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:nothin
g, 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    ??0exceptio
n@std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08    BB 42 00                       mov     dword ptr [e
si],    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 [e
cx],    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 [e
si],    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 giv
en a data point => Multi class classification problem
```

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning Objectives and Constraints

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

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

In [ ]:

```
! wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows N
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

3. Exploratory Data Analysis

In [62]:

```
import pandas as pd
import numpy  as np
from sklearn.manifold import TSNE
```

In [63]:

```
import matplotlib.pyplot as plt
```

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random
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
```

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
pip install lgboost
```

```
Collecting lgboost
Note: you may need to restart the kernel to use updated packages.

  ERROR: Could not find a version that satisfies the requirement lgboost (fr
om versions: none)
ERROR: No matching distribution found for lgboost
```

In [ ]:

```
pip install lgboost
```

In [ ]:

```
pip install xgboost
```

```
Collecting xgboost
  Using cached https://files.pythonhosted.org/packages/b1/11/cba4be5a737c643
1323b89b5ade818b3bbe1df6e8261c6c70221a767c5d9/xgboost-1.0.2-py3-none-win_amd
64.whl (https://files.pythonhosted.org/packages/b1/11/cba4be5a737c6431323b89
b5ade818b3bbe1df6e8261c6c70221a767c5d9/xgboost-1.0.2-py3-none-win_amd64.whl)
Requirement already satisfied: numpy in c:\users\thribhuvan\anaconda3\lib\si
te-packages (from xgboost) (1.16.5)
Requirement already satisfied: scipy in c:\users\thribhuvan\anaconda3\lib\si
te-packages (from xgboost) (1.3.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.0.2
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
#separating byte files and asm files

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

# we will check if the folder 'byteFiles' exists if it not there we will create a folder wi
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

# if we have folder called 'train' (train folder contains both .asm files and .bytes files)
# for every file that we have in our 'asmFiles' directory we check if it is ending with .by
# 'byteFiles' folder

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

3.1. Distribution of malware classes in whole data set

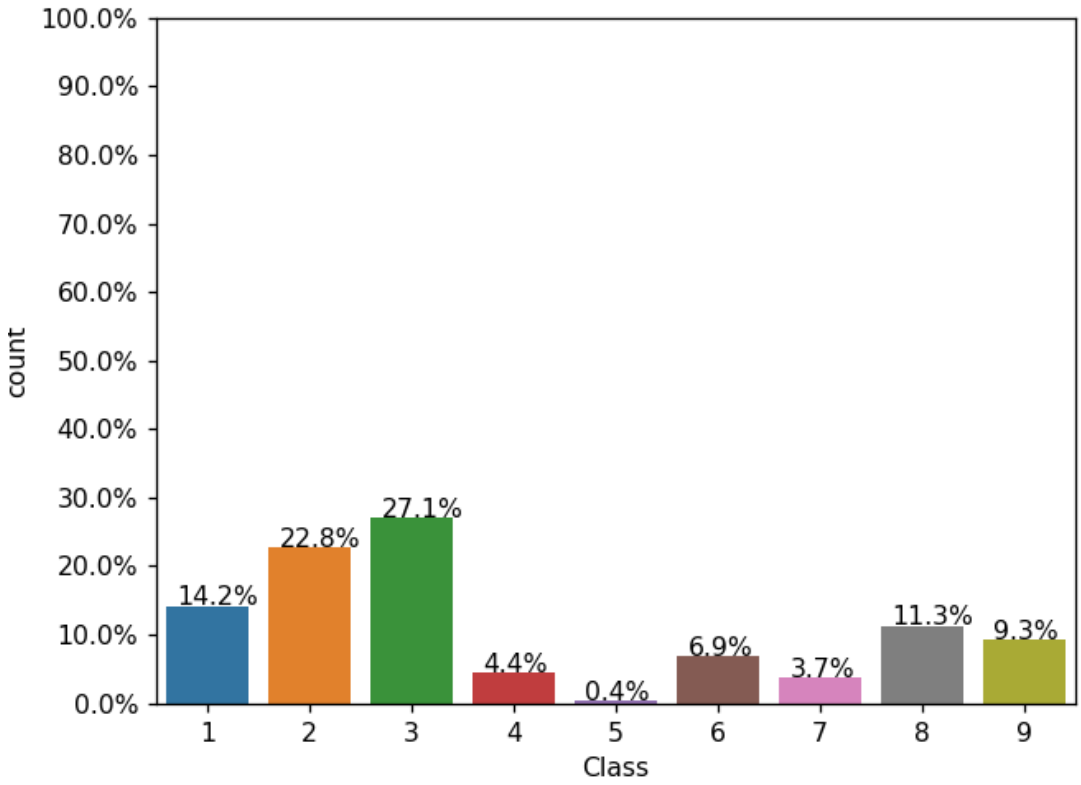
In [ ]:

```
Y=pd.read_csv("trainLabels.csv")
```

In [ ]:

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

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

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [ ]:

```
#file sizes of byte files

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

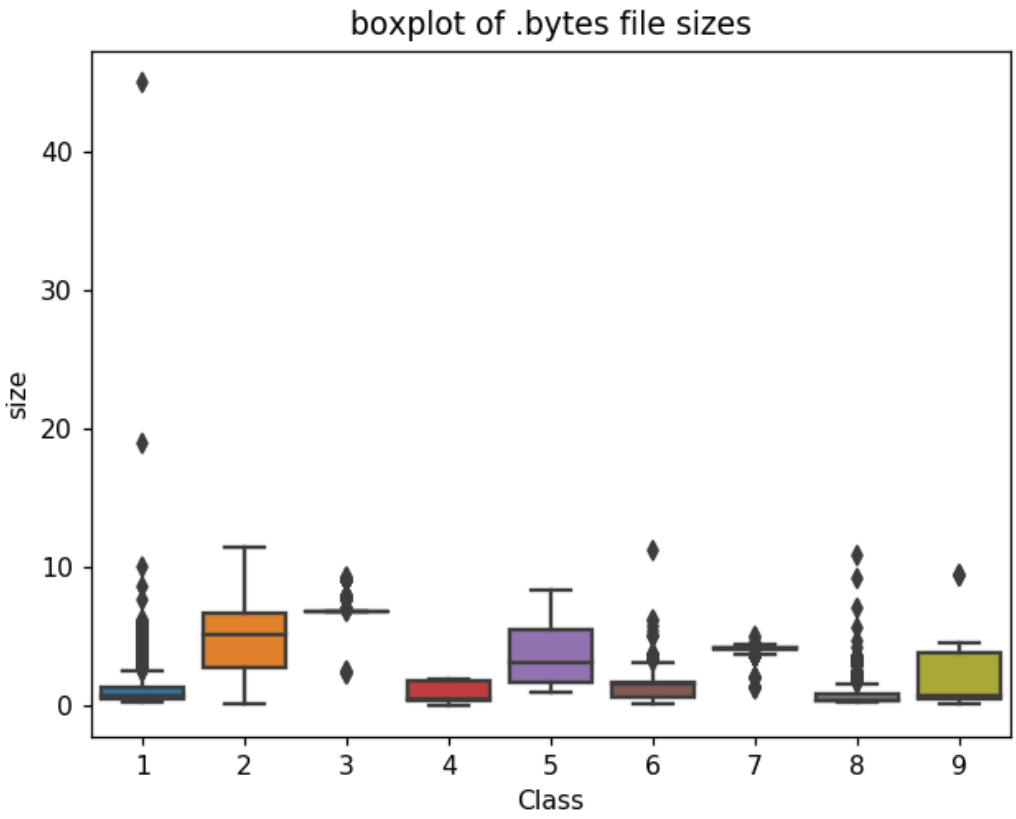
```
                     ID      size  Class
0  01azqd4InC7m9JpocGv5  4.234863      9
1  01IsoiSMh5gxyDYTl4CB  5.538818      2
2  01jsnpXSAlgw6aPeDxrU  3.887939      9
3  01kcPWA9K2BOxQeS5Rju  0.574219      1
4  01SuzwMJEIXsK7A8dQbl  0.370850      8
```

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

In [ ]:

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

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



3.2.3 feature extraction from byte files

In [ ]:

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

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,10,11,12,13,14,15,16,17,1
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i, row in enumerate(feature_matrix[k]):
        if i!=len(feature_matrix[k])-1:
            byte_feature_file.write(str(row)+",")
        else:
            byte_feature_file.write(str(row))
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [ ]:

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

Out[77]:

	ID	0	1	2	3	4	5	6	7	8	...	f7
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451

2 rows × 258 columns

In [ ]:

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

Out[76]:

	ID	size	Class
0	01azqd4InC7m9JpocGv5	4.234863	9
1	01IsoiSMh5gxyDYTl4CB	5.538818	2

In [ ]:

```
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[78]:

	ID	0	1	2	3	4	5	6	7	8	...	f9
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439

2 rows × 260 columns

In [41]:

```
byte_features_with_size=pd.read_csv("result_with_size.csv")
```

In [42]:

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

In [ ]:

```
result.head(2)
```

Out[4]:

	Unnamed: 0	ID	0	1	2	3	4	
0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.0018
1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.0038

2 rows × 261 columns

In [ ]:

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

In [ ]:

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

Out[64]:

	Unnamed: 0	Unnamed: 0.1	ID	0	1	2	3	
0	0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.0(
1	1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.0(
2	2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.0(
3	3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.0(
4	4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.0(

5 rows × 262 columns

3.2.4 Multivariate Analysis

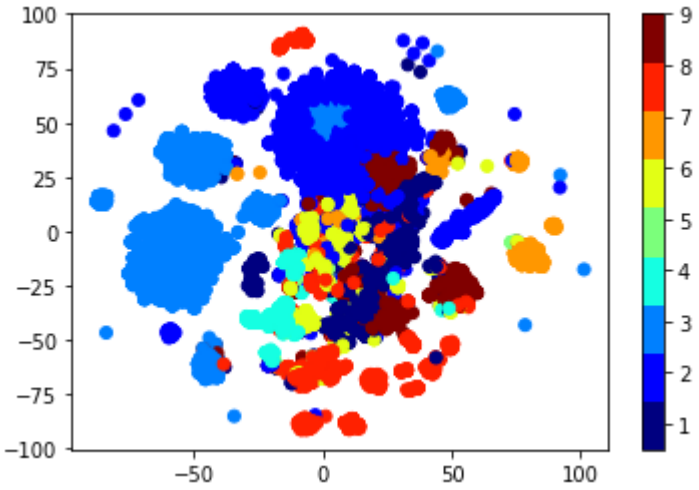
In [ ]:

```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class','Unnamed: 0','Unnamed: 0.1'], axis=1)
vis_x = results[:, 0]
vis_y = results[:, 1]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-66-d196fcab46fa> in <module>
      5 vis_x = results[:, 0]
      6 vis_y = results[:, 1]
----> 7 plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
      8 plt.colorbar(ticks=range(10))
      9 plt.clim(0.5, 9)

NameError: name 'plt' is not defined
```

In [ ]:

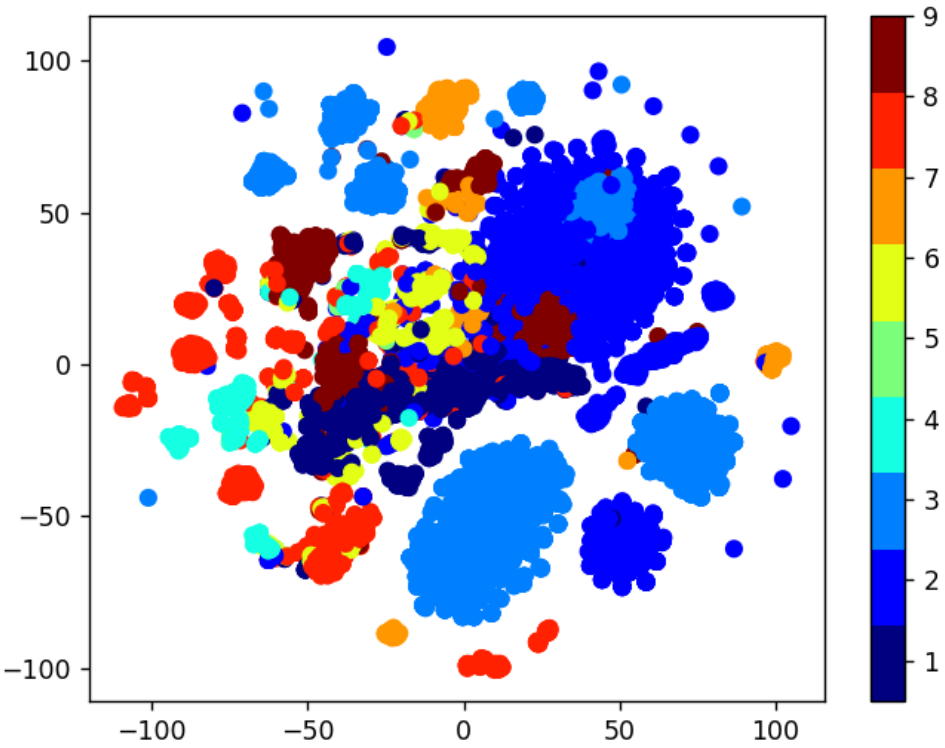
```
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



In [ ]:

```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.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()
```

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



In [ ]:

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

Train Test split

In [ ]:

```
result.head()
```

Out[52]:

	Unnamed: 0	Unnamed: 0.1	ID	0	1	2	3	
0	0	0.000000	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.0(
1	1	0.000092	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.0(
2	2	0.000184	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.0(
3	3	0.000276	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.0(
4	4	0.000368	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.0(

5 rows × 262 columns

In [ ]:

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class','Unnamed: 0',
# split the train data into train and cross validation by maintaining same distribution of
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

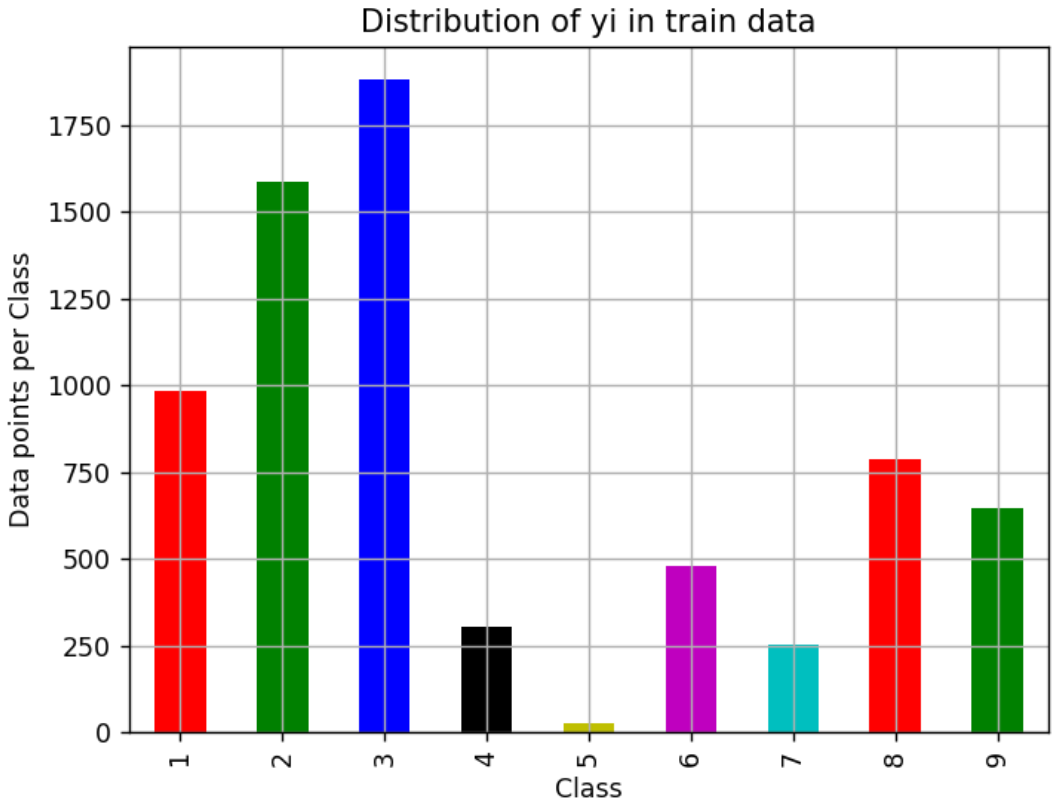
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
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
```

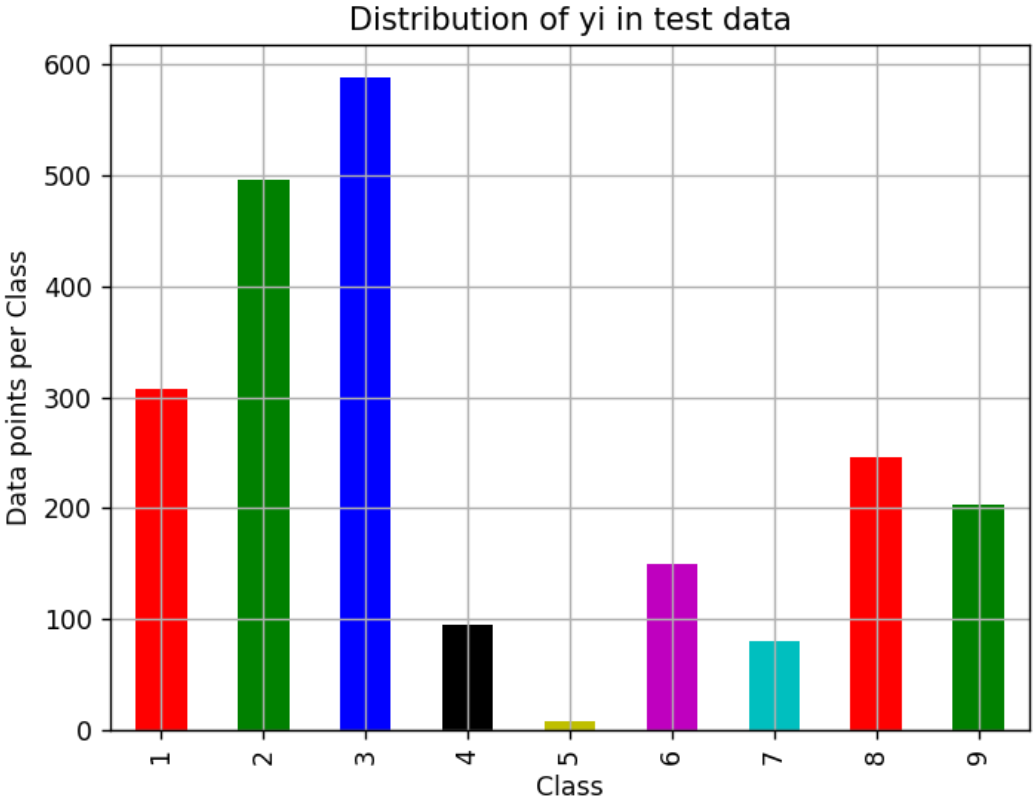
In [ ]:

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

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

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


print('-'*80)
my_colors = 'rgbkymc'
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.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '('

print('-'*80)
my_colors = 'rgbkymc'
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.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(',
```

