Row format>),
(149,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 150 stored elements in Compressed Sparse Row format>),
(150,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 75 stored elements in Compressed Sparse Row format>),
(151,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 159 stored elements in Compressed Sparse Row format>),
(152,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(153,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(154,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(155,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 154 stored elements in Compressed Sparse Row format>),
(156,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 311 stored elements in Compressed Sparse Row format>),
(157,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(158,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 177 stored elements in Compressed Sparse Row format>),
(159,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 242 stored elements in Compressed Sparse Row format>),
(160,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 276 stored elements in Compressed Sparse Row format>),
```

```
(161,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 338 stored elements in Compressed Sparse Row format>),
(162,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 305 stored elements in Compressed Sparse Row format>),
(163,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 200 stored elements in Compressed Sparse Row format>),
(164,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 266 stored elements in Compressed Sparse Row format>),
(165,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 36 stored elements in Compressed Sparse Row format>),
(166,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(167,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 129 stored elements in Compressed Sparse Row format>),
(168,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 97 stored elements in Compressed Sparse Row format>),
(169,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 116 stored elements in Compressed Sparse Row format>),
(170,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 334 stored elements in Compressed Sparse Row format>),
(171,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 220 stored elements in Compressed Sparse Row format>),
(172,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(173,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 36 stored elements in Compressed Sparse Row format>),
(174,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 311 stored elements in Compressed Sparse Row format>),
(175,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 285 stored elements in Compressed Sparse Row format>),
(176,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 248 stored elements in Compressed Sparse Row format>),
(177,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(178,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 366 stored elements in Compressed Sparse Row format>),
(179,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(180,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(181,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 257 stored elements in Compressed Sparse Row format>),
(182,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 280 stored elements in Compressed Sparse Row format>),
(183,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 174 stored elements in Compressed Sparse Row format>),
(184,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(185,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 72 stored elements in Compressed Sparse Row format>),
(186,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 350 stored elements in Compressed Sparse Row format>),
(187,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 136 stored elements in Compressed Sparse Row format>),
(188,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 209 stored elements in Compressed Sparse Row format>),
(189,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 255 stored elements in Compressed Sparse Row format>),
(190,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 300 stored elements in Compressed Sparse Row format>),
(191,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 33 stored elements in Compressed Sparse Row format>),
(192,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(193,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 155 stored elements in Compressed Sparse Row format>),
(194,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 251 stored elements in Compressed Sparse Row format>),
(195,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 165 stored elements in Compressed Sparse Row format>),
(196,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 67 stored elements in Compressed Sparse Row format>),
(197,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 245 stored elements in Compressed Sparse Row format>),
(198,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 354 stored elements in Compressed Sparse Row format>),
(199,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 223 stored elements in Compressed Sparse Row format>),
(200,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(201,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(202,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 108 stored elements in Compressed Sparse Row format>),
(203,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 253 stored elements in Compressed Sparse Row format>),
(204,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 188 stored elements in Compressed Sparse Row format>),
(205,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 181 stored elements in Compressed Sparse Row format>),
(206,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 213 stored elements in Compressed Sparse Row format>),
(207,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 212 stored elements in Compressed Sparse Row format>),
(208,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 279 stored elements in Compressed Sparse Row format>),
(209,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 218 stored elements in Compressed Sparse Row format>),
(210,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 269 stored elements in Compressed Sparse Row format>),
(211,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(212,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(213,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 304 stored elements in Compressed Sparse Row format>),
(214,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 276 stored elements in Compressed Sparse Row format>),
(215,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(216,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 275 stored elements in Compressed Sparse Row format>),
(217,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(218,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 224 stored elements in Compressed Sparse Row format>),
(219,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 137 stored elements in Compressed Sparse Row format>),
(220,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 324 stored elements in Compressed Sparse Row format>),
(221,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 207 stored elements in Compressed Sparse Row format>),
(222,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 358 stored elements in Compressed Sparse Row format>),
(223,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(224,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 418 stored elements in Compressed Sparse Row format>),
(225,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 182 stored elements in Compressed Sparse Row format>),
```

```
(226,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 200 stored elements in Compressed Sparse Row format>),
(227,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(228,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(229,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 287 stored elements in Compressed Sparse Row format>),
(230,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 214 stored elements in Compressed Sparse Row format>),
(231,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 75 stored elements in Compressed Sparse Row format>),
(232,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(233,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 286 stored elements in Compressed Sparse Row format>),
(234,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 346 stored elements in Compressed Sparse Row format>),
(235,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 137 stored elements in Compressed Sparse Row format>),
(236,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 255 stored elements in Compressed Sparse Row format>),
(237,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 207 stored elements in Compressed Sparse Row format>),
(238,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(239,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(240,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 88 stored elements in Compressed Sparse Row format>),
(241,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 265 stored elements in Compressed Sparse Row format>),
(242,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 166 stored elements in Compressed Sparse Row format>),
(243,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 284 stored elements in Compressed Sparse Row format>),
(244,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(245,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 141 stored elements in Compressed Sparse Row format>),
(246,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 285 stored elements in Compressed Sparse Row format>),
(247,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 211 stored elements in Compressed Sparse Row format>),
(248,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(249,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 342 stored elements in Compressed Sparse Row format>),
(250,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 222 stored elements in Compressed Sparse Row format>),
(251,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(252,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 128 stored elements in Compressed Sparse Row format>),
(253,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 73 stored elements in Compressed Sparse Row format>),
(254,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 264 stored elements in Compressed Sparse Row format>),
(255,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 119 stored elements in Compressed Sparse Row format>),
(256,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 219 stored elements in Compressed Sparse Row format>),
(257,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 38 stored elements in Compressed Sparse Row format>),
(258,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 180 stored elements in Compressed Sparse Row format>),
(259,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 129 stored elements in Compressed Sparse Row format>),
(260,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 72 stored elements in Compressed Sparse Row format>),
(261,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 249 stored elements in Compressed Sparse Row format>),
(262,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 159 stored elements in Compressed Sparse Row format>),
(263,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(264,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 229 stored elements in Compressed Sparse Row format>),
(265,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 74 stored elements in Compressed Sparse Row format>),
(266,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 214 stored elements in Compressed Sparse Row format>),
(267,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 183 stored elements in Compressed Sparse Row format>),
(268,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 70 stored elements in Compressed Sparse Row format>),
(269,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(270,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 221 stored elements in Compressed Sparse Row format>),
(271,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 199 stored elements in Compressed Sparse Row format>),
(272,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(273,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 273 stored elements in Compressed Sparse Row format>),
(274,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(275,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 213 stored elements in Compressed Sparse Row format>),
(276,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 260 stored elements in Compressed Sparse Row format>),
(277,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(278,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 38 stored elements in Compressed Sparse Row format>),
(279,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 129 stored elements in Compressed Sparse Row format>),
(280,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(281,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 297 stored elements in Compressed Sparse Row format>),
(282,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 116 stored elements in Compressed Sparse Row format>),
(283,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 354 stored elements in Compressed Sparse Row format>),
(284,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 307 stored elements in Compressed Sparse Row format>),
(285,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(286,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 144 stored elements in Compressed Sparse Row format>),
(287,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 338 stored elements in Compressed Sparse Row format>),
(288,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 294 stored elements in Compressed Sparse Row format>),
(289,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 32 stored elements in Compressed Sparse Row format>),
(290,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
```

```
(291,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 58 stored elements in Compressed Sparse Row format>),
(292,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(293,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(294,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 329 stored elements in Compressed Sparse Row format>),
(295,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(296,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(297,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 264 stored elements in Compressed Sparse Row format>),
(298,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(299,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 357 stored elements in Compressed Sparse Row format>),
(300,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 239 stored elements in Compressed Sparse Row format>),
(301,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 135 stored elements in Compressed Sparse Row format>),
(302,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 240 stored elements in Compressed Sparse Row format>),
(303,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(304,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 185 stored elements in Compressed Sparse Row format>),
(305,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 37 stored elements in Compressed Sparse Row format>),
(306,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 310 stored elements in Compressed Sparse Row format>),
(307,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 323 stored elements in Compressed Sparse Row format>),
(308,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 322 stored elements in Compressed Sparse Row format>),
(309,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 100 stored elements in Compressed Sparse Row format>),
(310,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 107 stored elements in Compressed Sparse Row format>),
(311,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 367 stored elements in Compressed Sparse Row format>),
(312,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 249 stored elements in Compressed Sparse Row format>),
(313,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 118 stored elements in Compressed Sparse Row format>),
(314,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 256 stored elements in Compressed Sparse Row format>),
(315,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 163 stored elements in Compressed Sparse Row format>),
(316,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 205 stored elements in Compressed Sparse Row format>),
(317,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(318,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 251 stored elements in Compressed Sparse Row format>),
(319,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 142 stored elements in Compressed Sparse Row format>),
(320,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 193 stored elements in Compressed Sparse Row format>),
(321,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 198 stored elements in Compressed Sparse Row format>),
(322,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 70 stored elements in Compressed Sparse Row format>),
(323,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 120 stored elements in Compressed Sparse Row format>),
(324,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 122 stored elements in Compressed Sparse Row format>),
(325,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 265 stored elements in Compressed Sparse Row format>),
(326,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 35 stored elements in Compressed Sparse Row format>),
(327,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 215 stored elements in Compressed Sparse Row format>),
(328,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 142 stored elements in Compressed Sparse Row format>),
(329,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 136 stored elements in Compressed Sparse Row format>),
(330,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 186 stored elements in Compressed Sparse Row format>),
(331,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 137 stored elements in Compressed Sparse Row format>),
(332,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(333,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 37 stored elements in Compressed Sparse Row format>),
(334,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 256 stored elements in Compressed Sparse Row format>),
(335,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(336,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 335 stored elements in Compressed Sparse Row format>),
(337,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 242 stored elements in Compressed Sparse Row format>),
(338,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(339,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(340,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 162 stored elements in Compressed Sparse Row format>),
(341,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 257 stored elements in Compressed Sparse Row format>),
(342,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 213 stored elements in Compressed Sparse Row format>),
(343,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(344,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 114 stored elements in Compressed Sparse Row format>),
(345,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 264 stored elements in Compressed Sparse Row format>),
(346,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 236 stored elements in Compressed Sparse Row format>),
(347,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 281 stored elements in Compressed Sparse Row format>),
(348,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(349,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(350,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 370 stored elements in Compressed Sparse Row format>),
(351,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 60 stored elements in Compressed Sparse Row format>),
(352,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 241 stored elements in Compressed Sparse Row format>),
(353,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 259 stored elements in Compressed Sparse Row format>),
(354,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 252 stored elements in Compressed Sparse Row format>),
(355,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
```