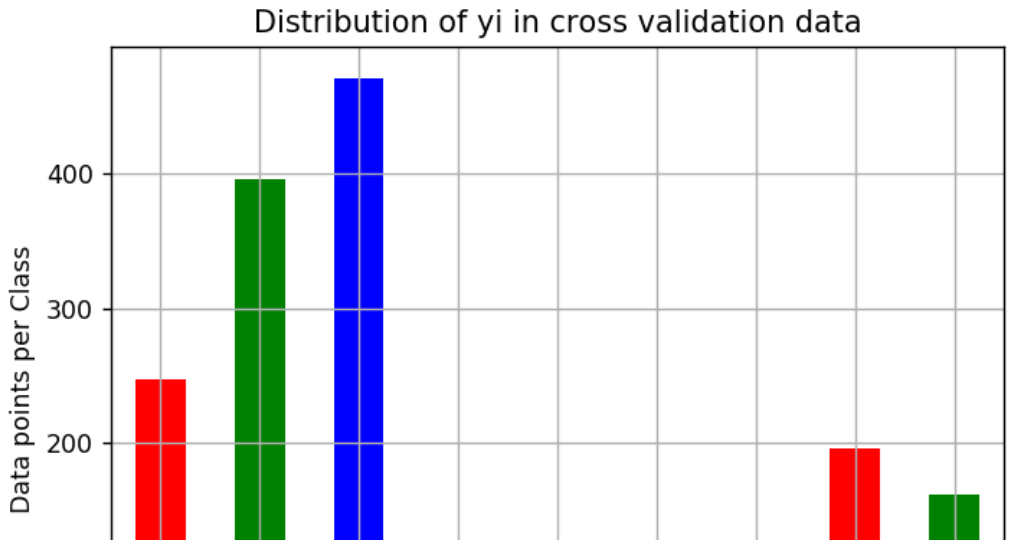
<IPython.core.display.Javascript object>




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

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




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

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




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

In [8]:

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

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

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

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

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

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

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

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

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

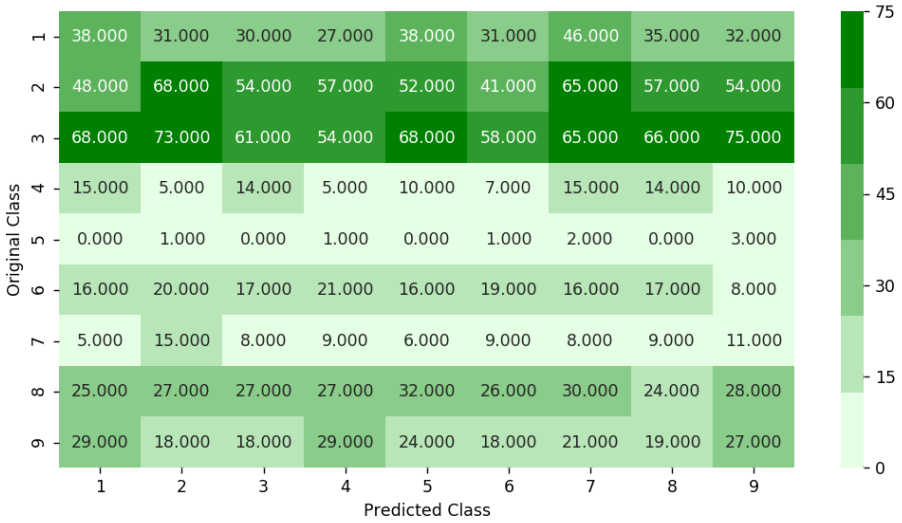
4.1.1. Random Model

In [ ]:

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

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

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


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

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

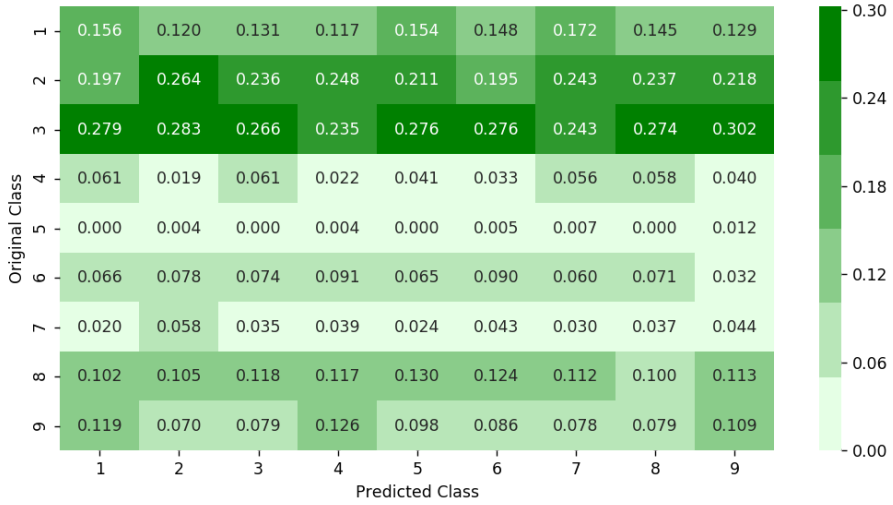
```
Log loss on Cross Validation Data using Random Model 2.45615644965
Log loss on Test Data using Random Model 2.48503905509
Number of misclassified points  88.5004599816
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

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



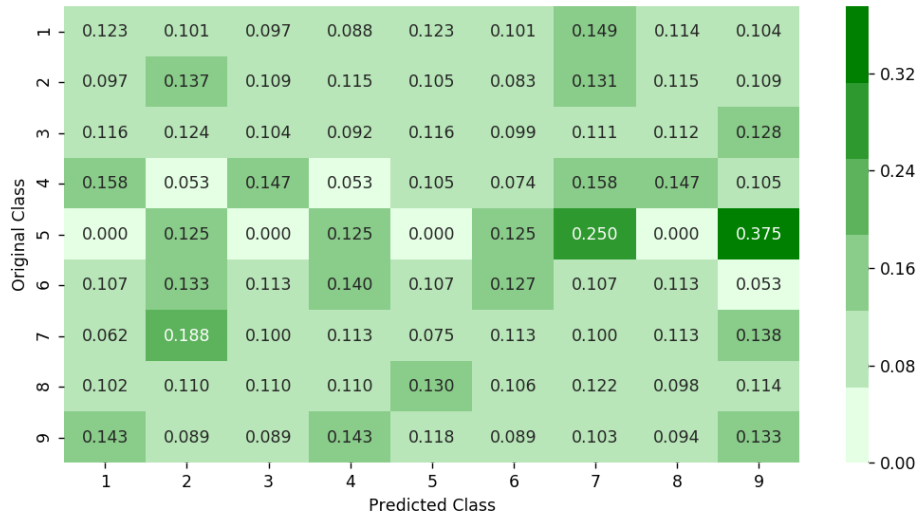

```
---------------------------------------------------- Precision matrix --------
------------------------------------------
```

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




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

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




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

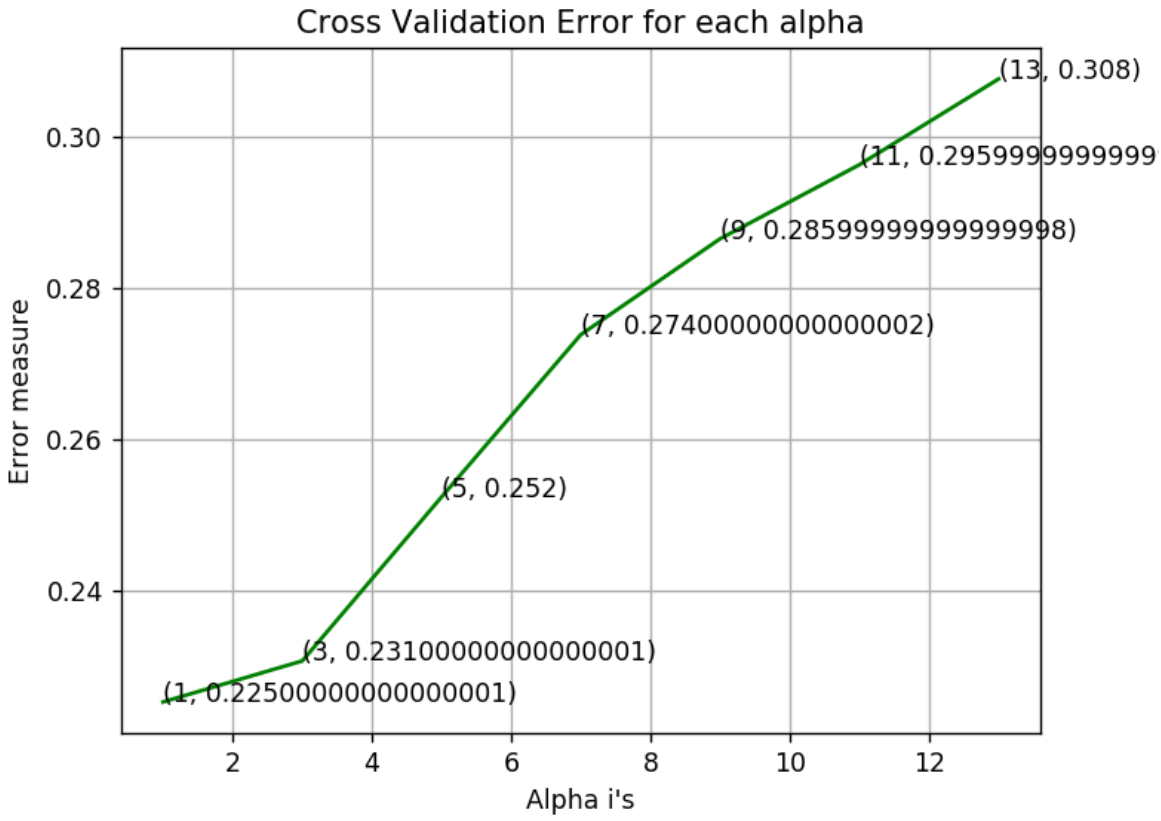
4.1.2. K Nearest Neighbour Classification

In [ ]:

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

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


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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

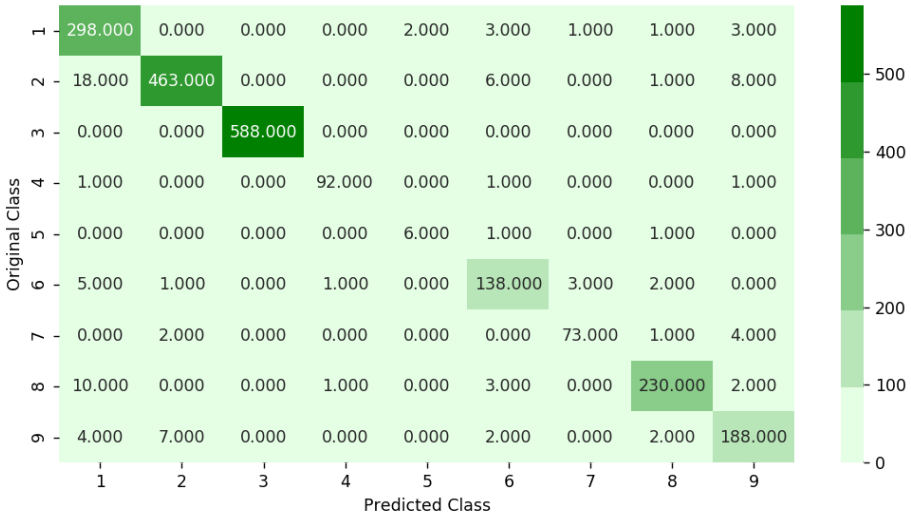
```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154

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




```
For values of best alpha =  1 The train log loss is: 0.0782947669247
```

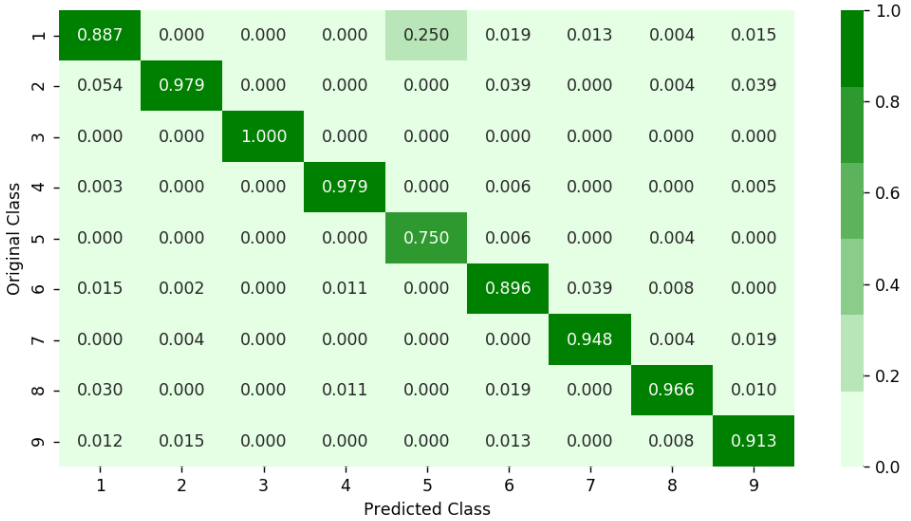
```
For values of best alpha =  1 The cross validation log loss is: 0.22538623
7304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```

<IPython.core.display.Javascript object>



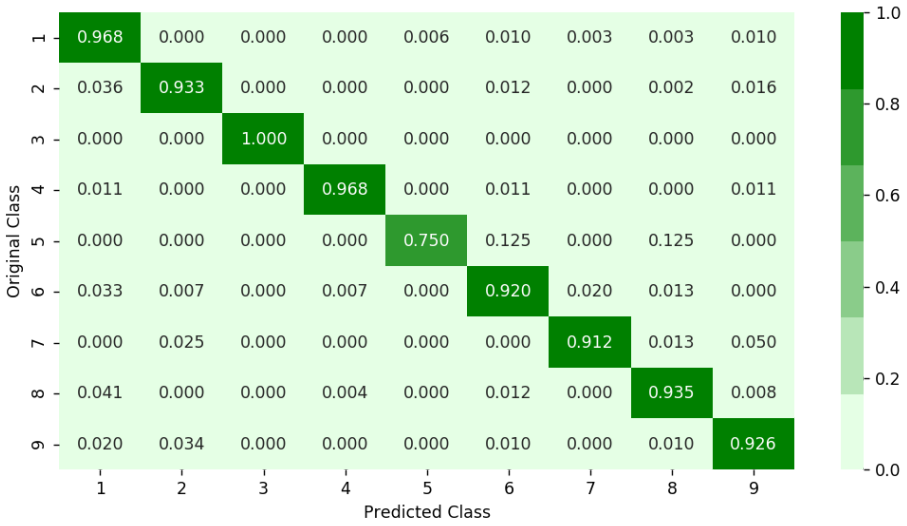

```
---------------------------------------------------- Precision matrix --------
-----------------------------------------
```

<IPython.core.display.Javascript object>



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

<IPython.core.display.Javascript object>




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

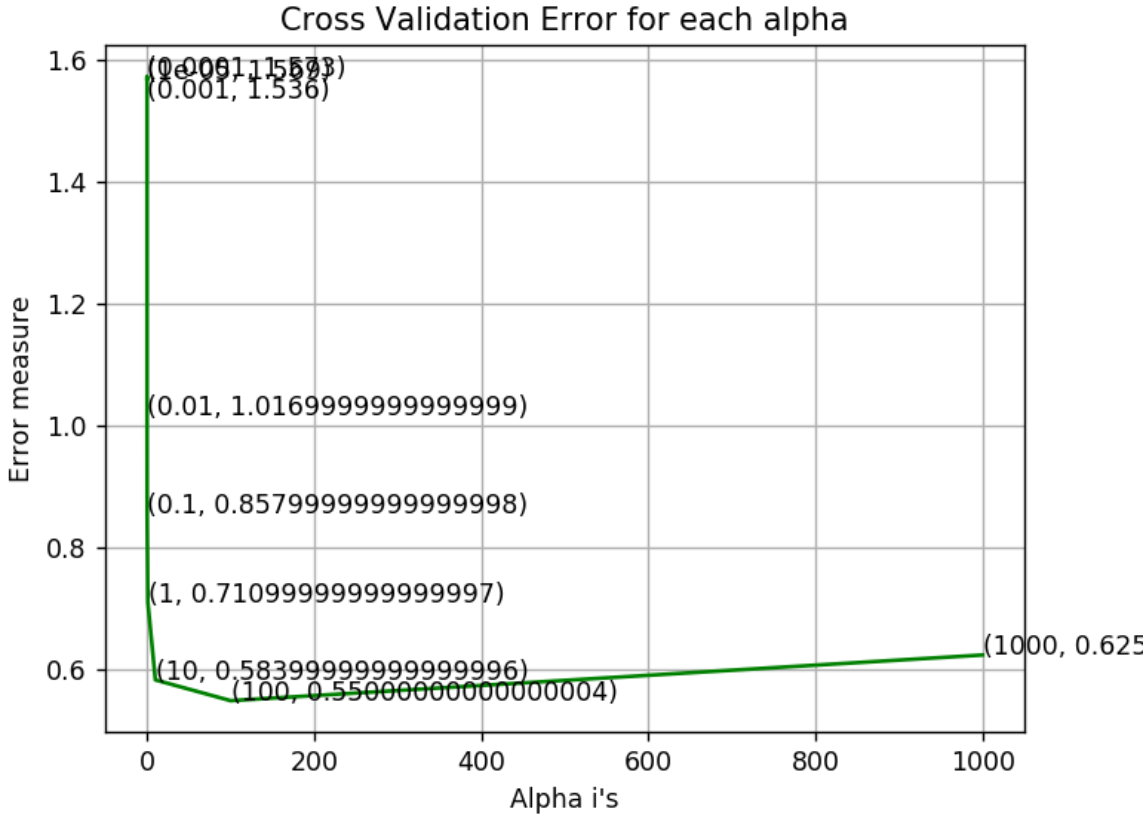
4.1.3. Logistic Regression

In [ ]:

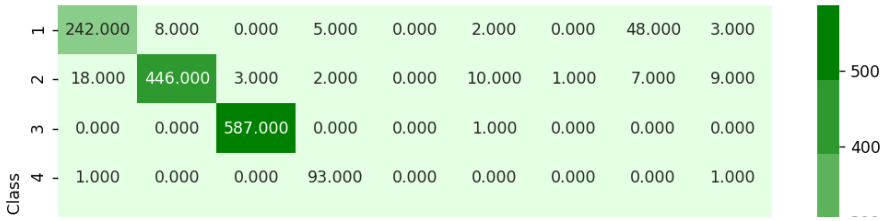
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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

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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
```

```
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =  100 is 0.549929846589
log_loss for c =  1000 is 0.624746769121

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

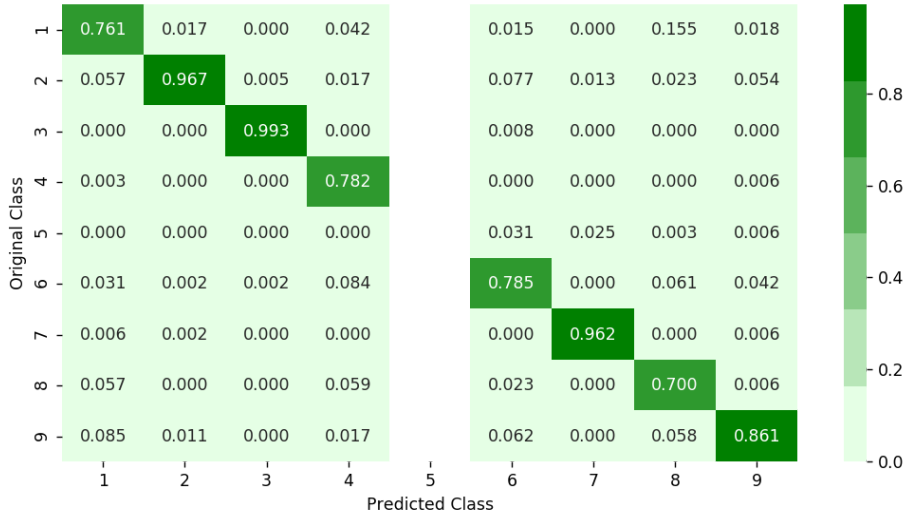



```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
---------------------------------------------------- Confusion matrix --------
-----------------------------------------

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



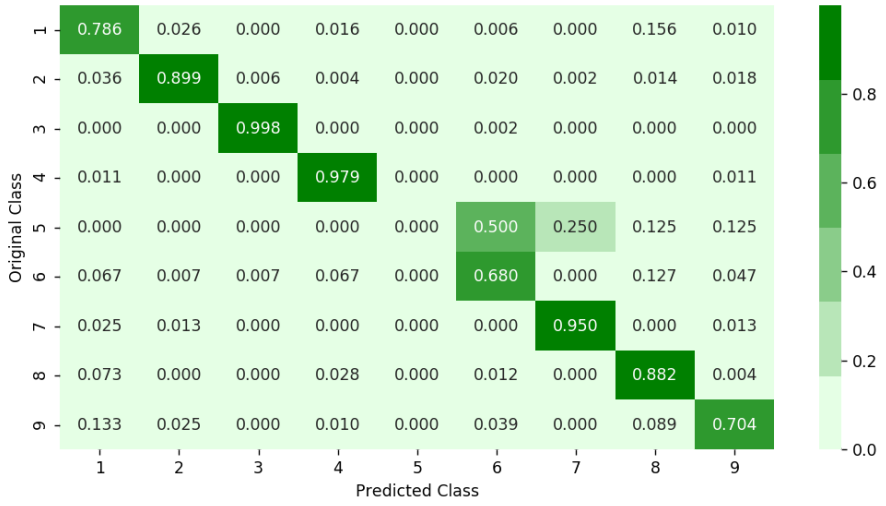
---------- Precision matrix --------

<IPython.core.display.Javascript object>



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

<IPython.core.display.Javascript object>




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

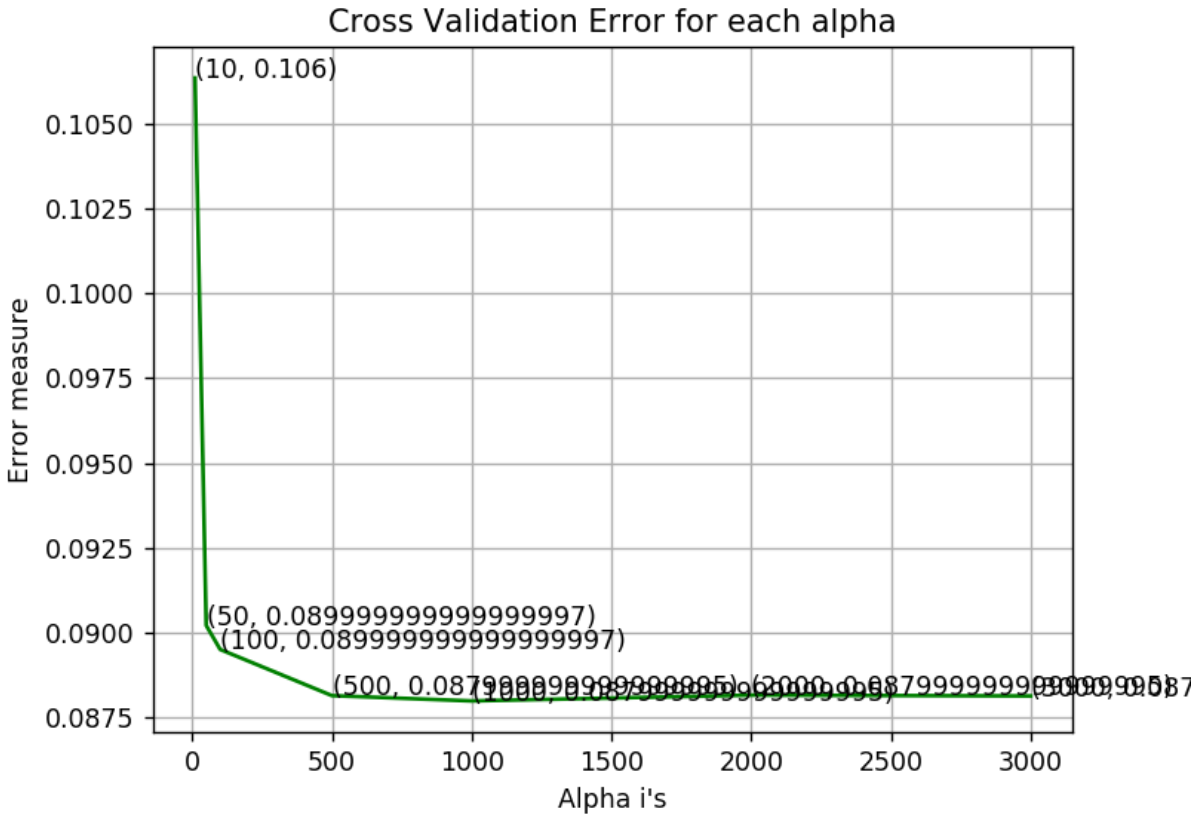
4.1.4. Random Forest Classifier

In [ ]:

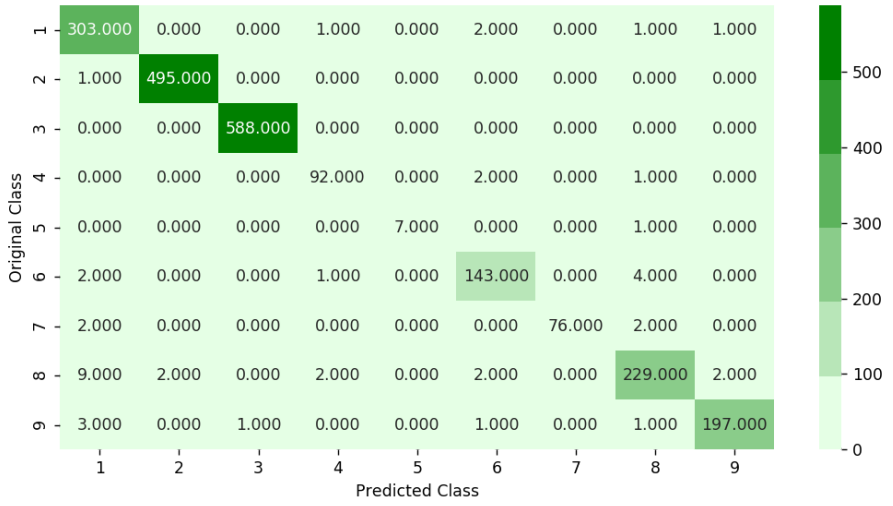
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=Non
# min_impurity_split=None, bootstrap=True, oob_score=False,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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.106357709164
log_loss for c =  50 is 0.0902124124145
log_loss for c =  100 is 0.0895043339776
log_loss for c =  500 is 0.0881420869288
log_loss for c =  1000 is 0.0879849524621
log_loss for c =  2000 is 0.0881566647295
log_loss for c =  3000 is 0.0881318948443

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

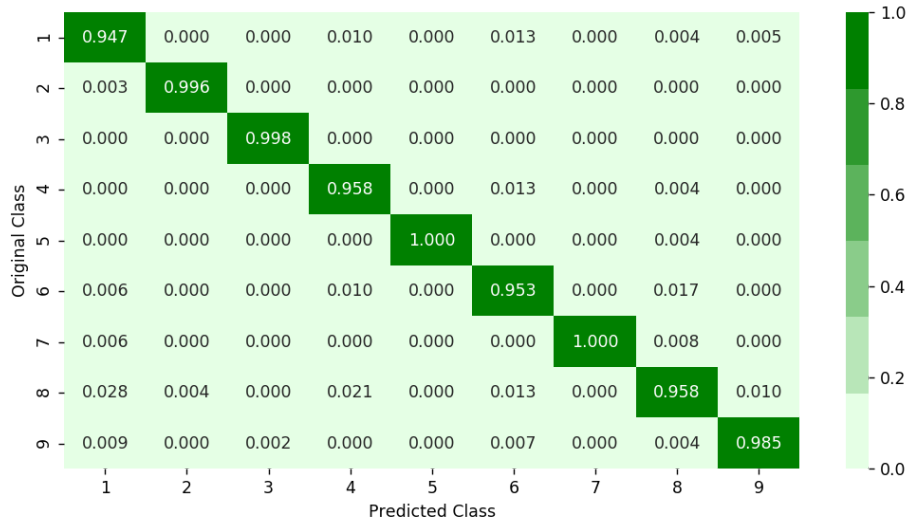



```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.0879849
524621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix --------
----------------------------------------

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



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

<IPython.core.display.Javascript object>



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

<IPython.core.display.Javascript object>

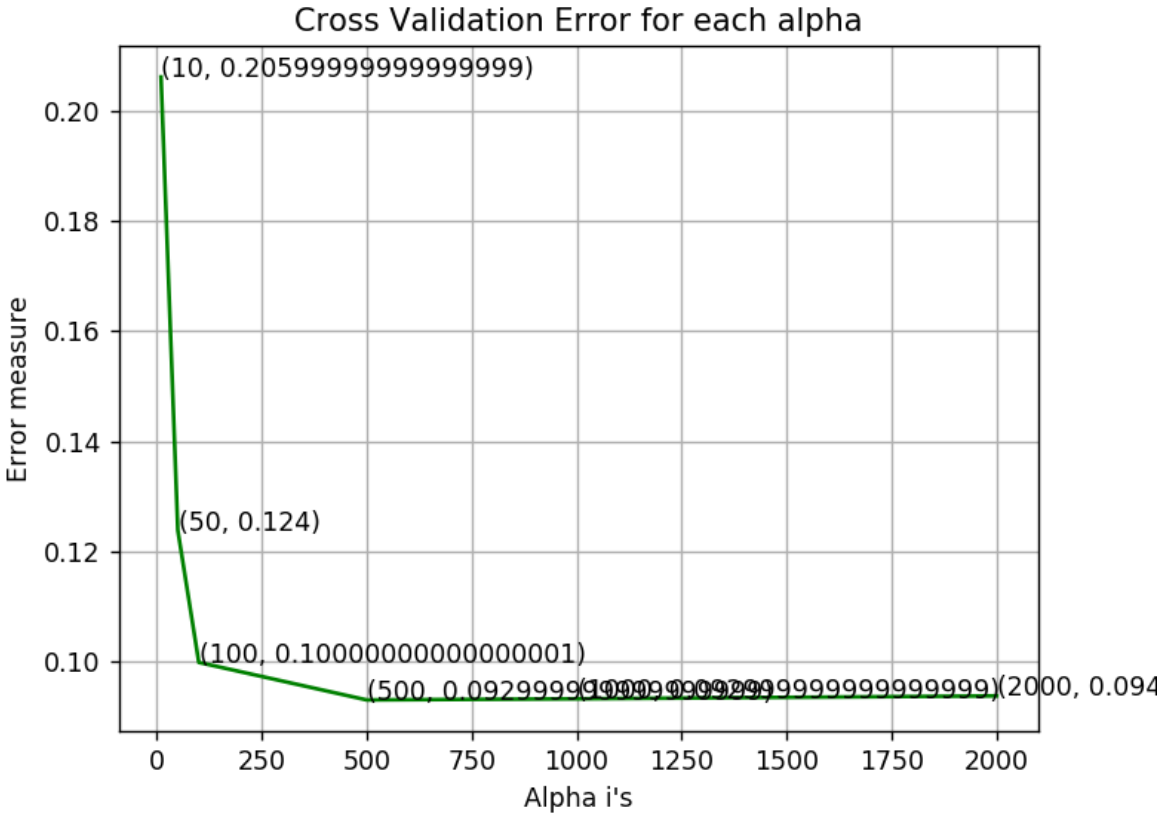
```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.5. XgBoost Classification