```
(356,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 354 stored elements in Compressed Sparse Row format>),
(357,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 387 stored elements in Compressed Sparse Row format>),
(358,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 346 stored elements in Compressed Sparse Row format>),
(359,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 290 stored elements in Compressed Sparse Row format>),
(360,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(361,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 169 stored elements in Compressed Sparse Row format>),
(362,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(363,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 144 stored elements in Compressed Sparse Row format>),
(364,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 246 stored elements in Compressed Sparse Row format>),
(365,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(366,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 36 stored elements in Compressed Sparse Row format>),
(367,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(368,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 347 stored elements in Compressed Sparse Row format>),
(369,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 215 stored elements in Compressed Sparse Row format>),
(370,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(371,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 203 stored elements in Compressed Sparse Row format>),
(372,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(373,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 342 stored elements in Compressed Sparse Row format>),
(374,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 204 stored elements in Compressed Sparse Row format>),
(375,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(376,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(377,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 63 stored elements in Compressed Sparse Row format>),
(378,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 68 stored elements in Compressed Sparse Row format>),
(379,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 155 stored elements in Compressed Sparse Row format>),
(380,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(381,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 192 stored elements in Compressed Sparse Row format>),
(382,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 183 stored elements in Compressed Sparse Row format>),
(383,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(384,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 68 stored elements in Compressed Sparse Row format>),
(385,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 236 stored elements in Compressed Sparse Row format>),
(386,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 63 stored elements in Compressed Sparse Row format>),
(387,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 227 stored elements in Compressed Sparse Row format>),
(388,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 48 stored elements in Compressed Sparse Row format>),
(389,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 303 stored elements in Compressed Sparse Row format>),
(390,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 230 stored elements in Compressed Sparse Row format>),
(391,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 337 stored elements in Compressed Sparse Row format>),
(392,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 256 stored elements in Compressed Sparse Row format>),
(393,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 139 stored elements in Compressed Sparse Row format>),
(394,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 226 stored elements in Compressed Sparse Row format>),
(395,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 255 stored elements in Compressed Sparse Row format>),
(396,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(397,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 236 stored elements in Compressed Sparse Row format>),
(398,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 41 stored elements in Compressed Sparse Row format>),
(399,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 173 stored elements in Compressed Sparse Row format>),
(400,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 297 stored elements in Compressed Sparse Row format>),
(401,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(402,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 58 stored elements in Compressed Sparse Row format>),
(403,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 212 stored elements in Compressed Sparse Row format>),
(404,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(405,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(406,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(407,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(408,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(409,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(410,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(411,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 240 stored elements in Compressed Sparse Row format>),
(412,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 210 stored elements in Compressed Sparse Row format>),
(413,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 181 stored elements in Compressed Sparse Row format>),
(414,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 255 stored elements in Compressed Sparse Row format>),
(415,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(416,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 163 stored elements in Compressed Sparse Row format>),
(417,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(418,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 167 stored elements in Compressed Sparse Row format>),
(419,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 348 stored elements in Compressed Sparse Row format>),
(420,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 142 stored elements in Compressed Sparse Row format>),
```

```
(421,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(422,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(423,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(424,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 208 stored elements in Compressed Sparse Row format>),
(425,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(426,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 144 stored elements in Compressed Sparse Row format>),
(427,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 120 stored elements in Compressed Sparse Row format>),
(428,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 186 stored elements in Compressed Sparse Row format>),
(429,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(430,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(431,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 98 stored elements in Compressed Sparse Row format>),
(432,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 240 stored elements in Compressed Sparse Row format>),
(433,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 248 stored elements in Compressed Sparse Row format>),
(434,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 106 stored elements in Compressed Sparse Row format>),
(435,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 303 stored elements in Compressed Sparse Row format>),
(436,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(437,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(438,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 252 stored elements in Compressed Sparse Row format>),
(439,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 270 stored elements in Compressed Sparse Row format>),
(440,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 203 stored elements in Compressed Sparse Row format>),
(441,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 200 stored elements in Compressed Sparse Row format>),
(442,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 67 stored elements in Compressed Sparse Row format>),
(443,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 139 stored elements in Compressed Sparse Row format>),
(444,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 206 stored elements in Compressed Sparse Row format>),
(445,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 205 stored elements in Compressed Sparse Row format>),
(446,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 117 stored elements in Compressed Sparse Row format>),
(447,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 311 stored elements in Compressed Sparse Row format>),
(448,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 228 stored elements in Compressed Sparse Row format>),
(449,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 261 stored elements in Compressed Sparse Row format>),
(450,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(451,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 326 stored elements in Compressed Sparse Row format>),
(452,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 259 stored elements in Compressed Sparse Row format>),
(453,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(454,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 232 stored elements in Compressed Sparse Row format>),
(455,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 217 stored elements in Compressed Sparse Row format>),
(456,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 14 stored elements in Compressed Sparse Row format>),
(457,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 198 stored elements in Compressed Sparse Row format>),
(458,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 120 stored elements in Compressed Sparse Row format>),
(459,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 243 stored elements in Compressed Sparse Row format>),
(460,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 262 stored elements in Compressed Sparse Row format>),
(461,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 238 stored elements in Compressed Sparse Row format>),
(462,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 190 stored elements in Compressed Sparse Row format>),
(463,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(464,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(465,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 203 stored elements in Compressed Sparse Row format>),
(466,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 140 stored elements in Compressed Sparse Row format>),
(467,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(468,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 371 stored elements in Compressed Sparse Row format>),
(469,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(470,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 352 stored elements in Compressed Sparse Row format>),
(471,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 267 stored elements in Compressed Sparse Row format>),
(472,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 365 stored elements in Compressed Sparse Row format>),
(473,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 128 stored elements in Compressed Sparse Row format>),
(474,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 162 stored elements in Compressed Sparse Row format>),
(475,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(476,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 147 stored elements in Compressed Sparse Row format>),
(477,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 30 stored elements in Compressed Sparse Row format>),
(478,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 208 stored elements in Compressed Sparse Row format>),
(479,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 298 stored elements in Compressed Sparse Row format>),
(480,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 211 stored elements in Compressed Sparse Row format>),
(481,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 112 stored elements in Compressed Sparse Row format>),
(482,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 89 stored elements in Compressed Sparse Row format>),
(483,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(484,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(485,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
```

```
(486,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 176 stored elements in Compressed Sparse Row format>),
(487,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 214 stored elements in Compressed Sparse Row format>),
(488,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 100 stored elements in Compressed Sparse Row format>),
(489,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 323 stored elements in Compressed Sparse Row format>),
(490,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(491,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(492,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 190 stored elements in Compressed Sparse Row format>),
(493,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 217 stored elements in Compressed Sparse Row format>),
(494,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 118 stored elements in Compressed Sparse Row format>),
(495,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 203 stored elements in Compressed Sparse Row format>),
(496,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(497,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 306 stored elements in Compressed Sparse Row format>),
(498,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(499,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 194 stored elements in Compressed Sparse Row format>),
(500,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(501,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 226 stored elements in Compressed Sparse Row format>),
(502,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 340 stored elements in Compressed Sparse Row format>),
(503,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(504,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(505,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(506,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(507,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 87 stored elements in Compressed Sparse Row format>),
(508,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 175 stored elements in Compressed Sparse Row format>),
(509,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 74 stored elements in Compressed Sparse Row format>),
(510,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 143 stored elements in Compressed Sparse Row format>),
(511,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 72 stored elements in Compressed Sparse Row format>),
(512,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 234 stored elements in Compressed Sparse Row format>),
(513,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 209 stored elements in Compressed Sparse Row format>),
(514,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 295 stored elements in Compressed Sparse Row format>),
(515,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 63 stored elements in Compressed Sparse Row format>),
(516,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(517,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 188 stored elements in Compressed Sparse Row format>),
(518,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 331 stored elements in Compressed Sparse Row format>),
(519,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 121 stored elements in Compressed Sparse Row format>),
(520,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 64 stored elements in Compressed Sparse Row format>),
(521,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 152 stored elements in Compressed Sparse Row format>),
(522,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 133 stored elements in Compressed Sparse Row format>),
(523,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 183 stored elements in Compressed Sparse Row format>),
(524,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 265 stored elements in Compressed Sparse Row format>),
(525,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 0 stored elements in Compressed Sparse Row format>),
(526,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(527,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 286 stored elements in Compressed Sparse Row format>),
(528,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 227 stored elements in Compressed Sparse Row format>),
(529,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(530,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 99 stored elements in Compressed Sparse Row format>),
(531,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(532,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 162 stored elements in Compressed Sparse Row format>),
(533,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 261 stored elements in Compressed Sparse Row format>),
(534,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 238 stored elements in Compressed Sparse Row format>),
(535,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 41 stored elements in Compressed Sparse Row format>),
(536,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(537,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 215 stored elements in Compressed Sparse Row format>),
(538,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 203 stored elements in Compressed Sparse Row format>),
(539,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 36 stored elements in Compressed Sparse Row format>),
(540,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 198 stored elements in Compressed Sparse Row format>),
(541,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 75 stored elements in Compressed Sparse Row format>),
(542,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 223 stored elements in Compressed Sparse Row format>),
(543,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 125 stored elements in Compressed Sparse Row format>),
(544,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 137 stored elements in Compressed Sparse Row format>),
(545,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(546,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 169 stored elements in Compressed Sparse Row format>),
(547,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(548,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(549,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 224 stored elements in Compressed Sparse Row format>),
(550,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 121 stored elements in Compressed Sparse Row format>),
```

```
(551,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 161 stored elements in Compressed Sparse Row format>),
(552,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 207 stored elements in Compressed Sparse Row format>),
(553,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 152 stored elements in Compressed Sparse Row format>),
(554,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 76 stored elements in Compressed Sparse Row format>),
(555,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(556,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 333 stored elements in Compressed Sparse Row format>),
(557,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 225 stored elements in Compressed Sparse Row format>),
(558,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 254 stored elements in Compressed Sparse Row format>),
(559,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(560,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(561,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 297 stored elements in Compressed Sparse Row format>),
(562,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 132 stored elements in Compressed Sparse Row format>),
(563,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(564,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 191 stored elements in Compressed Sparse Row format>),
(565,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 201 stored elements in Compressed Sparse Row format>),
(566,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 237 stored elements in Compressed Sparse Row format>),
(567,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(568,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(569,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 92 stored elements in Compressed Sparse Row format>),
(570,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(571,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(572,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 268 stored elements in Compressed Sparse Row format>),
(573,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 204 stored elements in Compressed Sparse Row format>),
(574,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 75 stored elements in Compressed Sparse Row format>),
(575,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 285 stored elements in Compressed Sparse Row format>),
(576,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 123 stored elements in Compressed Sparse Row format>),
(577,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 177 stored elements in Compressed Sparse Row format>),
(578,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 66 stored elements in Compressed Sparse Row format>),
(579,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 390 stored elements in Compressed Sparse Row format>),
(580,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 133 stored elements in Compressed Sparse Row format>),
(581,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 246 stored elements in Compressed Sparse Row format>),
(582,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 355 stored elements in Compressed Sparse Row format>),
(583,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 136 stored elements in Compressed Sparse Row format>),
(584,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 275 stored elements in Compressed Sparse Row format>),
(585,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 356 stored elements in Compressed Sparse Row format>),
(586,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 70 stored elements in Compressed Sparse Row format>),
(587,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 67 stored elements in Compressed Sparse Row format>),
(588,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 39 stored elements in Compressed Sparse Row format>),
(589,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 186 stored elements in Compressed Sparse Row format>),
(590,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 67 stored elements in Compressed Sparse Row format>),
(591,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 57 stored elements in Compressed Sparse Row format>),
(592,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 313 stored elements in Compressed Sparse Row format>),
(593,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 218 stored elements in Compressed Sparse Row format>),
(594,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 256 stored elements in Compressed Sparse Row format>),
(595,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 85 stored elements in Compressed Sparse Row format>),
(596,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 201 stored elements in Compressed Sparse Row format>),
(597,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 211 stored elements in Compressed Sparse Row format>),
(598,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 266 stored elements in Compressed Sparse Row format>),
(599,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(600,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 258 stored elements in Compressed Sparse Row format>),
(601,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 328 stored elements in Compressed Sparse Row format>),
(602,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(603,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 111 stored elements in Compressed Sparse Row format>),
(604,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 183 stored elements in Compressed Sparse Row format>),
(605,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 396 stored elements in Compressed Sparse Row format>),
(606,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(607,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 202 stored elements in Compressed Sparse Row format>),
(608,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 183 stored elements in Compressed Sparse Row format>),
(609,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 153 stored elements in Compressed Sparse Row format>),
(610,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(611,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(612,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 344 stored elements in Compressed Sparse Row format>),
(613,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 309 stored elements in Compressed Sparse Row format>),
(614,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 38 stored elements in Compressed Sparse Row format>),
(615,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
```

```
(616,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(617,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(618,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 222 stored elements in Compressed Sparse Row format>),
(619,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 251 stored elements in Compressed Sparse Row format>),
(620,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 259 stored elements in Compressed Sparse Row format>),
(621,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 221 stored elements in Compressed Sparse Row format>),
(622,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 288 stored elements in Compressed Sparse Row format>),
(623,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 352 stored elements in Compressed Sparse Row format>),
(624,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 253 stored elements in Compressed Sparse Row format>),
(625,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(626,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 152 stored elements in Compressed Sparse Row format>),
(627,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 279 stored elements in Compressed Sparse Row format>),
(628,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 204 stored elements in Compressed Sparse Row format>),
(629,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 132 stored elements in Compressed Sparse Row format>),
(630,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(631,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 167 stored elements in Compressed Sparse Row format>),
(632,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 214 stored elements in Compressed Sparse Row format>),
(633,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 201 stored elements in Compressed Sparse Row format>),
(634,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 210 stored elements in Compressed Sparse Row format>),
(635,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 284 stored elements in Compressed Sparse Row format>),
(636,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 58 stored elements in Compressed Sparse Row format>),
(637,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 338 stored elements in Compressed Sparse Row format>),
(638,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 154 stored elements in Compressed Sparse Row format>),
(639,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 225 stored elements in Compressed Sparse Row format>),
(640,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 125 stored elements in Compressed Sparse Row format>),
(641,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 371 stored elements in Compressed Sparse Row format>),
(642,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 123 stored elements in Compressed Sparse Row format>),
(643,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 30 stored elements in Compressed Sparse Row format>),
(644,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 344 stored elements in Compressed Sparse Row format>),
(645,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 30 stored elements in Compressed Sparse Row format>),
(646,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 295 stored elements in Compressed Sparse Row format>),
(647,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(648,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(649,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 100 stored elements in Compressed Sparse Row format>),
(650,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(651,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 215 stored elements in Compressed Sparse Row format>),
(652,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 67 stored elements in Compressed Sparse Row format>),
(653,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 147 stored elements in Compressed Sparse Row format>),
(654,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(655,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 261 stored elements in Compressed Sparse Row format>),
(656,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(657,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 139 stored elements in Compressed Sparse Row format>),
(658,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 207 stored elements in Compressed Sparse Row format>),
(659,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 202 stored elements in Compressed Sparse Row format>),
(660,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 160 stored elements in Compressed Sparse Row format>),
(661,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(662,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 156 stored elements in Compressed Sparse Row format>),
(663,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 118 stored elements in Compressed Sparse Row format>),
(664,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(665,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 125 stored elements in Compressed Sparse Row format>),
(666,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 279 stored elements in Compressed Sparse Row format>),
(667,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(668,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 182 stored elements in Compressed Sparse Row format>),
(669,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(670,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 390 stored elements in Compressed Sparse Row format>),
(671,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 263 stored elements in Compressed Sparse Row format>),
(672,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 118 stored elements in Compressed Sparse Row format>),
(673,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 269 stored elements in Compressed Sparse Row format>),
(674,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 197 stored elements in Compressed Sparse Row format>),
(675,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 344 stored elements in Compressed Sparse Row format>),
(676,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(677,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(678,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(679,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 127 stored elements in Compressed Sparse Row format>),
(680,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
```

```
(681,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 227 stored elements in Compressed Sparse Row format>),
(682,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 203 stored elements in Compressed Sparse Row format>),
(683,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 207 stored elements in Compressed Sparse Row format>),
(684,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(685,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 118 stored elements in Compressed Sparse Row format>),
(686,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 158 stored elements in Compressed Sparse Row format>),
(687,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 134 stored elements in Compressed Sparse Row format>),
(688,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 165 stored elements in Compressed Sparse Row format>),
(689,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 114 stored elements in Compressed Sparse Row format>),
(690,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(691,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 106 stored elements in Compressed Sparse Row format>),
(692,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 208 stored elements in Compressed Sparse Row format>),
(693,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 241 stored elements in Compressed Sparse Row format>),
(694,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 37 stored elements in Compressed Sparse Row format>),
(695,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 313 stored elements in Compressed Sparse Row format>),
(696,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(697,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 130 stored elements in Compressed Sparse Row format>),
(698,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 384 stored elements in Compressed Sparse Row format>),
(699,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 218 stored elements in Compressed Sparse Row format>),
(700,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 184 stored elements in Compressed Sparse Row format>),
(701,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 300 stored elements in Compressed Sparse Row format>),
(702,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 108 stored elements in Compressed Sparse Row format>),
(703,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 159 stored elements in Compressed Sparse Row format>),
(704,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 148 stored elements in Compressed Sparse Row format>),
(705,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 64 stored elements in Compressed Sparse Row format>),
(706,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 357 stored elements in Compressed Sparse Row format>),
(707,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 217 stored elements in Compressed Sparse Row format>),
(708,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 134 stored elements in Compressed Sparse Row format>),
(709,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 192 stored elements in Compressed Sparse Row format>),
(710,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 227 stored elements in Compressed Sparse Row format>),
(711,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 209 stored elements in Compressed Sparse Row format>),
(712,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 268 stored elements in Compressed Sparse Row format>),
(713,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 66 stored elements in Compressed Sparse Row format>),
(714,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 121 stored elements in Compressed Sparse Row format>),
(715,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 70 stored elements in Compressed Sparse Row format>),
(716,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 208 stored elements in Compressed Sparse Row format>),
(717,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 58 stored elements in Compressed Sparse Row format>),
(718,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 264 stored elements in Compressed Sparse Row format>),
(719,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 70 stored elements in Compressed Sparse Row format>),
(720,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(721,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(722,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 60 stored elements in Compressed Sparse Row format>),
(723,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(724,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(725,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 188 stored elements in Compressed Sparse Row format>),
(726,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(727,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 122 stored elements in Compressed Sparse Row format>),
(728,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 242 stored elements in Compressed Sparse Row format>),
(729,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 269 stored elements in Compressed Sparse Row format>),
(730,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 348 stored elements in Compressed Sparse Row format>),
(731,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 247 stored elements in Compressed Sparse Row format>),
(732,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 196 stored elements in Compressed Sparse Row format>),
(733,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 227 stored elements in Compressed Sparse Row format>),
(734,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 308 stored elements in Compressed Sparse Row format>),
(735,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 228 stored elements in Compressed Sparse Row format>),
(736,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 127 stored elements in Compressed Sparse Row format>),
(737,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(738,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 136 stored elements in Compressed Sparse Row format>),
(739,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 355 stored elements in Compressed Sparse Row format>),
(740,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 335 stored elements in Compressed Sparse Row format>),
(741,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 224 stored elements in Compressed Sparse Row format>),
(742,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(743,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(744,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 358 stored elements in Compressed Sparse Row format>),
(745,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
```

```
(746,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 275 stored elements in Compressed Sparse Row format>),
(747,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(748,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 213 stored elements in Compressed Sparse Row format>),
(749,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 246 stored elements in Compressed Sparse Row format>),
(750,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 119 stored elements in Compressed Sparse Row format>),
(751,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(752,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(753,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 338 stored elements in Compressed Sparse Row format>),
(754,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(755,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 156 stored elements in Compressed Sparse Row format>),
(756,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 130 stored elements in Compressed Sparse Row format>),
(757,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(758,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(759,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 134 stored elements in Compressed Sparse Row format>),
(760,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 243 stored elements in Compressed Sparse Row format>),
(761,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 357 stored elements in Compressed Sparse Row format>),
(762,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 273 stored elements in Compressed Sparse Row format>),
(763,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 367 stored elements in Compressed Sparse Row format>),
(764,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 309 stored elements in Compressed Sparse Row format>),
(765,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(766,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 314 stored elements in Compressed Sparse Row format>),
(767,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 170 stored elements in Compressed Sparse Row format>),
(768,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 287 stored elements in Compressed Sparse Row format>),
(769,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 171 stored elements in Compressed Sparse Row format>),
(770,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 226 stored elements in Compressed Sparse Row format>),
(771,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 67 stored elements in Compressed Sparse Row format>),
(772,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 227 stored elements in Compressed Sparse Row format>),
(773,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 264 stored elements in Compressed Sparse Row format>),
(774,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 109 stored elements in Compressed Sparse Row format>),
(775,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 41 stored elements in Compressed Sparse Row format>),
(776,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 64 stored elements in Compressed Sparse Row format>),
(777,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 173 stored elements in Compressed Sparse Row format>),
(778,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 184 stored elements in Compressed Sparse Row format>),
(779,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 94 stored elements in Compressed Sparse Row format>),
(780,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 126 stored elements in Compressed Sparse Row format>),
(781,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 296 stored elements in Compressed Sparse Row format>),
(782,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 115 stored elements in Compressed Sparse Row format>),
(783,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 321 stored elements in Compressed Sparse Row format>),
(784,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(785,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 205 stored elements in Compressed Sparse Row format>),
(786,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 64 stored elements in Compressed Sparse Row format>),
(787,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 308 stored elements in Compressed Sparse Row format>),
(788,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 70 stored elements in Compressed Sparse Row format>),
(789,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 405 stored elements in Compressed Sparse Row format>),
(790,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 115 stored elements in Compressed Sparse Row format>),
(791,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(792,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 324 stored elements in Compressed Sparse Row format>),
(793,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 224 stored elements in Compressed Sparse Row format>),
(794,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 65 stored elements in Compressed Sparse Row format>),
(795,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(796,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 217 stored elements in Compressed Sparse Row format>),
(797,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 305 stored elements in Compressed Sparse Row format>),
(798,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 225 stored elements in Compressed Sparse Row format>),
(799,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(800,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 278 stored elements in Compressed Sparse Row format>),
(801,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 33 stored elements in Compressed Sparse Row format>),
(802,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(803,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 13 stored elements in Compressed Sparse Row format>),
(804,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 144 stored elements in Compressed Sparse Row format>),
(805,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 144 stored elements in Compressed Sparse Row format>),
(806,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 197 stored elements in Compressed Sparse Row format>),
(807,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 171 stored elements in Compressed Sparse Row format>),
(808,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 205 stored elements in Compressed Sparse Row format>),
(809,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 361 stored elements in Compressed Sparse Row format>),
(810,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 134 stored elements in Compressed Sparse Row format>),
```