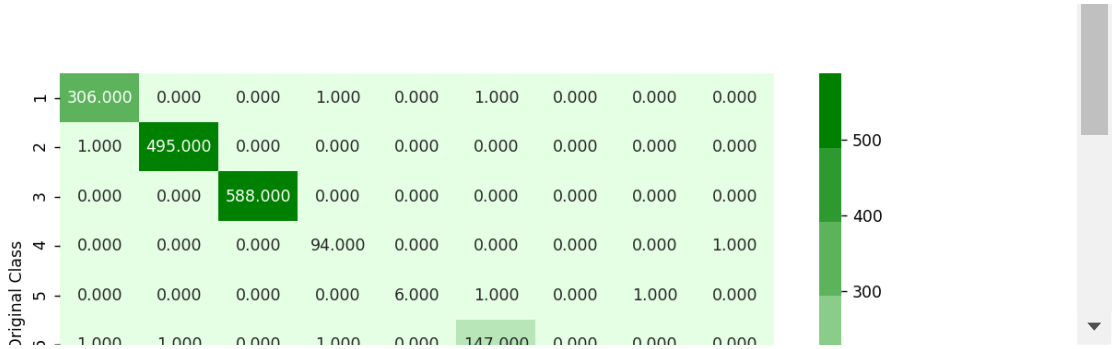
In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

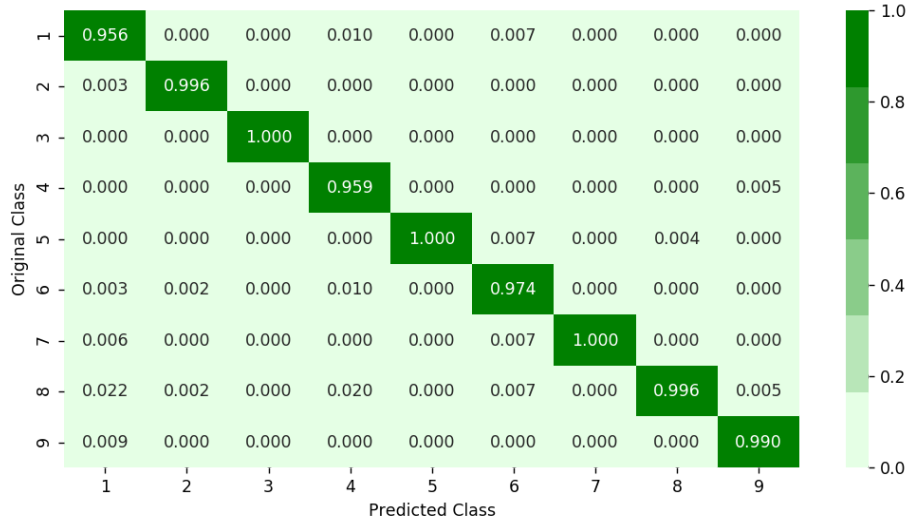



```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.09310356
81289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
---------------------------------------------------- Confusion matrix --------
----------------------------------------

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



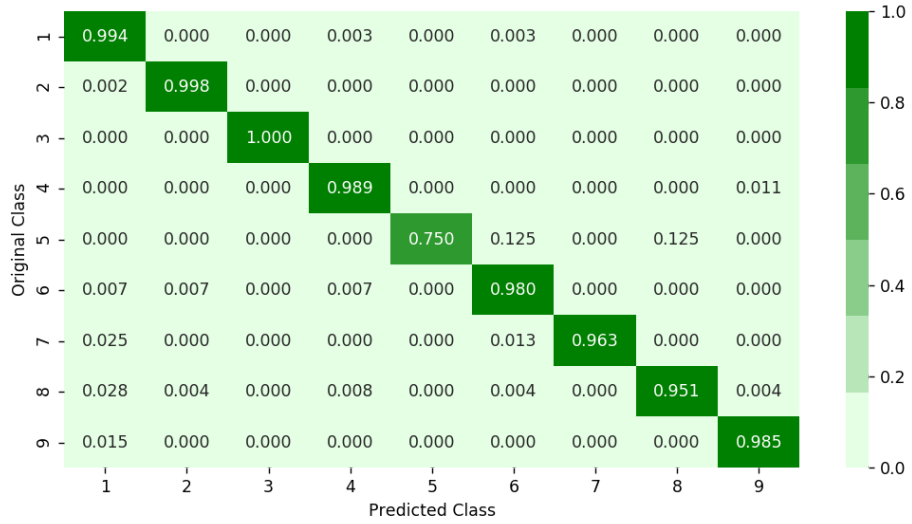
---------- Precision matrix --------

<IPython.core.display.Javascript object>



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

<IPython.core.display.Javascript object>



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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [ ]:

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

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

```
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.
4min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 10.1min remaining:  3.
1min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.
6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 14.2min finished
```

Out[75]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsa
mple_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, 0.1
5, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 1
0], 'colsample_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/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, max_depth=3)
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 [ ]:

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

source='train/'
files = os.listdir('train')
ID=df['Id'].tolist()
data=range(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'first')
    elif i%5==1:
        shutil.move(source+files[data[i]],'second')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'thrid')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fifth')
```

In [ ]:

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

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

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

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

```
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


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

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


def fourthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'de
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
```

```
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


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

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


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

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

In [ ]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the extract
# 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:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 53 columns

In [ ]:

```
len(Y)
```

Out[13]:

```
10868
```

In [ ]:

```
dfasm.head()
```

Out[4]:

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

5 rows × 52 columns

4.2.1.1 Files sizes of each .asm file

In [171]:

```
result_asm=pd.read_csv('result_asm.csv')
```

In [172]:

```
result_asm.head()
```

Out[172]:

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

5 rows × 54 columns

In [75]:

```
#file sizes of byte files

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

```
                     ID       size  Class
0  8VU3wPKq2lBY5bvnHQMS  12.485002      2
1  ErYMOiwT8h3yqvfkm9Pu   1.660654      4
2  dKNk2oDiqevnRTsgCmG0  10.842202      8
3  dFUABXPZzn3mL06OEjek   8.723547      9
4  EuH31db0IYoVintDaL9v   1.852554      1
```

In [210]:

```
final_feat[['asm_size','Class']].isnan()
```

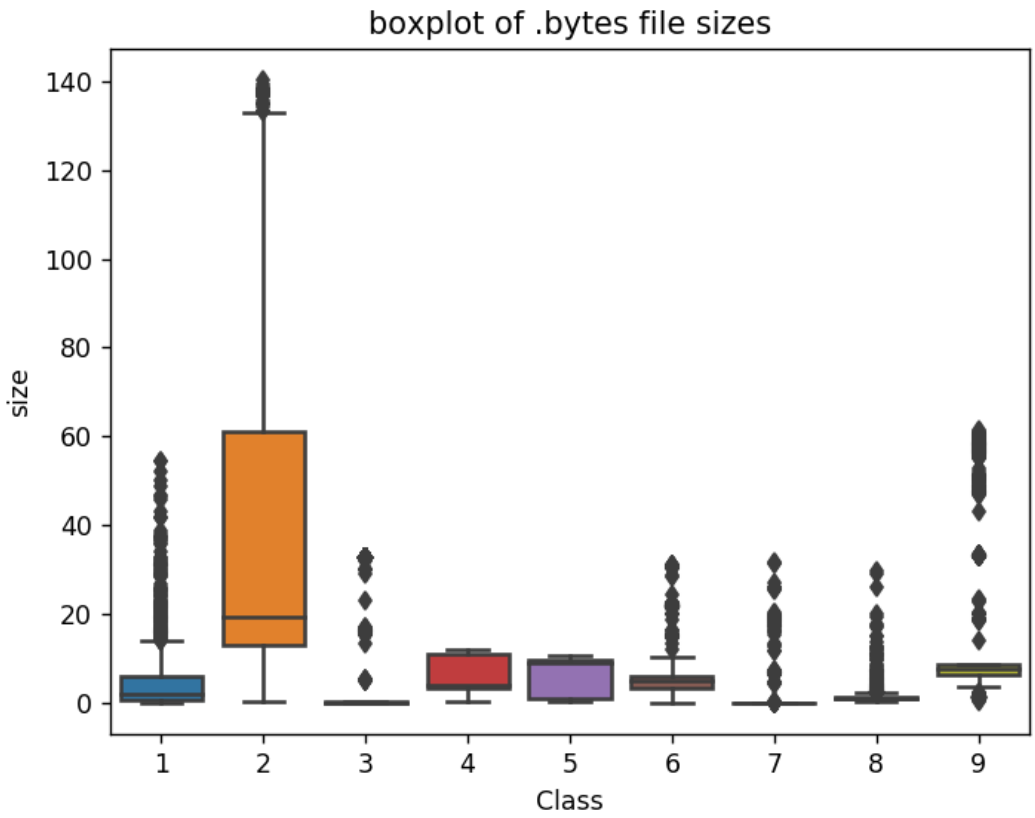
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-210-3d5c31f2b046> in <module>
----> 1 final_feat[['asm_size','Class']].isnan()

/opt/conda/lib/python3.7/site-packages/pandas/core/generic.py in __getattr__
(self, name)
   5272             if self._info_axis._can_hold_identifiers_and_holds_name(
name):
   5273                 return self[name]
-> 5274             return object.__getattribute__(self, name)
   5275
   5276     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'isnan'
```

4.2.1.2 Distribution of .asm file sizes

In [ ]:

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

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



In [ ]:

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

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

Out[140]:

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

5 rows × 54 columns

In [ ]:

```
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

Out[145]:

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

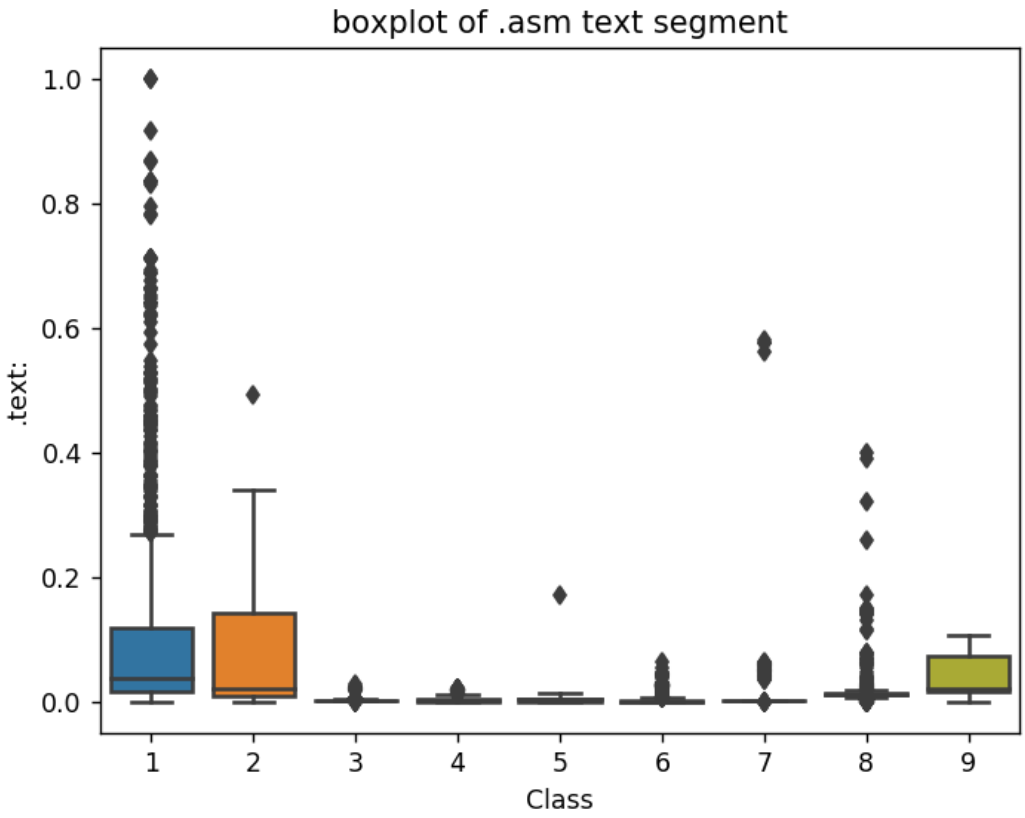
5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [ ]:

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

<IPython.core.display.Javascript object>

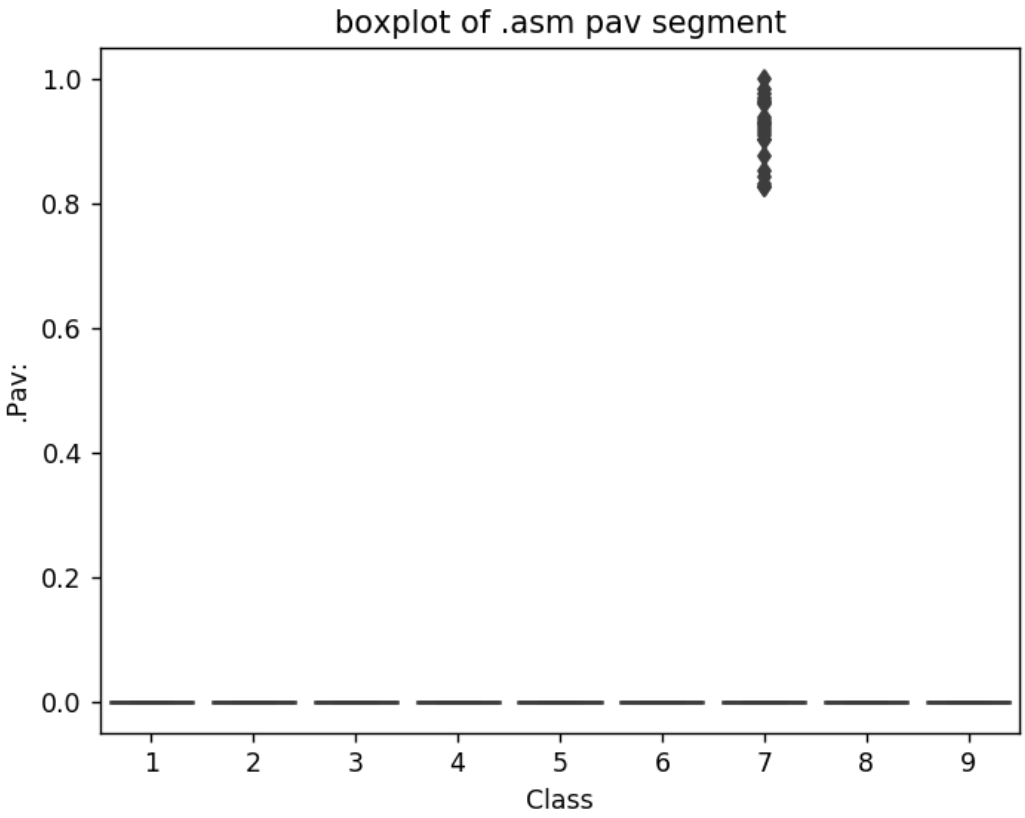


The plot is between Text and class
Class 1,2 and 9 can be easly separated

In [ ]:

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

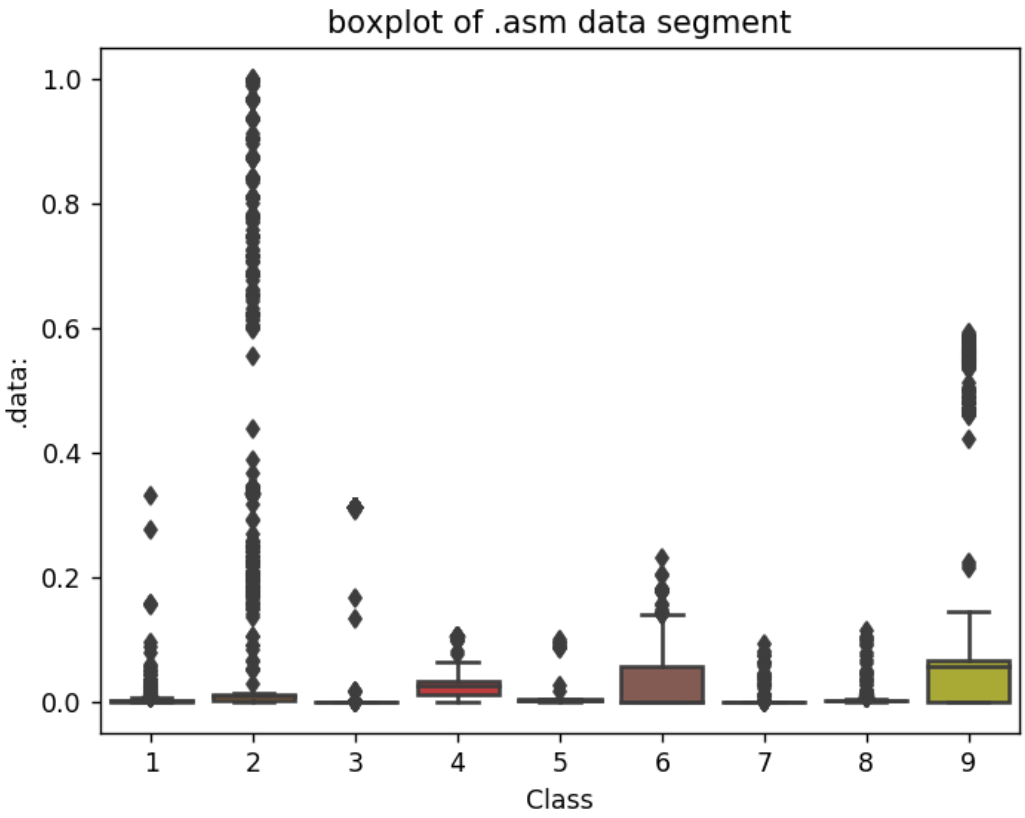
```
<IPython.core.display.Javascript object>
```