```
(811,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 169 stored elements in Compressed Sparse Row format>),
(812,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 164 stored elements in Compressed Sparse Row format>),
(813,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 131 stored elements in Compressed Sparse Row format>),
(814,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(815,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 33 stored elements in Compressed Sparse Row format>),
(816,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 223 stored elements in Compressed Sparse Row format>),
(817,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(818,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 201 stored elements in Compressed Sparse Row format>),
(819,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(820,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 160 stored elements in Compressed Sparse Row format>),
(821,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 358 stored elements in Compressed Sparse Row format>),
(822,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 214 stored elements in Compressed Sparse Row format>),
(823,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(824,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 73 stored elements in Compressed Sparse Row format>),
(825,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(826,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 189 stored elements in Compressed Sparse Row format>),
(827,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 67 stored elements in Compressed Sparse Row format>),
(828,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(829,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 311 stored elements in Compressed Sparse Row format>),
(830,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 206 stored elements in Compressed Sparse Row format>),
(831,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(832,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 180 stored elements in Compressed Sparse Row format>),
(833,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 221 stored elements in Compressed Sparse Row format>),
(834,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 107 stored elements in Compressed Sparse Row format>),
(835,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 226 stored elements in Compressed Sparse Row format>),
(836,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 212 stored elements in Compressed Sparse Row format>),
(837,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 213 stored elements in Compressed Sparse Row format>),
(838,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 218 stored elements in Compressed Sparse Row format>),
(839,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(840,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 157 stored elements in Compressed Sparse Row format>),
(841,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 266 stored elements in Compressed Sparse Row format>),
(842,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 68 stored elements in Compressed Sparse Row format>),
(843,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(844,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 153 stored elements in Compressed Sparse Row format>),
(845,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(846,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 168 stored elements in Compressed Sparse Row format>),
(847,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 77 stored elements in Compressed Sparse Row format>),
(848,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 387 stored elements in Compressed Sparse Row format>),
(849,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 169 stored elements in Compressed Sparse Row format>),
(850,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 0 stored elements in Compressed Sparse Row format>),
(851,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 294 stored elements in Compressed Sparse Row format>),
(852,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 233 stored elements in Compressed Sparse Row format>),
(853,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 198 stored elements in Compressed Sparse Row format>),
(854,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 358 stored elements in Compressed Sparse Row format>),
(855,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 191 stored elements in Compressed Sparse Row format>),
(856,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(857,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 175 stored elements in Compressed Sparse Row format>),
(858,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 159 stored elements in Compressed Sparse Row format>),
(859,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(860,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 253 stored elements in Compressed Sparse Row format>),
(861,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 284 stored elements in Compressed Sparse Row format>),
(862,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(863,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 76 stored elements in Compressed Sparse Row format>),
(864,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 223 stored elements in Compressed Sparse Row format>),
(865,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 358 stored elements in Compressed Sparse Row format>),
(866,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(867,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 32 stored elements in Compressed Sparse Row format>),
(868,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 321 stored elements in Compressed Sparse Row format>),
(869,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 204 stored elements in Compressed Sparse Row format>),
(870,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 208 stored elements in Compressed Sparse Row format>),
(871,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(872,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 66 stored elements in Compressed Sparse Row format>),
(873,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 212 stored elements in Compressed Sparse Row format>),
(874,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(875,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 335 stored elements in Compressed Sparse Row format>),
```

```
(876,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 107 stored elements in Compressed Sparse Row format>),
(877,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 68 stored elements in Compressed Sparse Row format>),
(878,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(879,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 216 stored elements in Compressed Sparse Row format>),
(880,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 136 stored elements in Compressed Sparse Row format>),
(881,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(882,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 255 stored elements in Compressed Sparse Row format>),
(883,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(884,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 100 stored elements in Compressed Sparse Row format>),
(885,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 240 stored elements in Compressed Sparse Row format>),
(886,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 309 stored elements in Compressed Sparse Row format>),
(887,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 263 stored elements in Compressed Sparse Row format>),
(888,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 121 stored elements in Compressed Sparse Row format>),
(889,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 62 stored elements in Compressed Sparse Row format>),
(890,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 148 stored elements in Compressed Sparse Row format>),
(891,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 33 stored elements in Compressed Sparse Row format>),
(892,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 123 stored elements in Compressed Sparse Row format>),
(893,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 72 stored elements in Compressed Sparse Row format>),
(894,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 278 stored elements in Compressed Sparse Row format>),
(895,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(896,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 69 stored elements in Compressed Sparse Row format>),
(897,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 74 stored elements in Compressed Sparse Row format>),
(898,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 211 stored elements in Compressed Sparse Row format>),
(899,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 121 stored elements in Compressed Sparse Row format>),
(900,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 284 stored elements in Compressed Sparse Row format>),
(901,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 134 stored elements in Compressed Sparse Row format>),
(902,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 168 stored elements in Compressed Sparse Row format>),
(903,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 111 stored elements in Compressed Sparse Row format>),
(904,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 130 stored elements in Compressed Sparse Row format>),
(905,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 230 stored elements in Compressed Sparse Row format>),
(906,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 31 stored elements in Compressed Sparse Row format>),
(907,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 233 stored elements in Compressed Sparse Row format>),
(908,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 59 stored elements in Compressed Sparse Row format>),
(909,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 248 stored elements in Compressed Sparse Row format>),
(910,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 251 stored elements in Compressed Sparse Row format>),
(911,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 274 stored elements in Compressed Sparse Row format>),
(912,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 71 stored elements in Compressed Sparse Row format>),
(913,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 130 stored elements in Compressed Sparse Row format>),
(914,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 204 stored elements in Compressed Sparse Row format>),
(915,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 117 stored elements in Compressed Sparse Row format>),
(916,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(917,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 296 stored elements in Compressed Sparse Row format>),
(918,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 266 stored elements in Compressed Sparse Row format>),
(919,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 190 stored elements in Compressed Sparse Row format>),
(920,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 60 stored elements in Compressed Sparse Row format>),
(921,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 322 stored elements in Compressed Sparse Row format>),
(922,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 61 stored elements in Compressed Sparse Row format>),
(923,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(924,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 126 stored elements in Compressed Sparse Row format>),
(925,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 320 stored elements in Compressed Sparse Row format>),
(926,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 74 stored elements in Compressed Sparse Row format>),
(927,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 334 stored elements in Compressed Sparse Row format>),
(928,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 260 stored elements in Compressed Sparse Row format>),
(929,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(930,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(931,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 345 stored elements in Compressed Sparse Row format>),
(932,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 60 stored elements in Compressed Sparse Row format>),
(933,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 197 stored elements in Compressed Sparse Row format>),
(934,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 100 stored elements in Compressed Sparse Row format>),
(935,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 278 stored elements in Compressed Sparse Row format>),
(936,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 334 stored elements in Compressed Sparse Row format>),
(937,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 256 stored elements in Compressed Sparse Row format>),
(938,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(939,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 235 stored elements in Compressed Sparse Row format>),
(940,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 123 stored elements in Compressed Sparse Row format>),
```

```
(941,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 253 stored elements in Compressed Sparse Row format>),
(942,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 231 stored elements in Compressed Sparse Row format>),
(943,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 209 stored elements in Compressed Sparse Row format>),
(944,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 123 stored elements in Compressed Sparse Row format>),
(945,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 171 stored elements in Compressed Sparse Row format>),
(946,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 63 stored elements in Compressed Sparse Row format>),
(947,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 70 stored elements in Compressed Sparse Row format>),
(948,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 227 stored elements in Compressed Sparse Row format>),
(949,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 381 stored elements in Compressed Sparse Row format>),
(950,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 276 stored elements in Compressed Sparse Row format>),
(951,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 265 stored elements in Compressed Sparse Row format>),
(952,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 163 stored elements in Compressed Sparse Row format>),
(953,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 262 stored elements in Compressed Sparse Row format>),
(954,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 230 stored elements in Compressed Sparse Row format>),
(955,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 64 stored elements in Compressed Sparse Row format>),
(956,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 137 stored elements in Compressed Sparse Row format>),
(957,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 216 stored elements in Compressed Sparse Row format>),
(958,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 118 stored elements in Compressed Sparse Row format>),
(959,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 114 stored elements in Compressed Sparse Row format>),
(960,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 217 stored elements in Compressed Sparse Row format>),
(961,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
       with 71 stored elements in Compressed Sparse Row format>),
(962,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
```

```
      with 61 stored elements in Compressed Sparse Row format>),
(963,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 231 stored elements in Compressed Sparse Row format>),
(964,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 34 stored elements in Compressed Sparse Row format>),
(965,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 295 stored elements in Compressed Sparse Row format>),
(966,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 140 stored elements in Compressed Sparse Row format>),
(967,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 342 stored elements in Compressed Sparse Row format>),
(968,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 66 stored elements in Compressed Sparse Row format>),
(969,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 62 stored elements in Compressed Sparse Row format>),
(970,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 336 stored elements in Compressed Sparse Row format>),
(971,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 68 stored elements in Compressed Sparse Row format>),
(972,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 72 stored elements in Compressed Sparse Row format>),
(973,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 220 stored elements in Compressed Sparse Row format>),
(974,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 100 stored elements in Compressed Sparse Row format>),
(975,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 69 stored elements in Compressed Sparse Row format>),
(976,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 37 stored elements in Compressed Sparse Row format>),
(977,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 304 stored elements in Compressed Sparse Row format>),
(978,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 245 stored elements in Compressed Sparse Row format>),
(979,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 65 stored elements in Compressed Sparse Row format>),
(980,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 115 stored elements in Compressed Sparse Row format>),
(981,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 268 stored elements in Compressed Sparse Row format>),
(982,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 162 stored elements in Compressed Sparse Row format>),
(983,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
      with 170 stored elements in Compressed Sparse Row format>),
(984,
```

```
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 38 stored elements in Compressed Sparse Row format>),
(985,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 212 stored elements in Compressed Sparse Row format>),
(986,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 115 stored elements in Compressed Sparse Row format>),
(987,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 66 stored elements in Compressed Sparse Row format>),
(988,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 59 stored elements in Compressed Sparse Row format>),
(989,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 71 stored elements in Compressed Sparse Row format>),
(990,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 222 stored elements in Compressed Sparse Row format>),
(991,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 237 stored elements in Compressed Sparse Row format>),
(992,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 259 stored elements in Compressed Sparse Row format>),
(993,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 362 stored elements in Compressed Sparse Row format>),
(994,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 65 stored elements in Compressed Sparse Row format>),
(995,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 176 stored elements in Compressed Sparse Row format>),
(996,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 243 stored elements in Compressed Sparse Row format>),
(997,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 65 stored elements in Compressed Sparse Row format>),
(998,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 30 stored elements in Compressed Sparse Row format>),
(999,
 <1x676 sparse matrix of type '<class 'numpy.int64'>'
	with 224 stored elements in Compressed Sparse Row format>),
...]
```

In [9]:

```
start = time.time()

for res_tuple in res:
    index = res_tuple[0]
    processed = res_tuple[1]
    opcodebivect[index, :] = processed
    if index % 100 == 0:
        print ("Done with the index ", index)
opcodebivect = opcodebivect.tocsr()
end = time.time()

print("opcodebivect time: ", (end-start))
print ("opcodebivect constructed")
```

```
Done with the index  0
```

```
Done with the index  100
Done with the index  200
Done with the index  300
Done with the index  400
Done with the index  500
Done with the index  600
Done with the index  700
Done with the index  800
Done with the index  900
Done with the index  1000
Done with the index  1100
Done with the index  1200
Done with the index  1300
Done with the index  1400
Done with the index  1500
Done with the index  1600
Done with the index  1700
Done with the index  1800
Done with the index  1900
Done with the index  2000
Done with the index  2100
Done with the index  2200
Done with the index  2300
Done with the index  2400
Done with the index  2500
Done with the index  2600
Done with the index  2700
Done with the index  2800
Done with the index  2900
Done with the index  3000
Done with the index  3100
Done with the index  3200
Done with the index  3300
Done with the index  3400
Done with the index  3500
Done with the index  3600
Done with the index  3700
Done with the index  3800
Done with the index  3900
Done with the index  4000
Done with the index  4100
Done with the index  4200
Done with the index  4300
Done with the index  4400
Done with the index  4500
Done with the index  4600
Done with the index  4700
Done with the index  4800
Done with the index  4900
Done with the index  5000
Done with the index  5100
Done with the index  5200
Done with the index  5300
Done with the index  5400
Done with the index  5500
Done with the index  5600
Done with the index  5700
Done with the index  5800
Done with the index  5900
Done with the index  6000
Done with the index  6100
Done with the index  6200
Done with the index  6300
Done with the index  6400
Done with the index  6500
```