In [ ]:

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

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

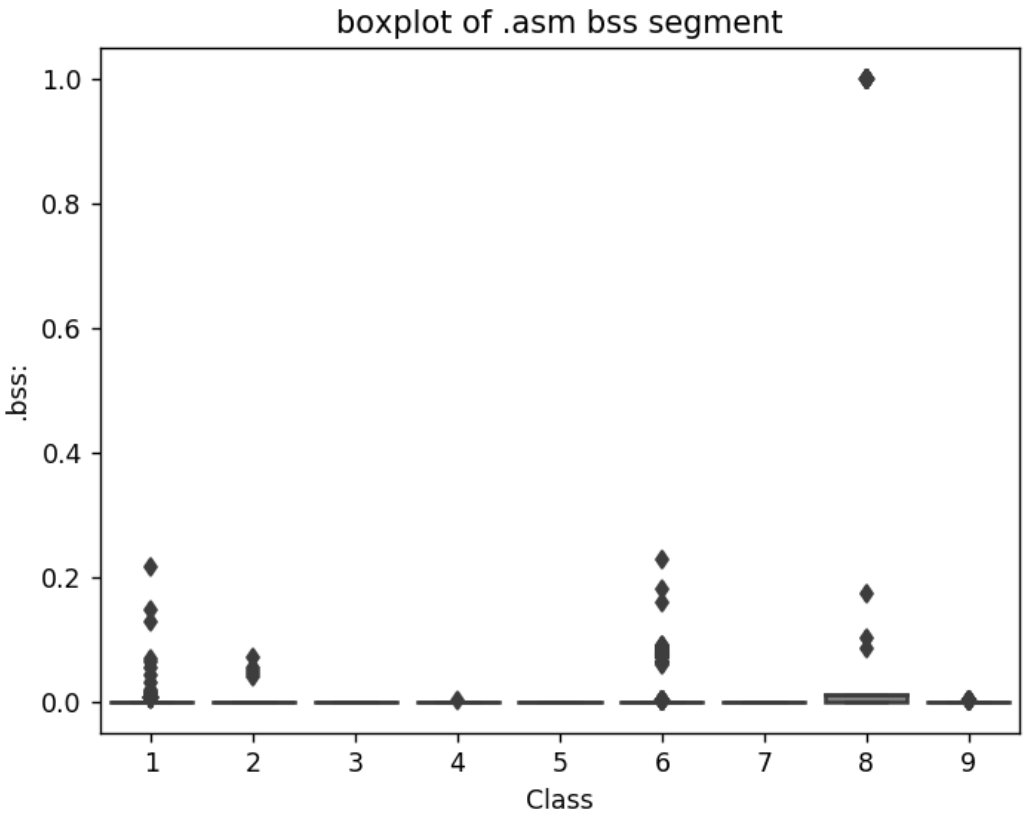


The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [ ]:

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

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

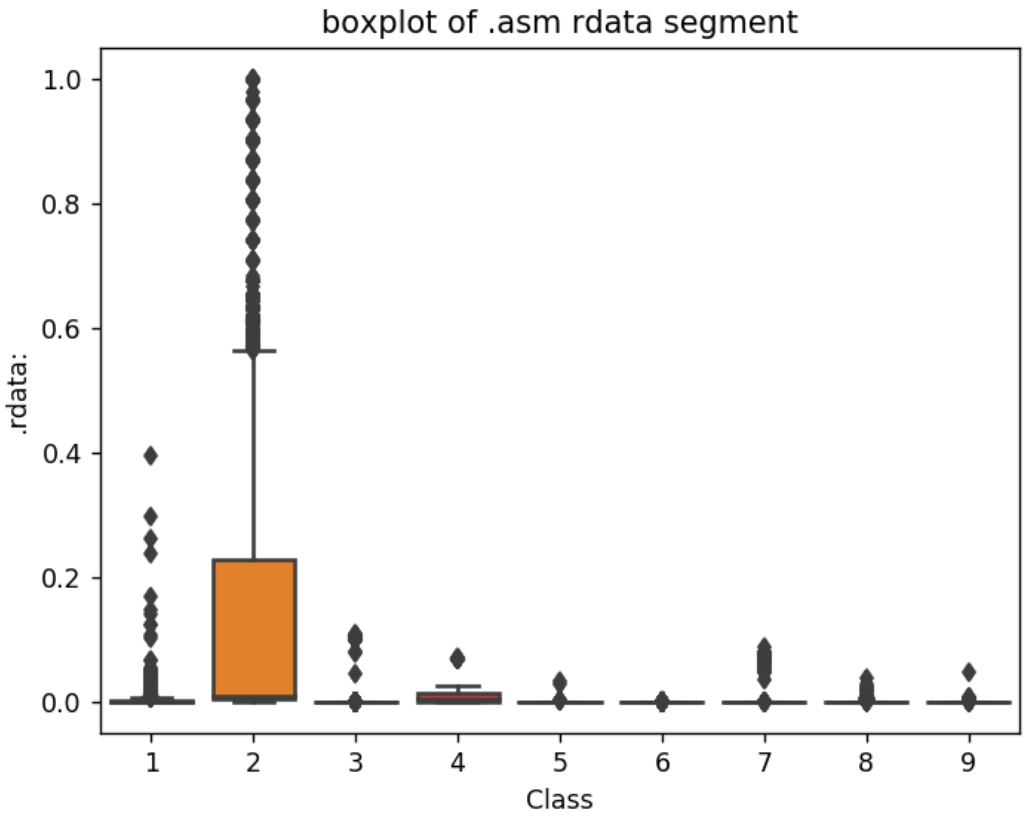


plot between bss segment and class label
very less number of files are having bss segment

In [ ]:

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

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



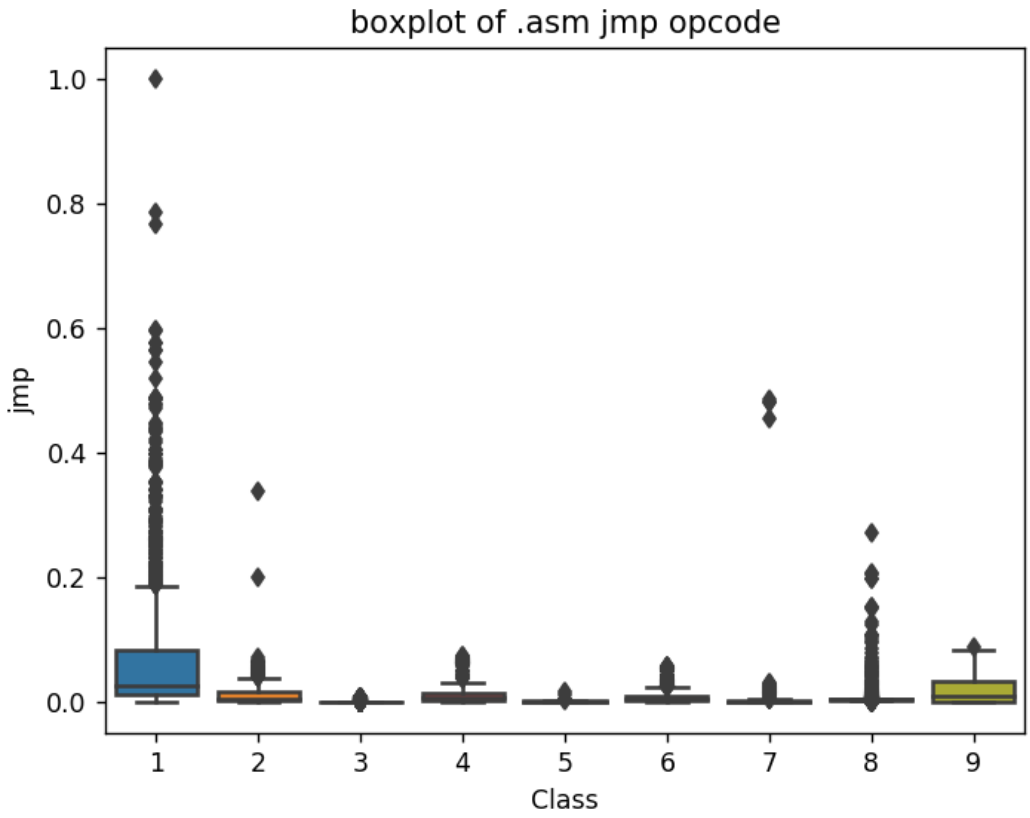
Plot between rdata segment and Class segment

Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [ ]:

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

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

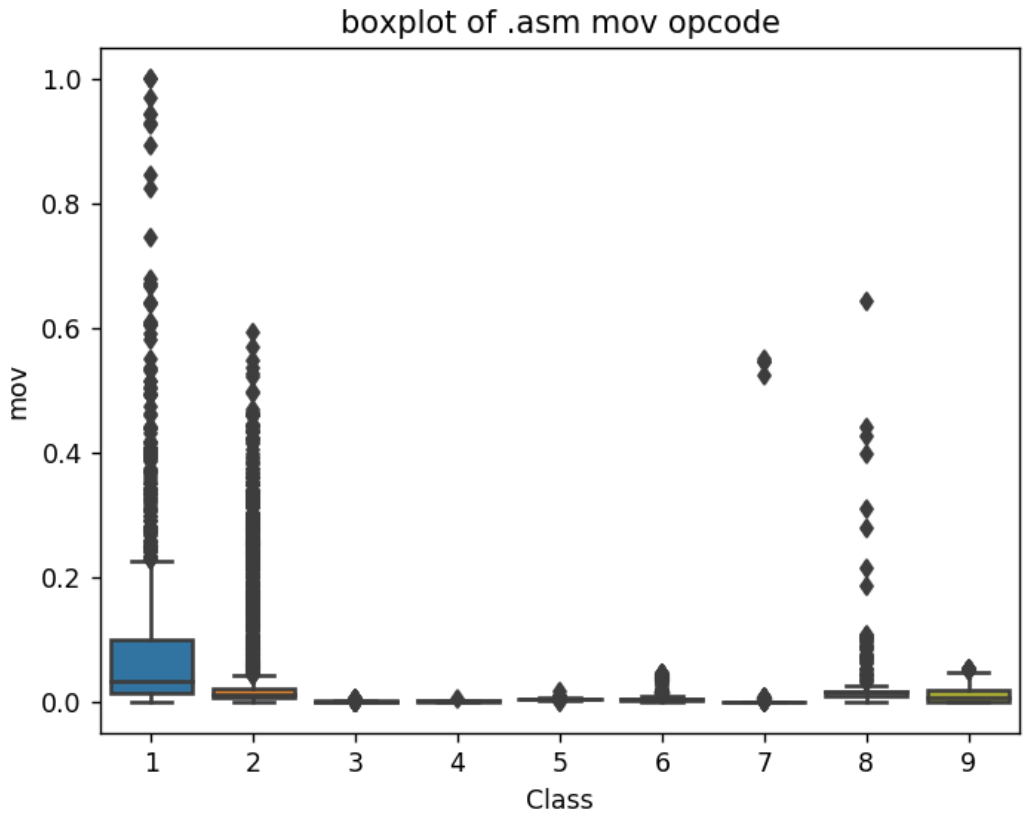


plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [ ]:

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

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

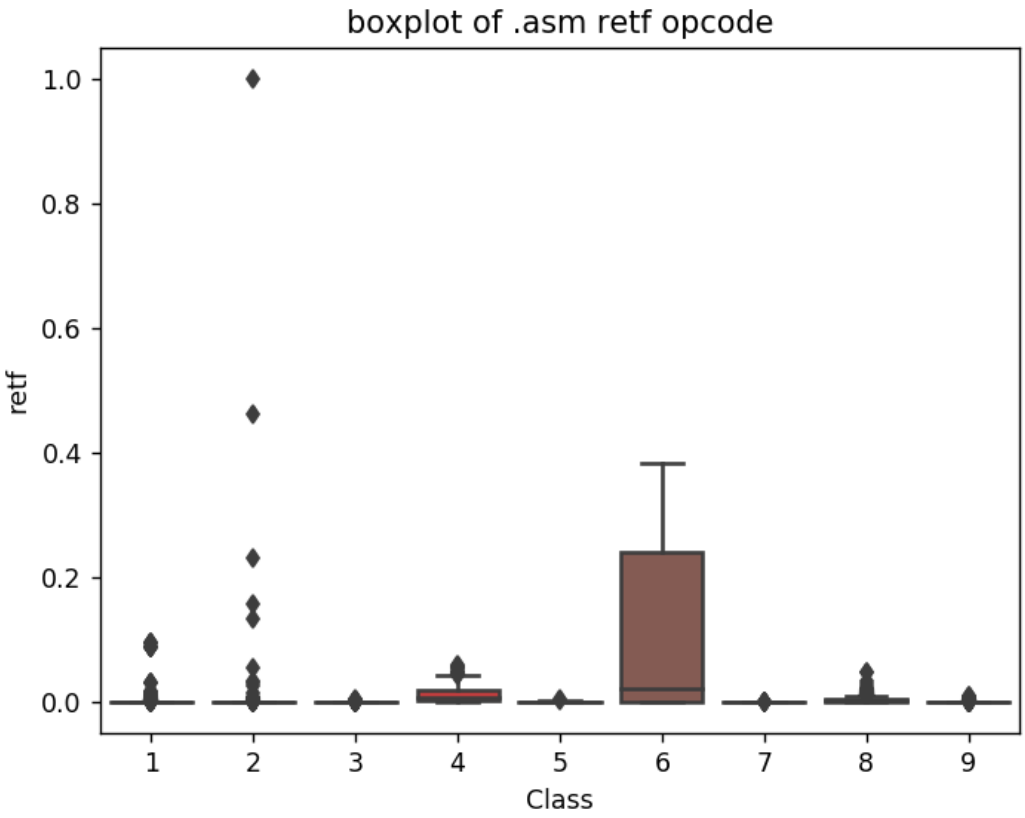


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [ ]:

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

<IPython.core.display.Javascript object>

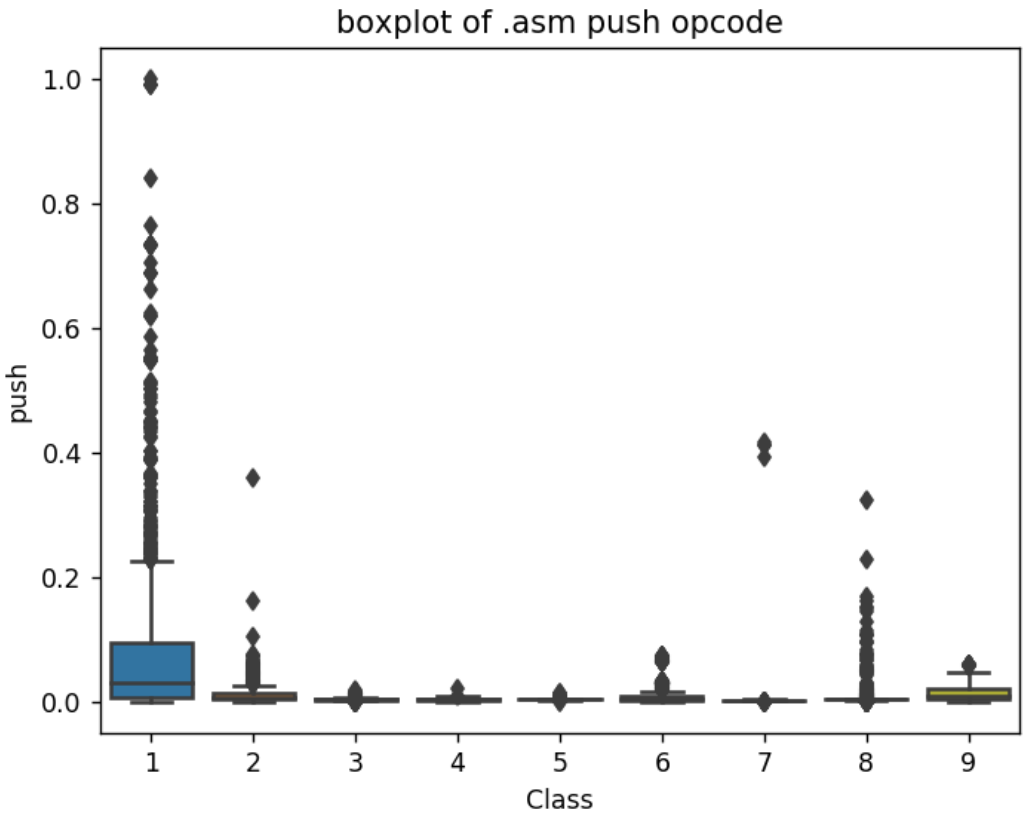


plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [ ]:

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

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



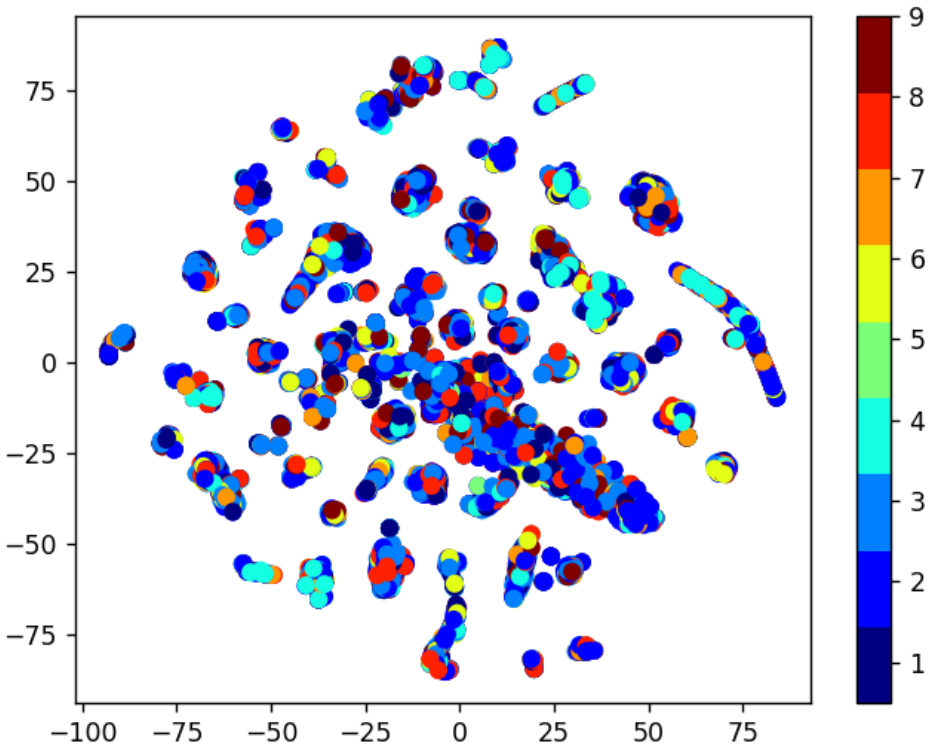
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

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-distributed-sto

#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()
```

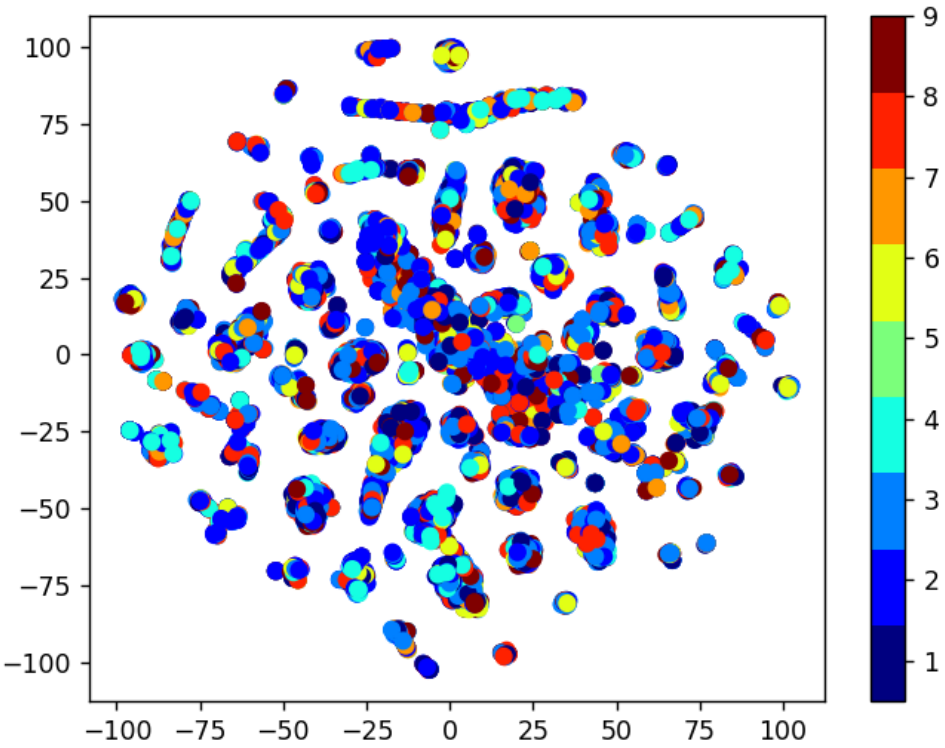
<IPython.core.display.Javascript object>



In [ ]:

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

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

<IPython.core.display.Javascript object>



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [ ]:

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

In [ ]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=a
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,st
```

In [ ]:

```
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

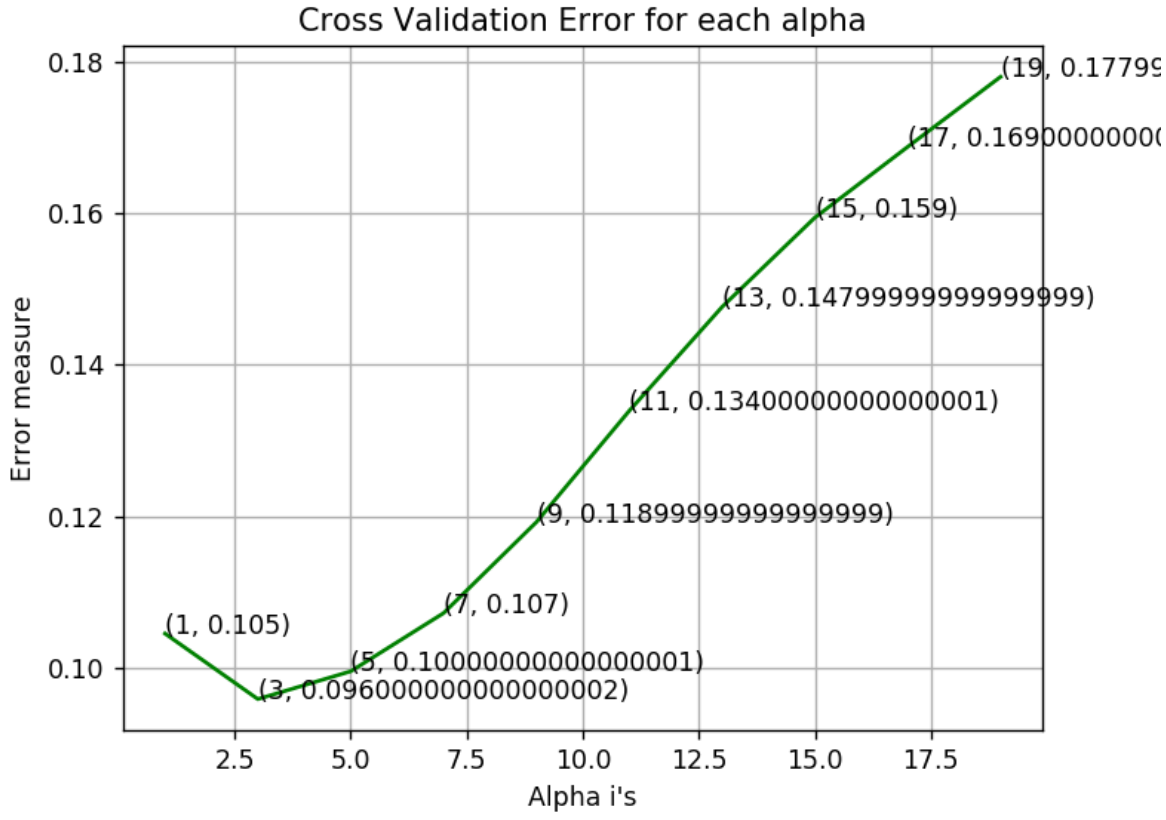
K-NN : K-Nearest Neighbors

In [ ]:

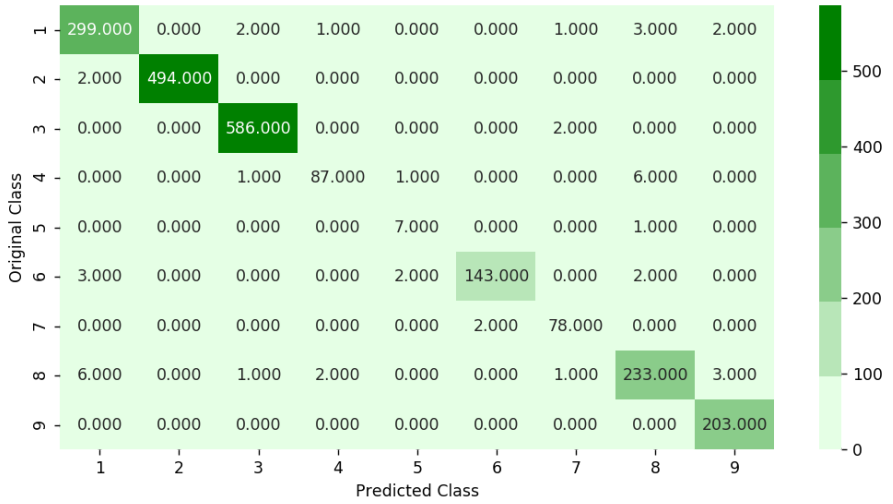
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/genera
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

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

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 0.104531321344
log_loss for k =  3 is 0.0958800580948
log_loss for k =  5 is 0.0995466557335
log_loss for k =  7 is 0.107227274345
log_loss for k =  9 is 0.119239543547
log_loss for k =  11 is 0.133926642781
log_loss for k =  13 is 0.147643793967
log_loss for k =  15 is 0.159439699615
log_loss for k =  17 is 0.16878376444
log_loss for k =  19 is 0.178020728839

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

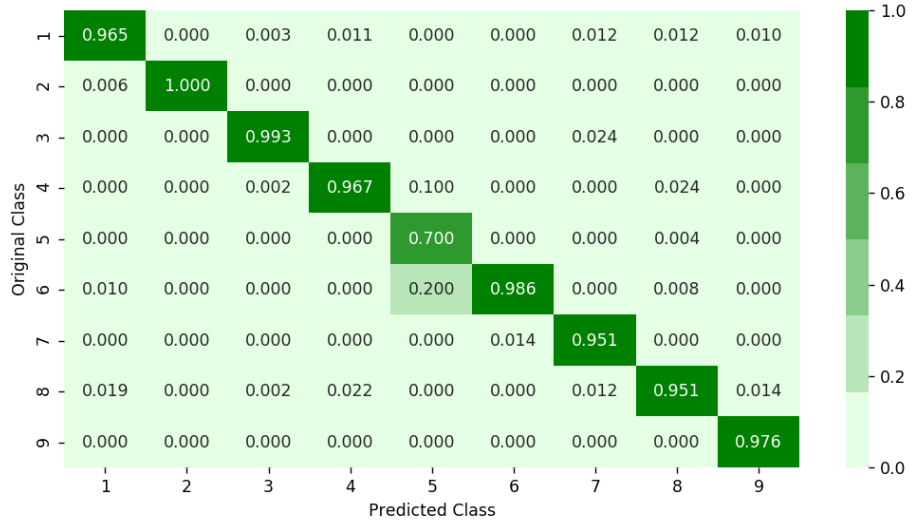



```
log loss for train data 0.0476773462198
log loss for cv data 0.0958800580948
log loss for test data 0.0894810720832
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix --------
-----------------------------------------

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



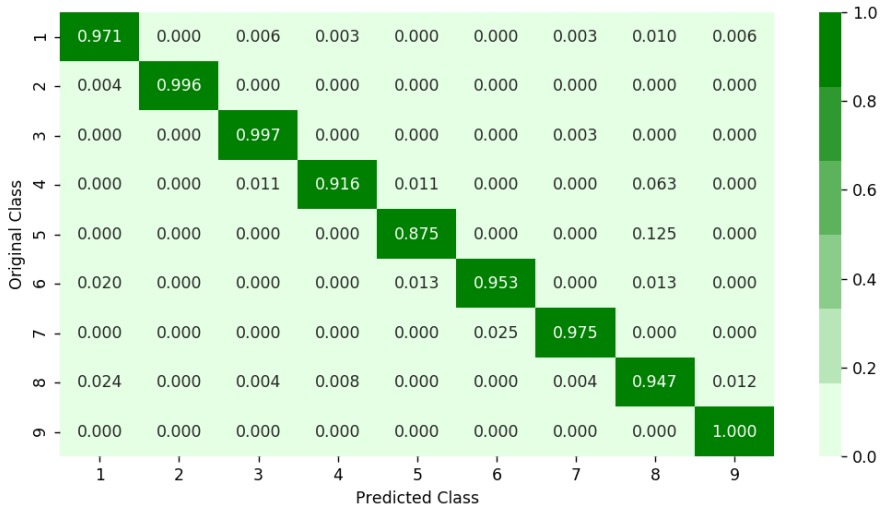
---------- Precision matrix --------

<IPython.core.display.Javascript object>



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

<IPython.core.display.Javascript object>




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

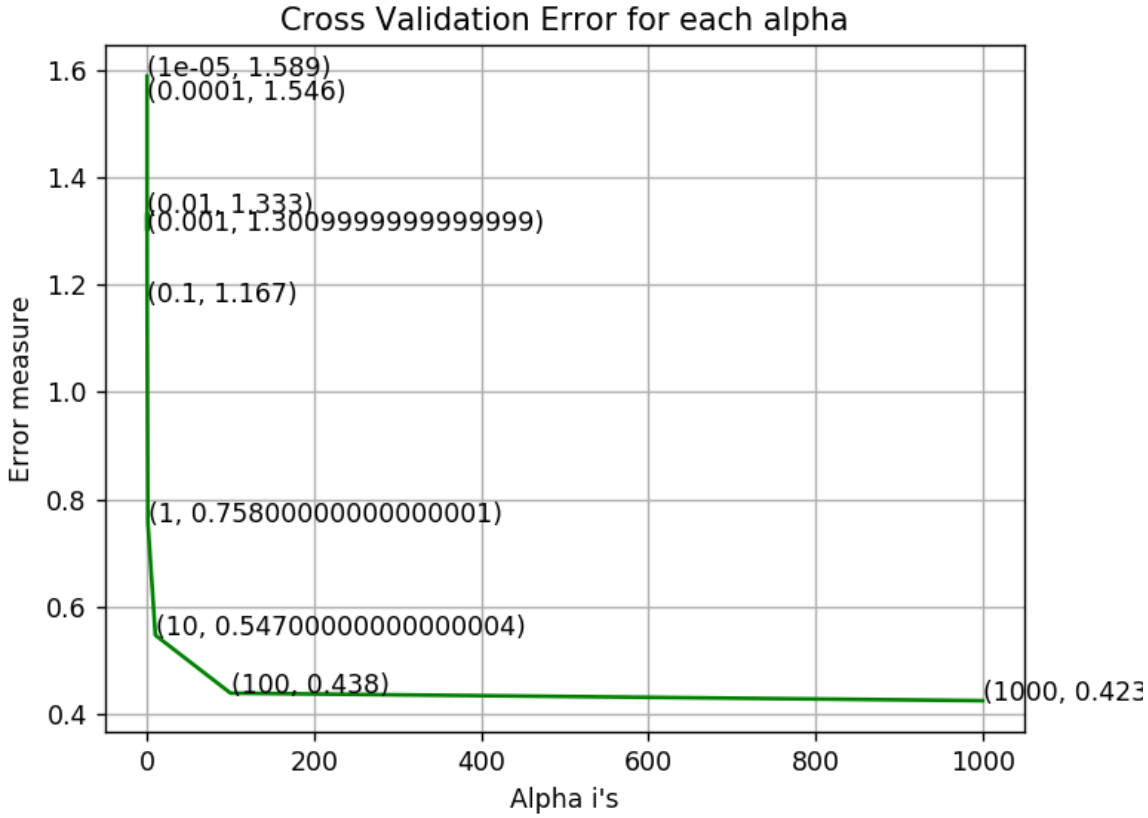
4.4.2 Logistic Regression

In [ ]:

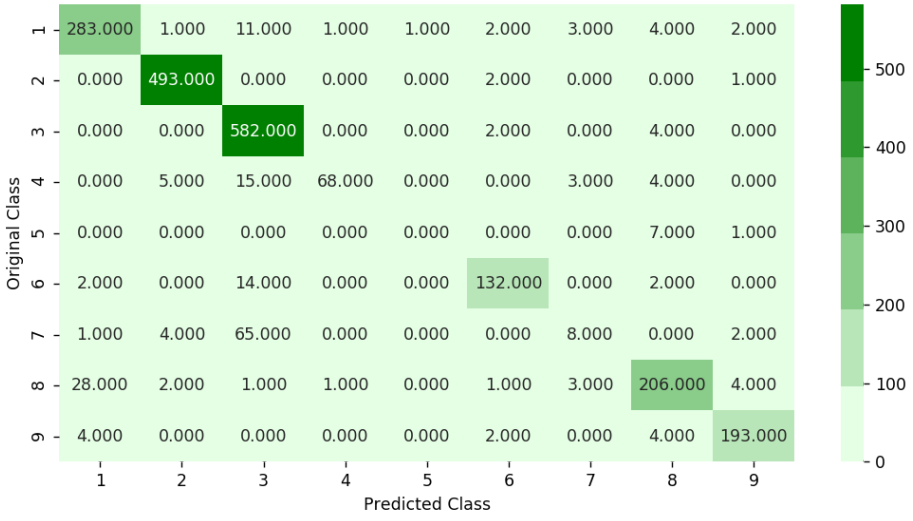
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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