```
Done with the index  6600
Done with the index  6700
Done with the index  6800
Done with the index  6900
Done with the index  7000
Done with the index  7100
Done with the index  7200
Done with the index  7300
Done with the index  7400
Done with the index  7500
Done with the index  7600
Done with the index  7700
Done with the index  7800
Done with the index  7900
Done with the index  8000
Done with the index  8100
Done with the index  8200
Done with the index  8300
Done with the index  8400
Done with the index  8500
Done with the index  8600
Done with the index  8700
Done with the index  8800
Done with the index  8900
Done with the index  9000
Done with the index  9100
Done with the index  9200
Done with the index  9300
Done with the index  9400
Done with the index  9500
Done with the index  9600
Done with the index  9700
Done with the index  9800
Done with the index  9900
Done with the index  10000
Done with the index  10100
Done with the index  10200
Done with the index  10300
Done with the index  10400
Done with the index  10500
Done with the index  10600
Done with the index  10700
Done with the index  10800
opcodebivect time:  3.604249954223633
opcodebivect constructed
```

In [9]:
```
opcodebivect
```

Out[9]:
```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'
        with 1877309 stored elements in Compressed Sparse Row format>
```

In [11]:
```
scipy.sparse.save_npz('opcodebigram.npz', opcodebivect)
```

In [17]:
```
opcodebivect=scipy.sparse.load_npz('opcodebigram.npz')
```

In [ ]:
```
# code for opcodetrivect
vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asmopcodetrigram)
opcodetrivect = scipy.sparse.csr_matrix((10868, len(asmopcodetrigram)))

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

In [ ]:
```
# code for opcodetetravect
vect = CountVectorizer(ngram_range=(4, 4), vocabulary = asmopcodetetragram)
opcodetetravect = scipy.sparse.csr_matrix((10868, len(asmopcodetetragram)))

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

Image Feature Extraction From ASM Files

In [35]:
```
import array
import imageio

def collect_img_asm():
    for asmfile in os.listdir("asmFiles"):
        filename = asmfile.split('.')[0]
        file = codecs.open("asmFiles/" + asmfile, 'rb')
        filelen = os.path.getsize("asmFiles/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imageio.imwrite('asm_image/' + filename + '.png',reshaped)
```

In [18]:
```
collect_img_asm()
```

In [ ]:
```
from IPython.display import Image
Image(filename='asm_image/deTXH9Zau7qmM0yfYsRS.png')
```

First 200 Image Pixels

In [37]:
```
imagefeatures = np.zeros((10868, 200))
#imagefeatures = np.zeros((10868, 800))
```

In [38]:
```
import cv2
for i, asmfile in enumerate(os.listdir("asmFiles")):
    img = cv2.imread("asm_image/" + asmfile.split('.')[0] + '.png')
    img_arr = img.flatten()[:200]
    imagefeatures[i, :] += img_arr
    if i % 100 == 0:
        print ("index done ", i)
```

```
index done  0
index done  100
index done  200
index done  300
index done  400
index done  500
index done  600
index done  700
index done  800
index done  900
index done  1000
index done  1100
index done  1200
```

```
index done  1300
index done  1400
index done  1500
index done  1600
index done  1700
index done  1800
index done  1900
index done  2000
index done  2100
index done  2200
index done  2300
index done  2400
index done  2500
index done  2600
index done  2700
index done  2800
index done  2900
index done  3000
index done  3100
index done  3200
index done  3300
index done  3400
index done  3500
index done  3600
index done  3700
index done  3800
index done  3900
index done  4000
index done  4100
index done  4200
index done  4300
index done  4400
index done  4500
index done  4600
index done  4700
index done  4800
index done  4900
index done  5000
index done  5100
index done  5200
index done  5300
index done  5400
index done  5500
index done  5600
index done  5700
index done  5800
index done  5900
index done  6000
index done  6100
index done  6200
index done  6300
index done  6400
index done  6500
index done  6600
index done  6700
index done  6800
index done  6900
index done  7000
index done  7100
index done  7200
index done  7300
index done  7400
index done  7500
index done  7600
index done  7700
```

```
index done  7800
index done  7900
index done  8000
index done  8100
index done  8200
index done  8300
index done  8400
index done  8500
index done  8600
index done  8700
index done  8800
index done  8900
index done  9000
index done  9100
index done  9200
index done  9300
index done  9400
index done  9500
index done  9600
index done  9700
index done  9800
index done  9900
index done  10000
index done  10100
index done  10200
index done  10300
index done  10400
index done  10500
index done  10600
index done  10700
index done  10800
```

In [39]:
```
imgfeatures_name = []
for i in range(200):
    imgfeatures_name.append('pix' + str(i))
```

In [40]:
```
imgfeatures_name
```

Out[40]:
```
['pix0',
 'pix1',
 'pix2',
 'pix3',
 'pix4',
 'pix5',
 'pix6',
 'pix7',
 'pix8',
 'pix9',
 'pix10',
 'pix11',
 'pix12',
 'pix13',
 'pix14',
 'pix15',
 'pix16',
 'pix17',
 'pix18',
 'pix19',
 'pix20',
 'pix21',
 'pix22',
 'pix23',
 'pix24',
```

```
'pix25',
'pix26',
'pix27',
'pix28',
'pix29',
'pix30',
'pix31',
'pix32',
'pix33',
'pix34',
'pix35',
'pix36',
'pix37',
'pix38',
'pix39',
'pix40',
'pix41',
'pix42',
'pix43',
'pix44',
'pix45',
'pix46',
'pix47',
'pix48',
'pix49',
'pix50',
'pix51',
'pix52',
'pix53',
'pix54',
'pix55',
'pix56',
'pix57',
'pix58',
'pix59',
'pix60',
'pix61',
'pix62',
'pix63',
'pix64',
'pix65',
'pix66',
'pix67',
'pix68',
'pix69',
'pix70',
'pix71',
'pix72',
'pix73',
'pix74',
'pix75',
'pix76',
'pix77',
'pix78',
'pix79',
'pix80',
'pix81',
'pix82',
'pix83',
'pix84',
'pix85',
'pix86',
'pix87',
'pix88',
'pix89',
```

```
'pix90',
'pix91',
'pix92',
'pix93',
'pix94',
'pix95',
'pix96',
'pix97',
'pix98',
'pix99',
'pix100',
'pix101',
'pix102',
'pix103',
'pix104',
'pix105',
'pix106',
'pix107',
'pix108',
'pix109',
'pix110',
'pix111',
'pix112',
'pix113',
'pix114',
'pix115',
'pix116',
'pix117',
'pix118',
'pix119',
'pix120',
'pix121',
'pix122',
'pix123',
'pix124',
'pix125',
'pix126',
'pix127',
'pix128',
'pix129',
'pix130',
'pix131',
'pix132',
'pix133',
'pix134',
'pix135',
'pix136',
'pix137',
'pix138',
'pix139',
'pix140',
'pix141',
'pix142',
'pix143',
'pix144',
'pix145',
'pix146',
'pix147',
'pix148',
'pix149',
'pix150',
'pix151',
'pix152',
'pix153',
'pix154',
```

```
'pix155',
'pix156',
'pix157',
'pix158',
'pix159',
'pix160',
'pix161',
'pix162',
'pix163',
'pix164',
'pix165',
'pix166',
'pix167',
'pix168',
'pix169',
'pix170',
'pix171',
'pix172',
'pix173',
'pix174',
'pix175',
'pix176',
'pix177',
'pix178',
'pix179',
'pix180',
'pix181',
'pix182',
'pix183',
'pix184',
'pix185',
'pix186',
'pix187',
'pix188',
'pix189',
'pix190',
'pix191',
'pix192',
'pix193',
'pix194',
'pix195',
'pix196',
'pix197',
'pix198',
'pix199',
'pix200',
'pix201',
'pix202',
'pix203',
'pix204',
'pix205',
'pix206',
'pix207',
'pix208',
'pix209',
'pix210',
'pix211',
'pix212',
'pix213',
'pix214',
'pix215',
'pix216',
'pix217',
'pix218',
'pix219',
```

```
'pix220',
'pix221',
'pix222',
'pix223',
'pix224',
'pix225',
'pix226',
'pix227',
'pix228',
'pix229',
'pix230',
'pix231',
'pix232',
'pix233',
'pix234',
'pix235',
'pix236',
'pix237',
'pix238',
'pix239',
'pix240',
'pix241',
'pix242',
'pix243',
'pix244',
'pix245',
'pix246',
'pix247',
'pix248',
'pix249',
'pix250',
'pix251',
'pix252',
'pix253',
'pix254',
'pix255',
'pix256',
'pix257',
'pix258',
'pix259',
'pix260',
'pix261',
'pix262',
'pix263',
'pix264',
'pix265',
'pix266',
'pix267',
'pix268',
'pix269',
'pix270',
'pix271',
'pix272',
'pix273',
'pix274',
'pix275',
'pix276',
'pix277',
'pix278',
'pix279',
'pix280',
'pix281',
'pix282',
'pix283',
'pix284',
```

```
'pix285',
'pix286',
'pix287',
'pix288',
'pix289',
'pix290',
'pix291',
'pix292',
'pix293',
'pix294',
'pix295',
'pix296',
'pix297',
'pix298',
'pix299',
'pix300',
'pix301',
'pix302',
'pix303',
'pix304',
'pix305',
'pix306',
'pix307',
'pix308',
'pix309',
'pix310',
'pix311',
'pix312',
'pix313',
'pix314',
'pix315',
'pix316',
'pix317',
'pix318',
'pix319',
'pix320',
'pix321',
'pix322',
'pix323',
'pix324',
'pix325',
'pix326',
'pix327',
'pix328',
'pix329',
'pix330',
'pix331',
'pix332',
'pix333',
'pix334',
'pix335',
'pix336',
'pix337',
'pix338',
'pix339',
'pix340',
'pix341',
'pix342',
'pix343',
'pix344',
'pix345',
'pix346',
'pix347',
'pix348',
'pix349',
```

```
'pix350',
'pix351',
'pix352',
'pix353',
'pix354',
'pix355',
'pix356',
'pix357',
'pix358',
'pix359',
'pix360',
'pix361',
'pix362',
'pix363',
'pix364',
'pix365',
'pix366',
'pix367',
'pix368',
'pix369',
'pix370',
'pix371',
'pix372',
'pix373',
'pix374',
'pix375',
'pix376',
'pix377',
'pix378',
'pix379',
'pix380',
'pix381',
'pix382',
'pix383',
'pix384',
'pix385',
'pix386',
'pix387',
'pix388',
'pix389',
'pix390',
'pix391',
'pix392',
'pix393',
'pix394',
'pix395',
'pix396',
'pix397',
'pix398',
'pix399',
'pix400',
'pix401',
'pix402',
'pix403',
'pix404',
'pix405',
'pix406',
'pix407',
'pix408',
'pix409',
'pix410',
'pix411',
'pix412',
'pix413',
'pix414',
```

```
'pix415',
'pix416',
'pix417',
'pix418',
'pix419',
'pix420',
'pix421',
'pix422',
'pix423',
'pix424',
'pix425',
'pix426',
'pix427',
'pix428',
'pix429',
'pix430',
'pix431',
'pix432',
'pix433',
'pix434',
'pix435',
'pix436',
'pix437',
'pix438',
'pix439',
'pix440',
'pix441',
'pix442',
'pix443',
'pix444',
'pix445',
'pix446',
'pix447',
'pix448',
'pix449',
'pix450',
'pix451',
'pix452',
'pix453',
'pix454',
'pix455',
'pix456',
'pix457',
'pix458',
'pix459',
'pix460',
'pix461',
'pix462',
'pix463',
'pix464',
'pix465',
'pix466',
'pix467',
'pix468',
'pix469',
'pix470',
'pix471',
'pix472',
'pix473',
'pix474',
'pix475',
'pix476',
'pix477',
'pix478',
'pix479',
```

```
'pix480',
'pix481',
'pix482',
'pix483',
'pix484',
'pix485',
'pix486',
'pix487',
'pix488',
'pix489',
'pix490',
'pix491',
'pix492',
'pix493',
'pix494',
'pix495',
'pix496',
'pix497',
'pix498',
'pix499',
'pix500',
'pix501',
'pix502',
'pix503',
'pix504',
'pix505',
'pix506',
'pix507',
'pix508',
'pix509',
'pix510',
'pix511',
'pix512',
'pix513',
'pix514',
'pix515',
'pix516',
'pix517',
'pix518',
'pix519',
'pix520',
'pix521',
'pix522',
'pix523',
'pix524',
'pix525',
'pix526',
'pix527',
'pix528',
'pix529',
'pix530',
'pix531',
'pix532',
'pix533',
'pix534',
'pix535',
'pix536',
'pix537',
'pix538',
'pix539',
'pix540',
'pix541',
'pix542',
'pix543',
'pix544',
```

```
'pix545',
'pix546',
'pix547',
'pix548',
'pix549',
'pix550',
'pix551',
'pix552',
'pix553',
'pix554',
'pix555',
'pix556',
'pix557',
'pix558',
'pix559',
'pix560',
'pix561',
'pix562',
'pix563',
'pix564',
'pix565',
'pix566',
'pix567',
'pix568',
'pix569',
'pix570',
'pix571',
'pix572',
'pix573',
'pix574',
'pix575',
'pix576',
'pix577',
'pix578',
'pix579',
'pix580',
'pix581',
'pix582',
'pix583',
'pix584',
'pix585',
'pix586',
'pix587',
'pix588',
'pix589',
'pix590',
'pix591',
'pix592',
'pix593',
'pix594',
'pix595',
'pix596',
'pix597',
'pix598',
'pix599',
'pix600',
'pix601',
'pix602',
'pix603',
'pix604',
'pix605',
'pix606',
'pix607',
'pix608',
'pix609',
```

```
'pix610',
'pix611',
'pix612',
'pix613',
'pix614',
'pix615',
'pix616',
'pix617',
'pix618',
'pix619',
'pix620',
'pix621',
'pix622',
'pix623',
'pix624',
'pix625',
'pix626',
'pix627',
'pix628',
'pix629',
'pix630',
'pix631',
'pix632',
'pix633',
'pix634',
'pix635',
'pix636',
'pix637',
'pix638',
'pix639',
'pix640',
'pix641',
'pix642',
'pix643',
'pix644',
'pix645',
'pix646',
'pix647',
'pix648',
'pix649',
'pix650',
'pix651',
'pix652',
'pix653',
'pix654',
'pix655',
'pix656',
'pix657',
'pix658',
'pix659',
'pix660',
'pix661',
'pix662',
'pix663',
'pix664',
'pix665',
'pix666',
'pix667',
'pix668',
'pix669',
'pix670',
'pix671',
'pix672',
'pix673',
'pix674',
```

```
'pix675',
'pix676',
'pix677',
'pix678',
'pix679',
'pix680',
'pix681',
'pix682',
'pix683',
'pix684',
'pix685',
'pix686',
'pix687',
'pix688',
'pix689',
'pix690',
'pix691',
'pix692',
'pix693',
'pix694',
'pix695',
'pix696',
'pix697',
'pix698',
'pix699',
'pix700',
'pix701',
'pix702',
'pix703',
'pix704',
'pix705',
'pix706',
'pix707',
'pix708',
'pix709',
'pix710',
'pix711',
'pix712',
'pix713',
'pix714',
'pix715',
'pix716',
'pix717',
'pix718',
'pix719',
'pix720',
'pix721',
'pix722',
'pix723',
'pix724',
'pix725',
'pix726',
'pix727',
'pix728',
'pix729',
'pix730',
'pix731',
'pix732',
'pix733',
'pix734',
'pix735',
'pix736',
'pix737',
'pix738',
'pix739',
```

```
 'pix740',
 'pix741',
 'pix742',
 'pix743',
 'pix744',
 'pix745',
 'pix746',
 'pix747',
 'pix748',
 'pix749',
 'pix750',
 'pix751',
 'pix752',
 'pix753',
 'pix754',
 'pix755',
 'pix756',
 'pix757',
 'pix758',
 'pix759',
 'pix760',
 'pix761',
 'pix762',
 'pix763',
 'pix764',
 'pix765',
 'pix766',
 'pix767',
 'pix768',
 'pix769',
 'pix770',
 'pix771',
 'pix772',
 'pix773',
 'pix774',
 'pix775',
 'pix776',
 'pix777',
 'pix778',
 'pix779',
 'pix780',
 'pix781',
 'pix782',
 'pix783',
 'pix784',
 'pix785',
 'pix786',
 'pix787',
 'pix788',
 'pix789',
 'pix790',
 'pix791',
 'pix792',
 'pix793',
 'pix794',
 'pix795',
 'pix796',
 'pix797',
 'pix798',
 'pix799']
```

In [41]:
```
imagefeatures
```

Out[41]:
```
array([[ 46.,  46.,  46., ...,  32.,  32.,  32.],
```

```
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       ...,
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       [ 72.,  72.,  72., ..., 103., 104., 104.]])
```

In [42]:
```
from sklearn.preprocessing import normalize
imgdf = pd.DataFrame(normalize(imagefeatures, axis=0), columns=imgfeatures_na
```

In [43]:
```
imgdf['ID'] = result.ID
```

In [44]:
```
imgdf.head()
```

Out[44]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	
0	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	(
1	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0
2	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0
3	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0
4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0

5 rows × 801 columns

In [45]:
```
import joblib
joblib.dump(imgdf, 'img_800_df')
```

Out[45]:
```
['img_800_df']
```

In [18]:
```
import joblib
img_df=joblib.load('img_df')
```

In [19]:
```
img_df.head()
```

Out[19]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	
0	0.013153	0.013153	0.013153	0.026990	0.026990	0.026990	0.025855	0.025855	0.(
1	0.020587	0.020587	0.020587	0.016054	0.016054	0.016054	0.016639	0.016639	0.
2	0.020587	0.020587	0.020587	0.016054	0.016054	0.016054	0.016639	0.016639	0.
3	0.020587	0.020587	0.020587	0.016054	0.016054	0.016054	0.016639	0.016639	0.
4	0.020587	0.020587	0.020587	0.016054	0.016054	0.016054	0.016639	0.016639	0.

5 rows × 201 columns

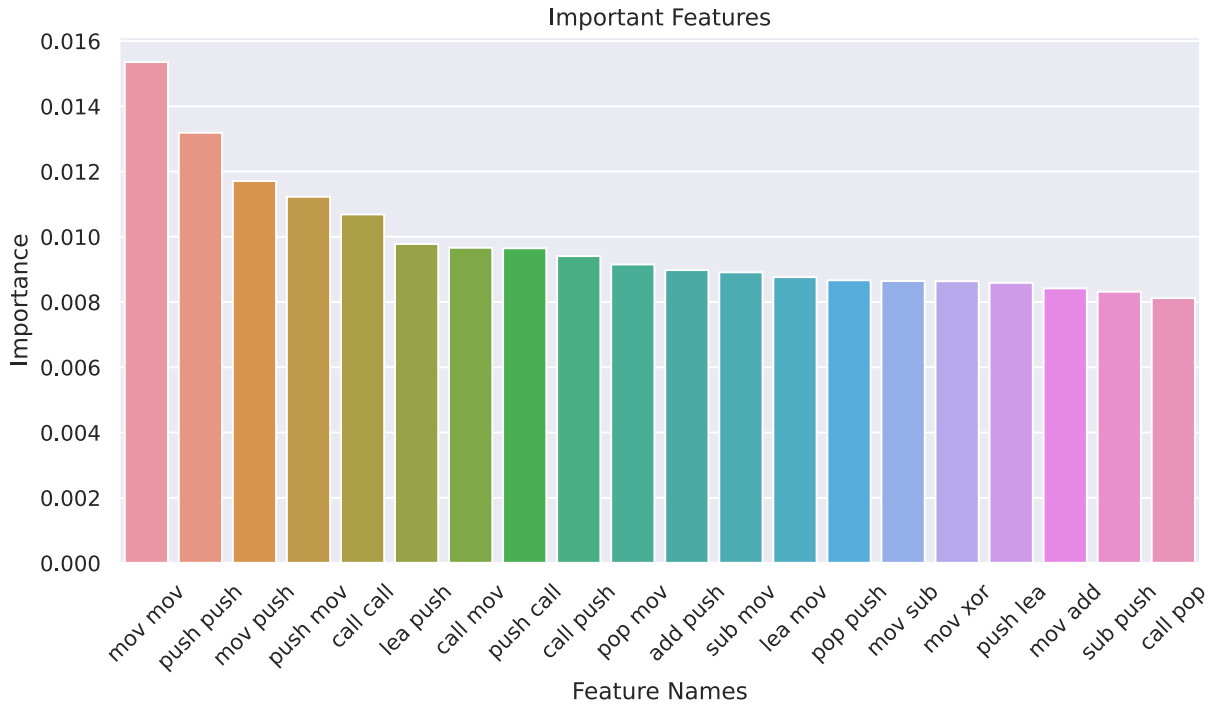
Important Feature Selection Using Random Forest

In [12]:
```
def imp_features(data, features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
```

```
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    sns.set()
    plt.figure(figsize = (10, 5))
    ax = sns.barplot(x = imp_feature_name, y = imp_value)
    ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
    sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
    plt.title('Important Features')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')
    return imp_feature_indx[:keep]
```

In [13]:

```
from sklearn.preprocessing import normalize
op_bi_indxes = imp_features(normalize(opcodebivect, axis = 0), asmopcodebigra
```



In [14]:

```
asmopcodebigram
```

Out[14]:

```
['jmp jmp',
 'jmp mov',
 'jmp retf',
 'jmp push',
 'jmp pop',
 'jmp xor',
 'jmp retn',
 'jmp nop',
 'jmp sub',
 'jmp inc',
 'jmp dec',
 'jmp add',
 'jmp imul',
 'jmp xchg',
 'jmp or',
 'jmp shr',
 'jmp cmp',
 'jmp call',
 'jmp shl',
 'jmp ror',
 'jmp rol',
```

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

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

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

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

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

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

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

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

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

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

```
'movzx jnb',
'movzx jz',
'movzx rtn',
'movzx lea',
'movzx movzx']
```

In [20]:
```
opcodebivect_normalized = normalize(opcodebivect, axis = 0)
```

In [17]:
```
opcodebivect
```

Out[17]:
```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'
        with 1877309 stored elements in Compressed Sparse Row format>
```

In [23]:
```
opcodebivect_normalized
```

Out[23]:
```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'
        with 1877309 stored elements in Compressed Sparse Column format>
```

In [21]:
```
type(opcodebivect_normalized), type(asmopcodebigram)
```

Out[21]:
```
(scipy.sparse.csc.csc_matrix, list)
```

In [25]:
```
op_bi_df = pd.DataFrame(opcodebivect_normalized.toarray(), columns = asmopcod
op_bi_df.head()
```

Out[25]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	
0	0.000000	0.000710	0.0	0.000280	0.000000	0.000140	0.000000	0.0	0.000000	0
1	0.026392	0.002657	0.0	0.002452	0.000000	0.002095	0.000703	0.0	0.016409	0
2	0.001374	0.028437	0.0	0.022765	0.002994	0.024441	0.007730	0.0	0.009323	0
3	0.000000	0.000000	0.0	0.000070	0.000000	0.000000	0.000000	0.0	0.000000	0
4	0.000000	0.000122	0.0	0.000070	0.000000	0.000000	0.000000	0.0	0.000000	0

5 rows × 676 columns

In [26]:
```
for col in op_bi_df.columns:
    if col not in np.take(asmopcodebigram, op_bi_indxes):
        op_bi_df.drop(col, axis = 1, inplace = True)
```

In [28]:
```
op_bi_df.to_csv('op_bi.csv')
```

In [49]:
```
op_bi_df = pd.read_csv('op_bi.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [50]:
```
op_bi_df['ID'] = result.ID
op_bi_df.head()
```