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


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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

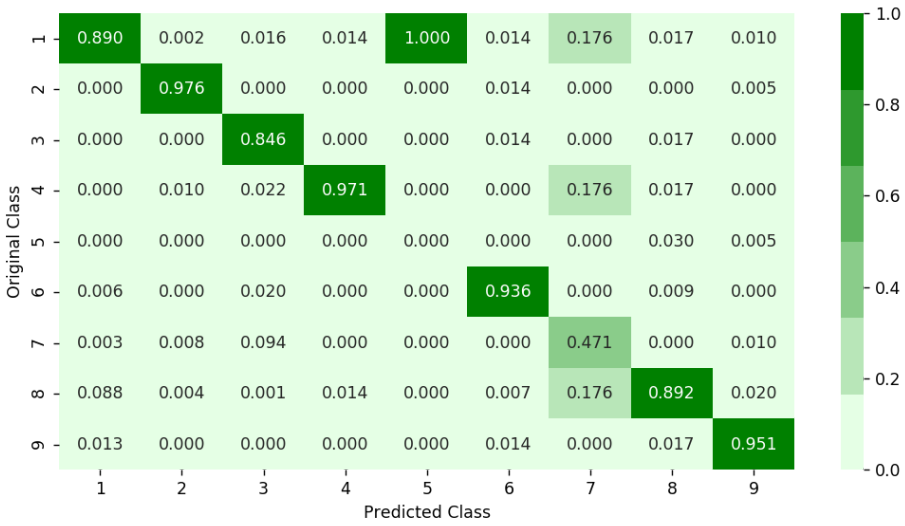
```
log_loss for c =  0.1 is 1.16705751378
log_loss for c =  1 is 0.757667807779
log_loss for c =  10 is 0.546533939819
log_loss for c =  100 is 0.438414998062
log_loss for c =  1000 is 0.424423536526

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



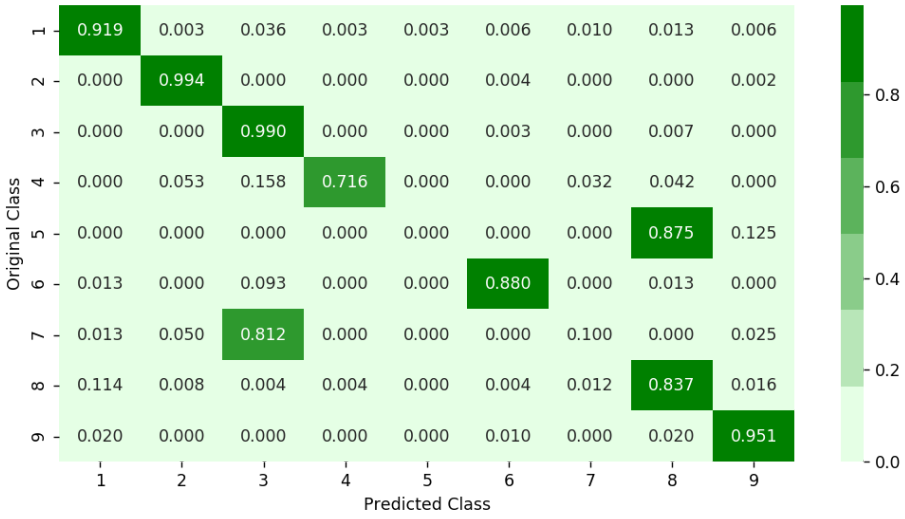

```
log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points  9.61361545538
---------------------------------------------------- Confusion matrix --------
----------------------------------------

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



```
---------------------------------------------------- Precision matrix --------
-----------------------------------------
<IPython.core.display.Javascript object>
```




```
Sum of columns in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
---------------------------------------------------- Recall matrix -----------
--------------------------------------
<IPython.core.display.Javascript object>
```




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

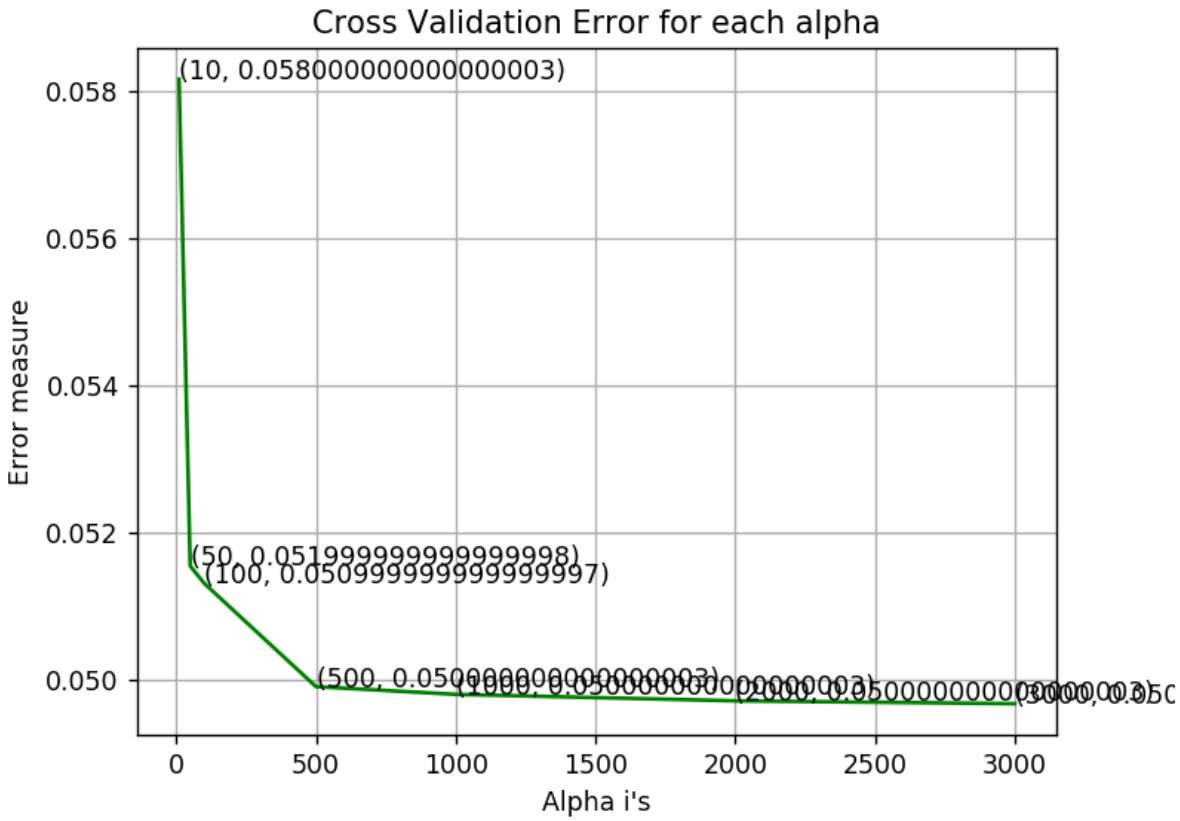
4.4.3 Random Forest Classifier

In [ ]:

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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

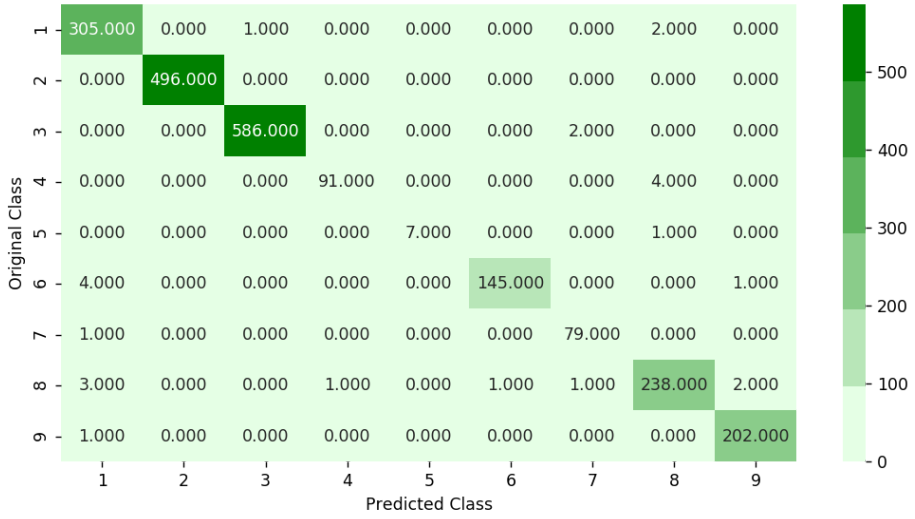
```
log_loss for c =  10 is 0.0581657906023
```

```
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

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



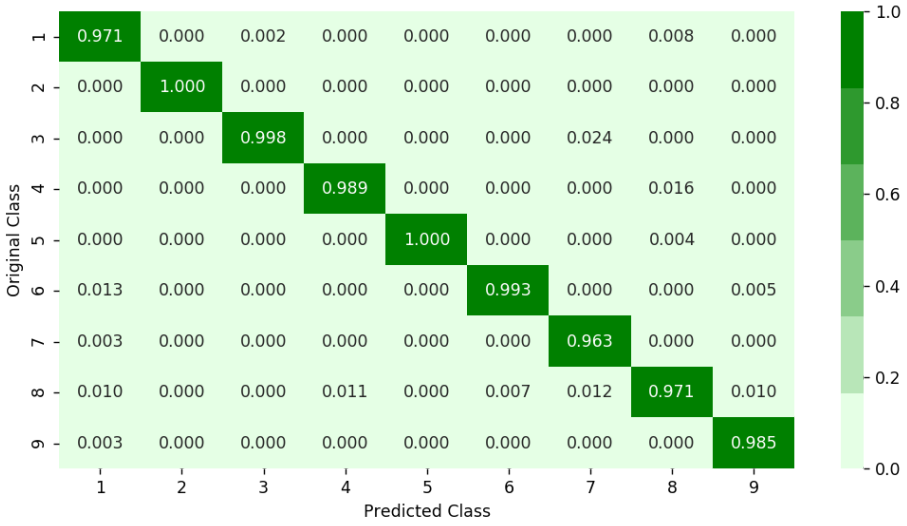

```
log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points  1.14995400184
---------------------------------------------------- Confusion matrix --------
----------------------------------------
```

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



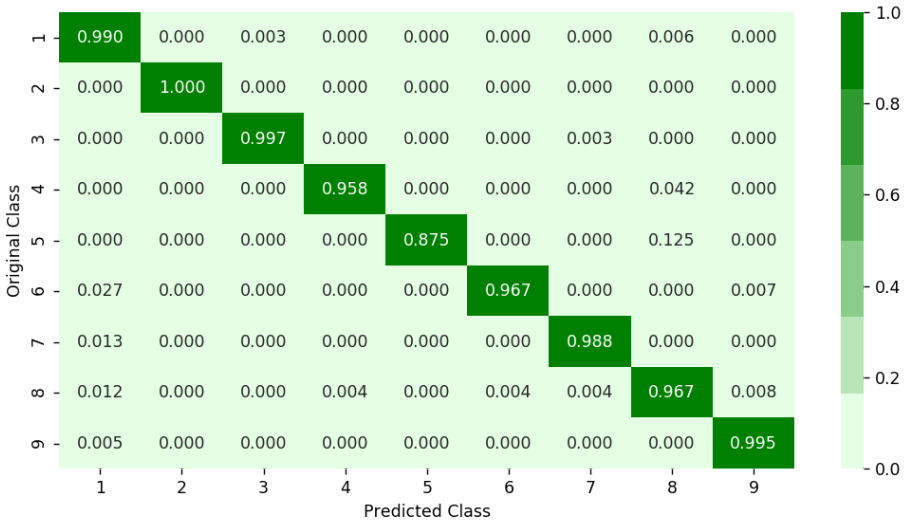
```
---------------------------------------------------- Precision matrix --------
-----------------------------------------
```

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




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

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



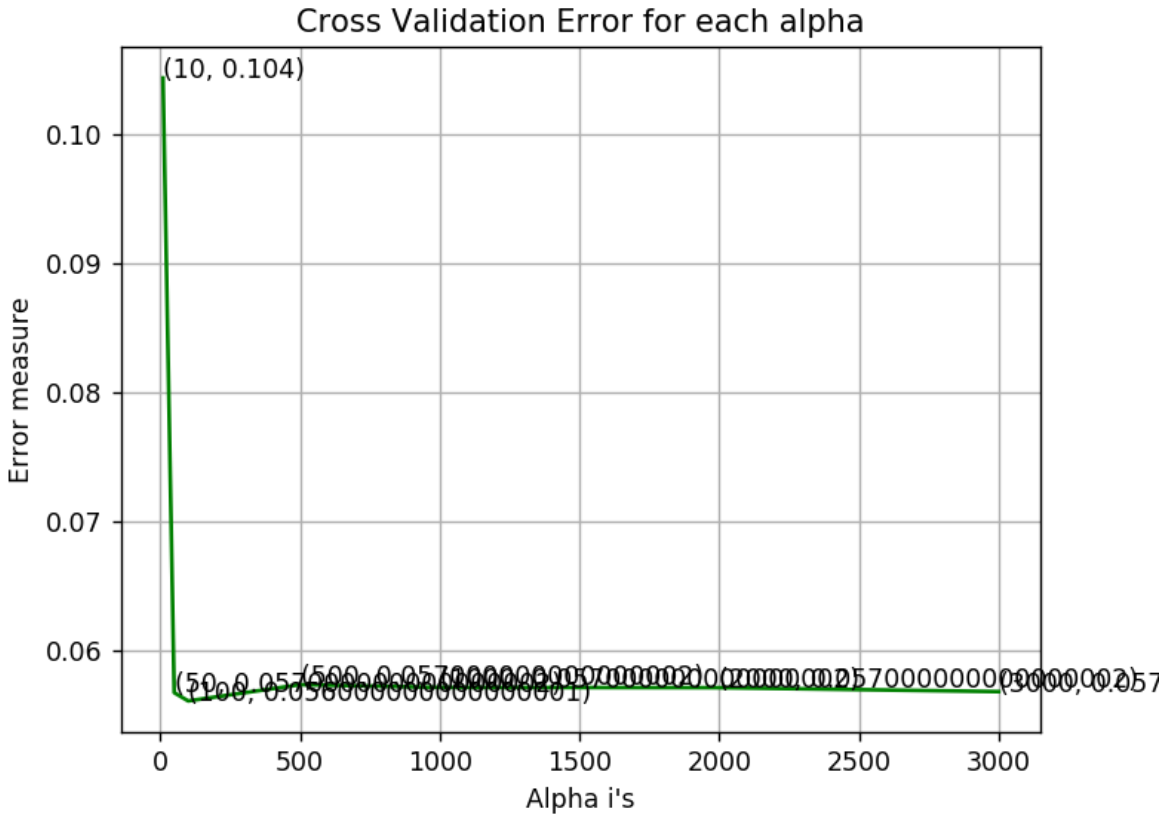

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.4.4 XgBoost Classifier

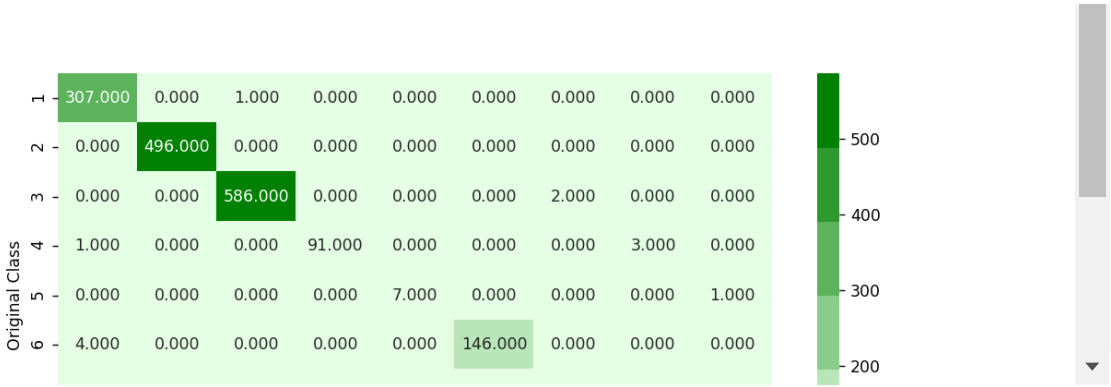
In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

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.classes_, eps=1e-1

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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
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

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

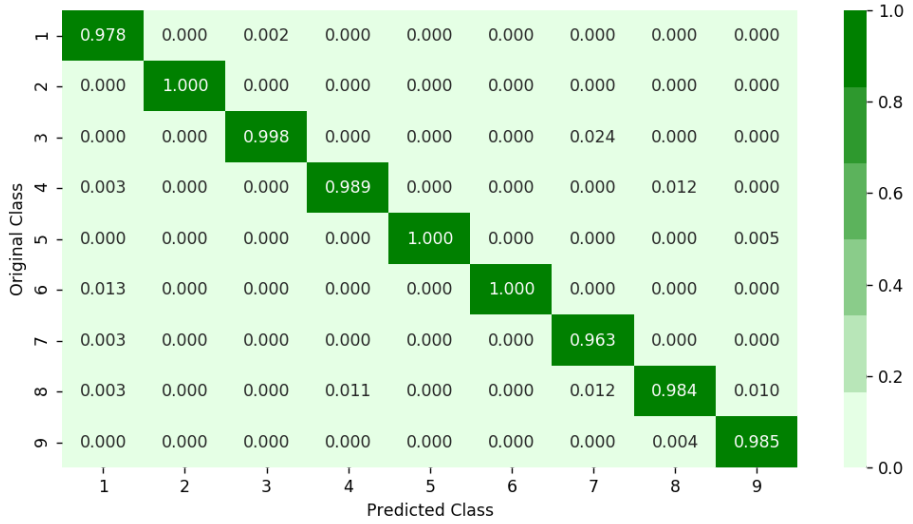