Out[50]:

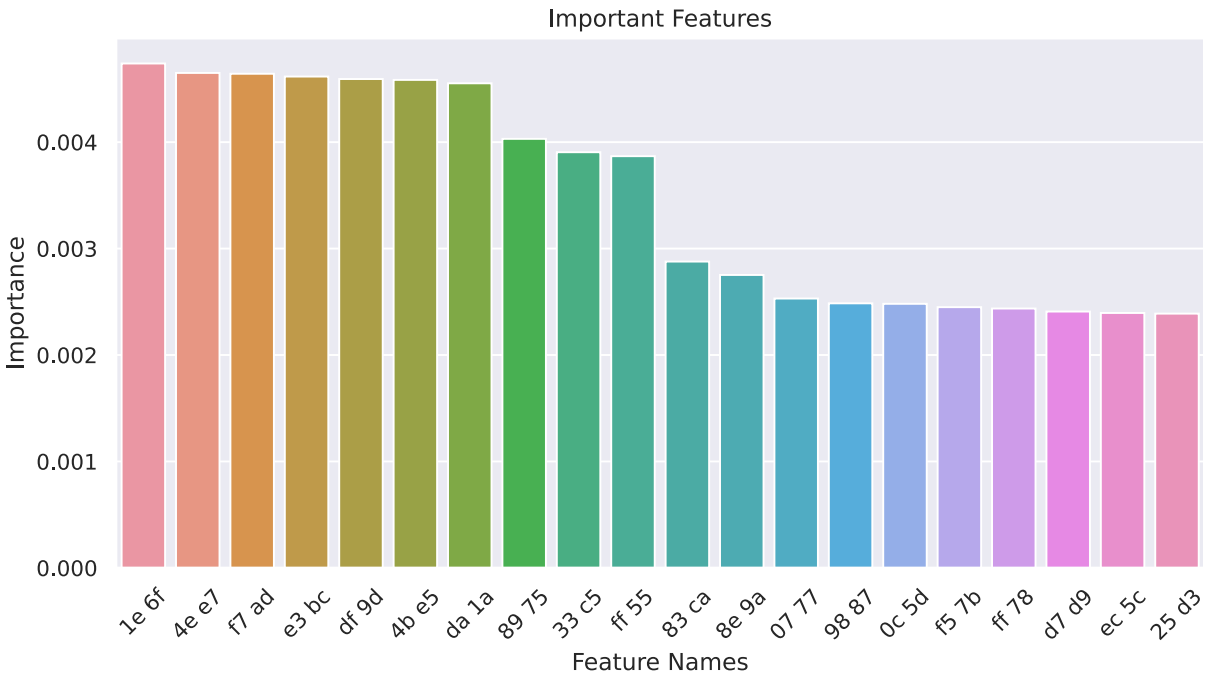
	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp add	jmp cmp	
0	0.000000	0.000710	0.000280	0.000000	0.000140	0.000000	0.000985	0.000000	0

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp add	jmp cmp	
1	0.026392	0.002657	0.002452	0.000000	0.002095	0.016409	0.025127	0.000898	(
2	0.001374	0.028437	0.022765	0.002994	0.024441	0.009323	0.027591	0.031431	
3	0.000000	0.000000	0.000070	0.000000	0.000000	0.000000	0.000000	0.000000	(
4	0.000000	0.000122	0.000070	0.000000	0.000000	0.000000	0.000000	0.000000	(

5 rows × 201 columns

Important Feature Among Byte Bi-Gram

In [37]:
```
byte_bi_indxes = imp_features(normalize(bytebigram_vect, axis = 0), byte_bigr
```



In [38]:
```
np.save('byte_bi_indx', byte_bi_indxes)
```

In [19]:
```
import numpy as np
byte_bi_indxes = np.load('byte_bi_indx.npy')
```

In [39]:
```
top_byte_bi = np.zeros((10868, 0))
for i in byte_bi_indxes:
    sliced = bytebigram_vect[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

In [40]:
```
byte_bi_df = pd.DataFrame(top_byte_bi, columns = np.take(byte_bigram_vocab, b
```

In [41]:
```
byte_bi_df.to_csv('byte_bi.csv')
```

In [20]:
```
byte_bi_df = pd.read_csv('byte_bi.csv').drop('Unnamed: 0', axis = 1).fillna(0
```

In [21]:
```
byte_bi_df['ID'] = result.ID
byte_bi_df.head()
```

Out[21]:

	1e 6f	4e e7	f7 ad	e3 bc	df 9d	4b e5	da 1a	89 75	33 c5	ff 55	...	10 85	98 35	00 6a	c0
0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	68.0	36.0	228.0	...	442.0	0.0	419.0	57
1	32.0	18.0	30.0	26.0	26.0	30.0	28.0	14.0	22.0	72.0	...	19.0	20.0	28.0	31
2	0.0	0.0	0.0	0.0	0.0	1.0	1.0	69.0	2.0	14.0	...	255.0	1.0	112.0	217
3	15.0	18.0	14.0	12.0	16.0	12.0	17.0	12.0	9.0	105.0	...	14.0	12.0	20.0	26
4	2.0	4.0	6.0	6.0	7.0	5.0	9.0	7.0	10.0	9.0	...	5.0	4.0	124.0	134

5 rows × 301 columns

Important Feature Among Opcode 3-Gram

In [ ]:
```
op_tri_indxes = imp_features(normalize(opcodetrivect, axis = 0), asmopcodetri
```

In [ ]:
```
op_tri_df = pd.SparseDataFrame(normalize(opcodetrivect, axis = 0), columns =
op_tri_df = op_tri_df.loc[:, np.intersect1d(op_tri_df.columns, np.take(asmopc
```

In [ ]:
```
op_tri_df.to_dense().to_csv('op_tri.csv')
```

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

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

Important Feature Among Opcode 4-Gram

In [ ]:
```
op_tetra_indxes = imp_features(normalize(opcodetetravect, axis = 0), asmopcod
```

In [ ]:
```
op_tetra_df = pd.SparseDataFrame(normalize(opcodetetravect, axis = 0), column
op_tetra_df = op_tetra_df.loc[:, np.intersect1d(op_tetra_df.columns, np.take(
```

In [ ]:
```
op_tetra_df.to_dense().to_csv('op_tetra.csv')
```

In [ ]:
```
op_tetra_df = pd.read_csv('op_tetra.csv').drop('Unnamed: 0', axis = 1).fillna
```

In [ ]:
```
op_tetra_df['ID'] = result.ID
op_tetra_df.head()
```

Advanced features Adding 300 bytebigram,200 opcode bigram,200 opcode trigram,200 opcode tetragram ,first 200 image pixels

In [51]:
```
final_data = pd.concat([result_x, op_bi_df, byte_bi_df, op_tri_df, op_tetra_d
```

In [53]:
```
final_data = final_data.drop('ID', axis = 1)
```

In [54]:
```
final_data.head()
```

Out[54]:

	0	1	2	3	4	5	6	7
0	0.037960	0.088366	0.034627	0.036951	0.007875	0.003800	0.004795	0.006934
1	0.005980	0.011765	0.003431	0.003242	0.003633	0.003515	0.003511	0.005809
2	0.081074	0.006632	0.000314	0.000500	0.000624	0.000225	0.000232	0.000686
3	0.005230	0.007822	0.001832	0.001787	0.001985	0.001802	0.001841	0.003045
4	0.029241	0.002876	0.000993	0.000966	0.002361	0.000909	0.000953	0.001517

5 rows × 1607 columns

In [55]:
```
final_data.to_csv('final_data_800.csv')
```

In [2]:
```
import pandas as pd
final_data = pd.read_csv('final_data_800.csv')
```

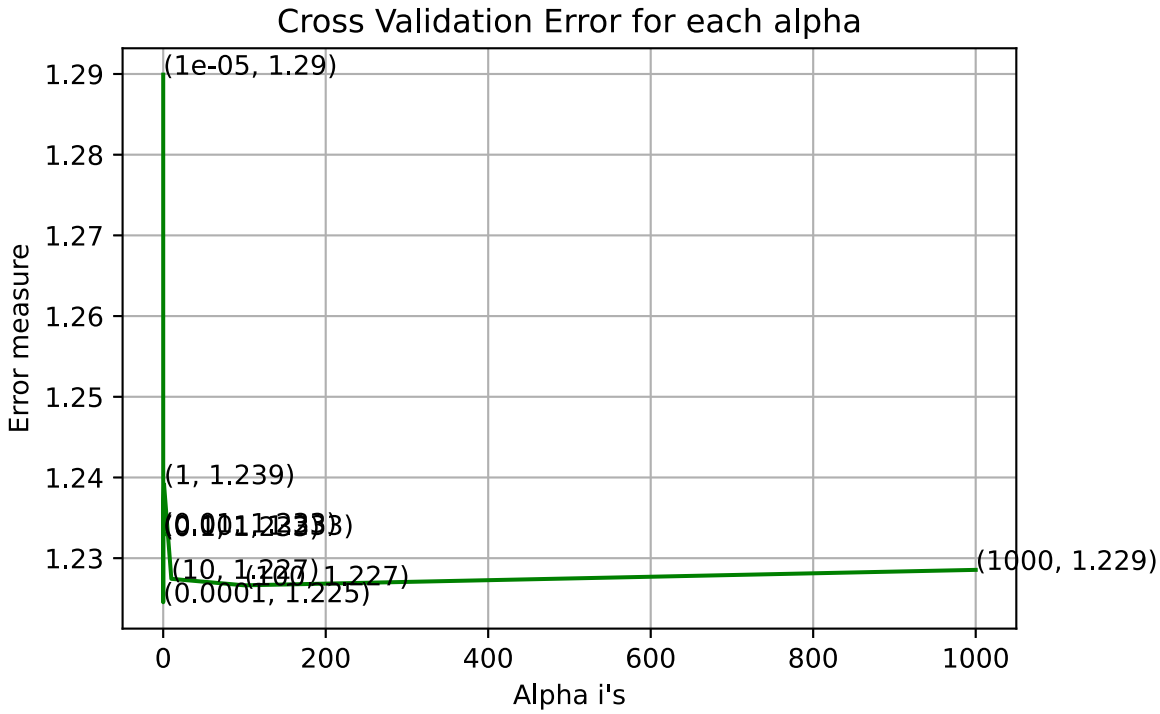
In [57]:
```
x_train_final, x_test_final, y_train_final, y_test_final = train_test_split(f
x_trn_final, x_cv_final, y_trn_final, y_cv_final = train_test_split(x_train_f
```

Machine Learning Models on ASM Features + Byte Features + Advanced Features

In [59]:
```
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_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(x_trn_final,y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=logistic

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

best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  1e-05 is 1.289918346717167
log_loss for c =  0.0001 is 1.2245410659334697
log_loss for c =  0.001 is 1.2329023678256839
log_loss for c =  0.01 is 1.2331802273529182
```

```
log_loss for c =  0.1 is 1.2327550182682634
log_loss for c =  1 is 1.239169910345953
log_loss for c =  10 is 1.2274409594851208
log_loss for c =  100 is 1.226633737735359
log_loss for c =  1000 is 1.2285518521847545
```



In [51]:

```
logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='ba
logisticR.fit(x_trn_final,y_trn_final)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(x_trn_final,y_trn_final)

predict_y = sig_clf.predict_proba(x_trn_final)
print ('log loss for train data',(log_loss(y_trn_final, predict_y, labels=log
predict_y = sig_clf.predict_proba(x_cv_final)
print ('log loss for cv data',(log_loss(y_cv_final, predict_y, labels=logisti
predict_y = sig_clf.predict_proba(x_test_final)
print ('log loss for test data',(log_loss(y_test_final, predict_y, labels=log
```

```
log loss for train data 1.1994363701867812
log loss for cv data 1.2426840531063095
log loss for test data 1.2052375634639327
```

In [60]:

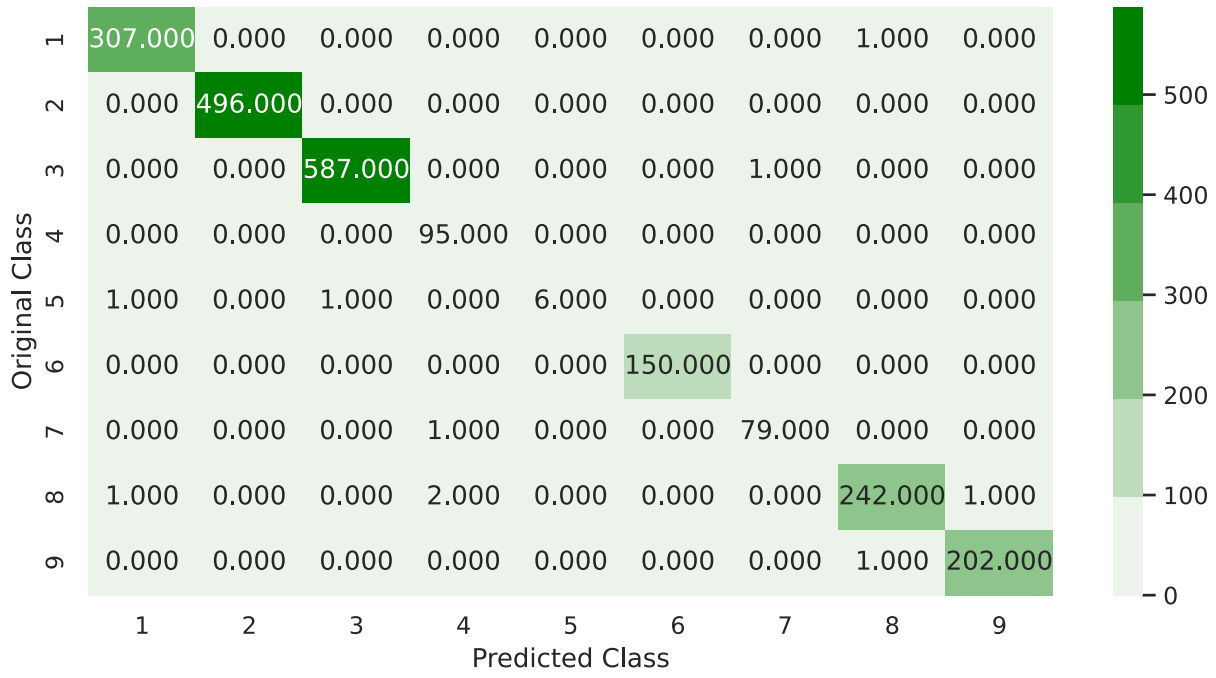
```
from sklearn.metrics import plot_confusion_matrix
y_predict_final = sig_clf.predict(x_test_final)
## Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/e
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=10
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamm
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppi
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE
# get_score(importance_type='weight') -> get the feature importance
# ----------------------
```

```
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-onlin
# ----------------------

alpha=[10,100,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i, eval_metric='mlogloss')
    x_cfl.fit(x_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(x_trn_final, y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=x_cfl.cl

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()
print (y_predict_final, y_test_final)
#plot_confusion_matrix(y_test_final, y_predict_final)
```

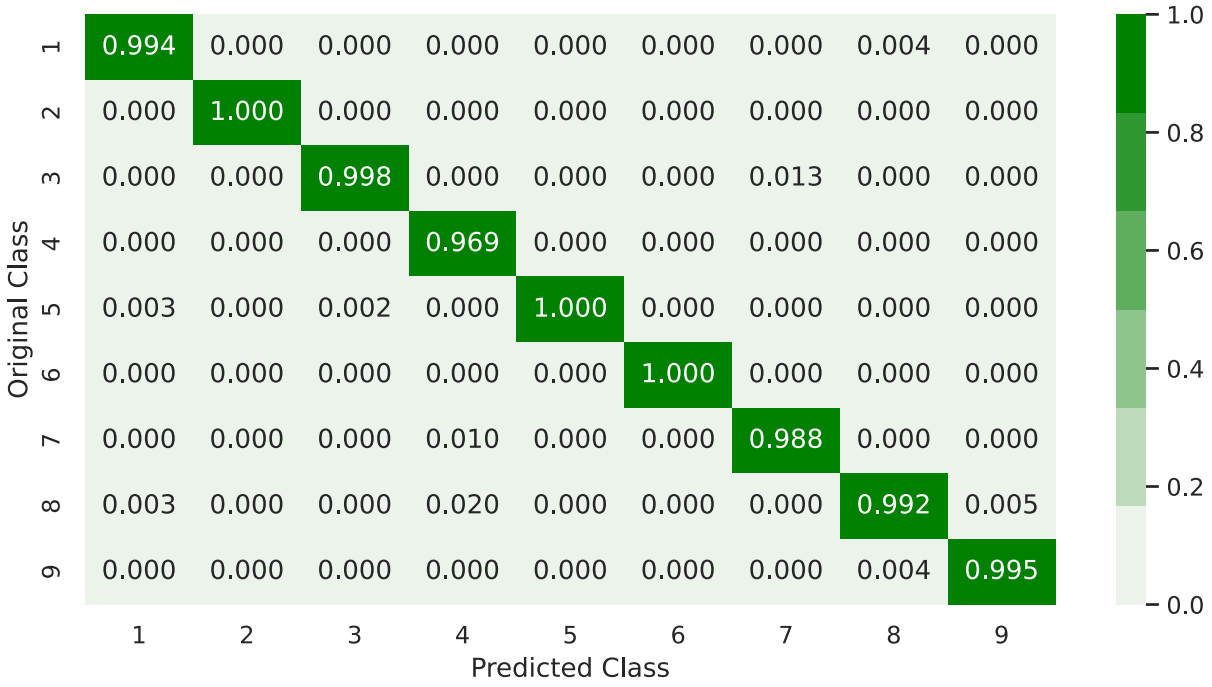
In [78]:

```
plot_confusion_matrix(y_test_final.to_numpy(), y_predict_final)
```

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

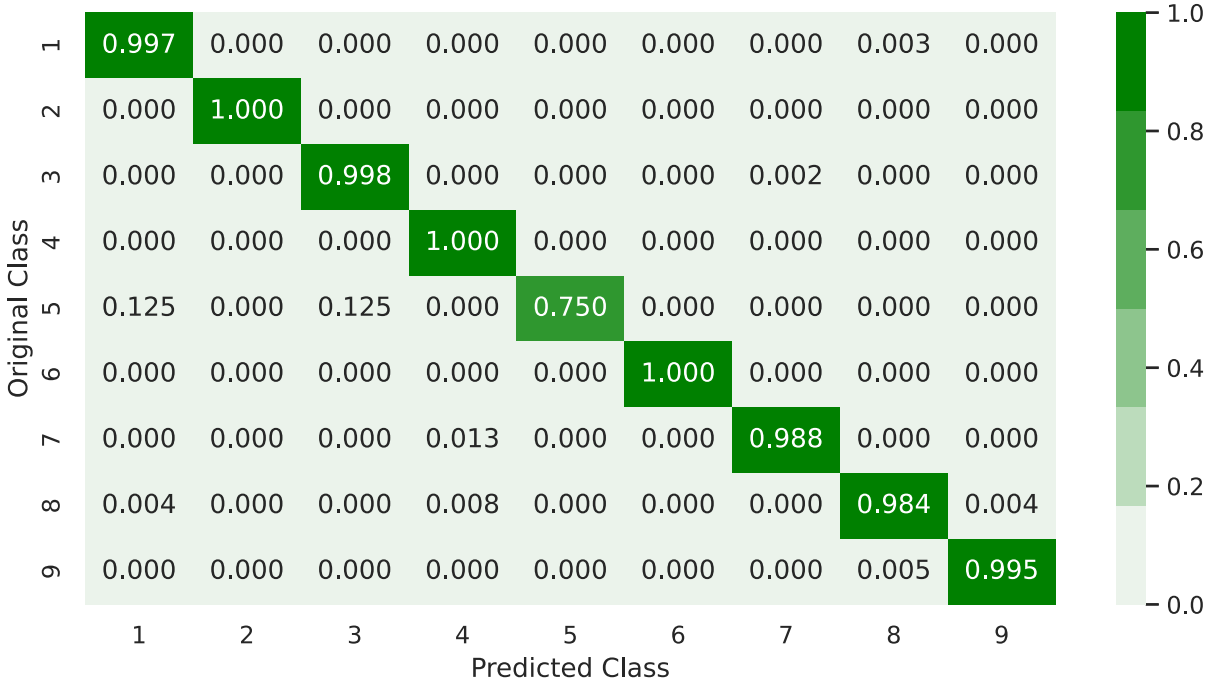



```
---------------------------------------------------- Precision matrix ----------
----------------------------------------
```



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.]

In [ ]:

```
x_cfl=XGBClassifier(n_estimators=2000, nthread=-1, eval_metric='mlogloss')
x_cfl.fit(x_trn_final,y_trn_final, verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_trn_final, y_trn_final)

predict_y = sig_clf.predict_proba(x_trn_final)
#alpha=[10,100,1000,2000]
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
```

For values of best alpha = 0.01 The train log loss is: 0.010187974436441512 For values

of best alpha = 0.01 The cross validation log loss is: 0.02395762856614576 For values of best alpha = 0.01 The test log loss is: 0.018309505637434106

Procedure: 1.First I took the byte file and made Exploratory Data Analysis.

2.used uni-gram count features and applied machine learning models.

3.preprocessed the asm file and extracted various segment count as features.

4.applied machine learning models on asm segment count.

5.combined byte features and asm segment features.

6.applied machine learning models on combined features.

7.extracted features like byte bigram,opcode bi gram and 200 pixels of asm image.

8.appplied machine learning models on combined features.

In [82]:
```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','log loss']
ptable.add_row(["random","Byte files","2.45"])
ptable.add_row(["knn","Byte files","0.48"])
ptable.add_row(["Logistic Regression","Byte files","0.52"])
ptable.add_row(["Random Forest Classifier ","Byte files","0.06"])
ptable.add_row(["XgBoost Classification","Byte files","0.07"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["knn","asmfiles","0.21"])
ptable.add_row(["Logistic Regression","asmfiles","0.38"])
ptable.add_row(["Random Forest Classifier ","asmfiles","0.03"])
ptable.add_row(["XgBoost Classification","asmfiles","0.04"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Random Forest Classifier ","Byte files+asmfiles","0.04"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles","0.02"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Logistic Regression","Byte files+asmfiles+advanced features"
ptable.add_row(["XgBoost Classification","Byte files+asmfiles+advanced featur
print(ptable)
```

```
+-------------------------------------------------------------------------
-+
|                              Model Comparision
|
+--------------------------+---------------------------------------+---------
-+
|           Model          |                Features               | log loss
|
+--------------------------+---------------------------------------+---------
-+
|           random         |              Byte files               |   2.45
|
|            knn           |              Byte files               |   0.48
|
|    Logistic Regression   |              Byte files               |   0.52
|
| Random Forest Classifier |              Byte files               |   0.06
|
|   XgBoost Classification |              Byte files               |   0.07
|
|                          |                                       |
```

knn	asmfiles	0.21
Logistic Regression	asmfiles	0.38
Random Forest Classifier	asmfiles	0.03
XgBoost Classification	asmfiles	0.04
Random Forest Classifier	Byte files+asmfiles	0.04
XgBoost Classification	Byte files+asmfiles	0.02
Logistic Regression	Byte files+asmfiles+advanced features	1.12
XgBoost Classification	Byte files+asmfiles+advanced features	0.01