```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.05607503
8646
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
---------------------------------------------------- Confusion matrix --------
----------------------------------------

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



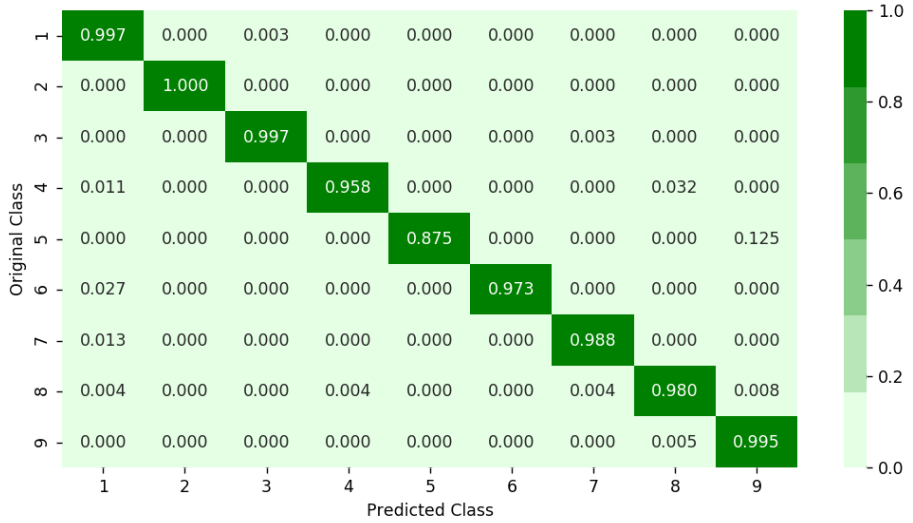
---------- Precision matrix --------

<IPython.core.display.Javascript object>



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

<IPython.core.display.Javascript object>



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

4.4.5 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_jobs=-1,)
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:   3
9.3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.3min remaining:   2
3.0s
[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[163]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsa
mple_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, 0.1
5, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 1
0], 'colsample_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/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5,
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 [ ]:

```
result.head()
```

Out[171]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.0(
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0(
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0(
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0(
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0(

5 rows × 260 columns

In [ ]:

```
result_asm.head()
```

Out[174]:

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

5 rows × 54 columns

In [ ]:

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

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

In [ ]:

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

Out[182]:

	0	1	2	3	4	5	6	7	8	
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0

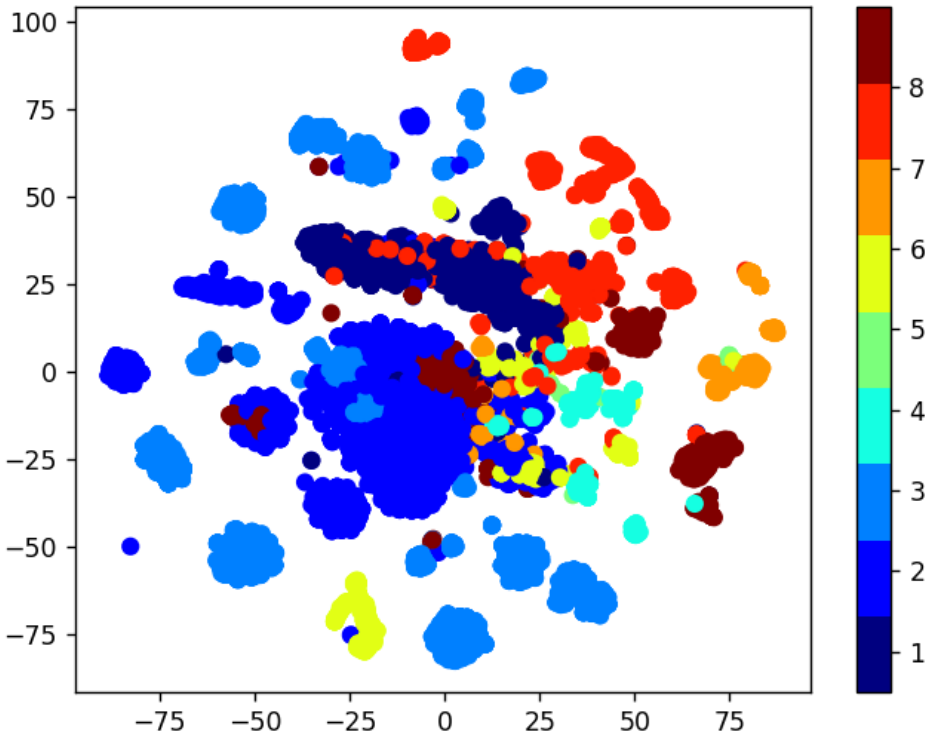
5 rows × 307 columns

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()
```

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



4.5.3. Train and Test split

In [ ]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,st
```

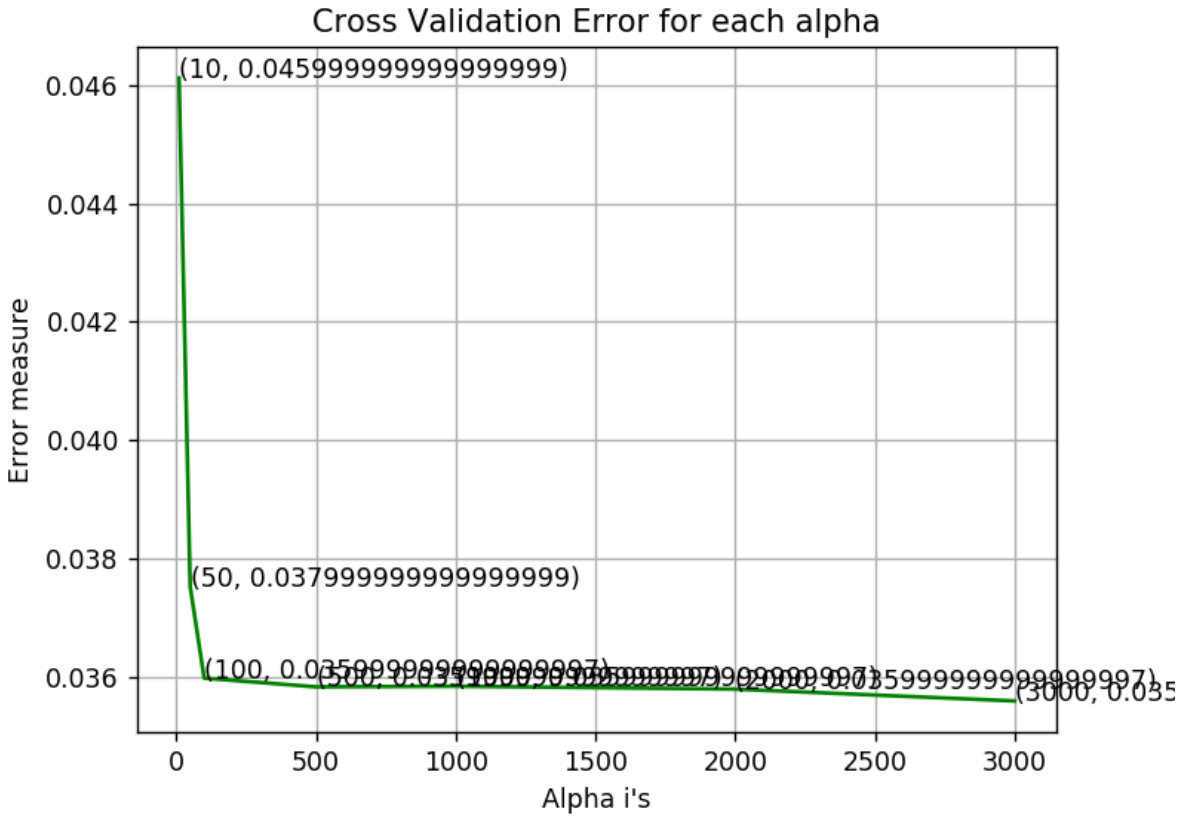
4.5.4. Random Forest Classifier on final features

In [ ]:

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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

```
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
```

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




```
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: 0.0355909
487962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

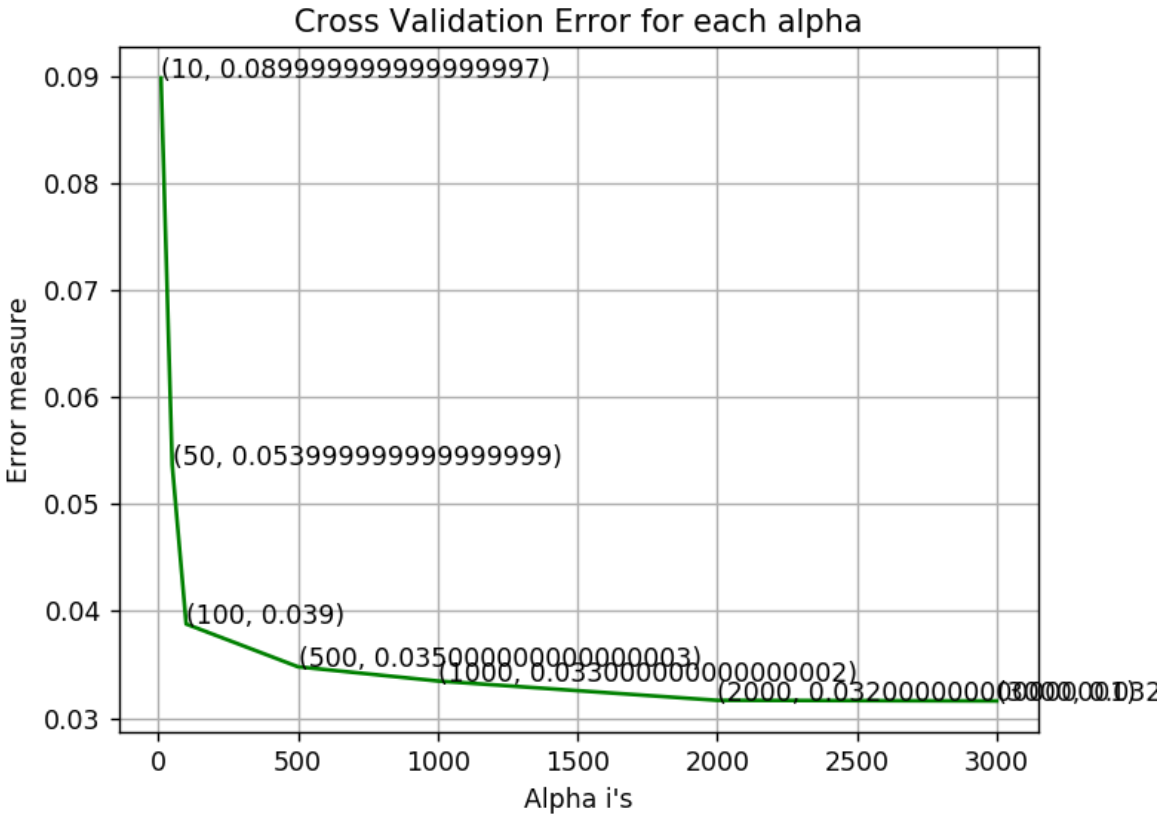
4.5.5. XgBoost Classifier on final features

In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

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.classes_, eps=1e

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

```
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


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




```
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: 0.0315972
694477
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_jobs=-1,)
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.
6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remaining:  1.
8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remaining:   4
4.5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

Out[187]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsa
mple_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, 0.1
5, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 1
0], 'colsample_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': 0.1
5, 'colsample_bytree': 0.3}
```

In [ ]:

```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.3,
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 is:",log_loss(
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
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: 0.0344955
487471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

In [ ]:

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

In [ ]:

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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,1
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()
```

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video (https://www.youtube.com/watch?v=VLQTRILGz5Y#t=13m11s)) and include pixel intensity features to improve the logloss

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

2. if you are having computation power limitations, you can try using google cola
b, 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: http
s://www.youtube.com/channel/UCRH_z-oM0LROvHPe_KYR4Wg (we suggest you to use GCP ov
er Colab)
```

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

https://askubuntu.com/a/341637

In [ ]:

```
hexa_corpus="00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,
#bought all the values as hexa corpus into the file
from nltk.util import ngrams
byte_bigram_column=['ID'] #added ID to know the file
for i in (hexa_corpus.split(",")):
    for j in (hexa_corpus.split(",")):
        byte_bigram_column.append(i+' '+j) #made a Column i.e 0e 0f which account to 255*25



byte_bigram_file=open('bigram_file.csv','w+')
byte_bigram_file.write(','.join(byte_bigram_column))
byte_bigram_file.write("\n")




for file in tqdm(byteFiles):
    temp = dict(zip(byte_bigram_column, [0]*len(byte_bigram_column))).copy()
    temp['ID']=str(file)
    all_line=[]
    if(file.endswith("txt")):
        with open(os.path.join('byteFiles', file),"r") as text_file:
            for line in (text_file):
                bi_g=[]
                all_line.extend(line.rstrip().split(" "))
            bi_g = [' '.join(x) for x in list(ngrams(all_line, 2))]
            for i in bi_g:
                temp[i.lower()]+=1
            byte_features=[]
            byte_features = [str(temp[x]) for x in byte_bigram_column]
            byte_bigram_file.write(','.join(byte_features))
            byte_bigram_file.write("\n")
            del temp
byte_bigram_file.close()
```

In [ ]:

```
df_bigram=pd.read_csv('byte_bigram.csv',nrows=5)
```

In [ ]:

```
df_bigram.iloc[:,1:]
```

Out[17]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	...	?? f7	?? f8	?? f9	?? fa	? f
0	31432	1788	187	239	324	63	53	155	117	185	...	0	0	0	0	
1	6047	44	27	78	16	55	79	36	21	51	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	13889	180	137	109	155	12	5	23	116	6	...	0	0	0	0	
4	12576	618	160	174	579	573	589	171	586	183	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10863	172855	12683	1596	1572	1138	243	342	1184	1340	741	...	0	0	0	0	
10864	5208	39	23	73	18	56	24	34	20	57	...	0	0	0	0	
10865	49433	255	165	193	147	151	136	141	149	131	...	0	0	0	0	
10866	299893	1377	1578	1249	1238	1343	1173	1317	1387	1150	...	0	0	0	0	
10867	5959	47	28	87	18	52	84	38	24	49	...	0	0	0	0	

10868 rows × 66049 columns

In [ ]:

```

```

In [ ]:

```
dense_matrix=np.array(df_bigram.iloc[:,1:]) #pandas data frame to numpy matrix
sparse_matrix=sparse.csr_matrix(dense_matrix) #numpy matrix to sparse
```

In [ ]:

```

```

In [ ]:

```
sparse_matrix=sparse.csr_matrix(dense_matrix) #numpy matrix to sparse
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-4daec1daa9d7> in <module>
----> 1 sparse_matrix=sparse.csr_matrix(dense_matrix) #numpy matrix to spars
e

NameError: name 'sparse' is not defined
```

In [ ]:

```
sparse_matrix.shape
```

Out[28]:

```
(10868, 66049)
```

bigram_feature saving npz format and preprocessing

In [ ]:

```
df_bigram=pd.read_csv('byte_bigram.csv',nrows=5)
```

In [ ]:

```
from sklearn.preprocessing import normalize
bigram_process_vect= normalize(sparse_matrix, axis = 0)
```

In [10]:

```
from scipy import sparse
#sparse.save_npz("bigram_process_vect.npz", bigram_process_vect)
bigram_process_vect=sparse.load_npz('bigram_process_vect.npz')

import pickle
with open("bigram_columns.txt","rb") as fp:
    bigram_columns=pickle.load(fp)
with open("bigram_bytefile.txt","rb") as fp:
    bigram_bytefile=pickle.load(fp)
bigram_process_dense=bigram_process_vect.todense()
del bigram_process_vect
import pandas as pd
bigram_process_vect_df=pd.DataFrame(bigram_process_dense[0:,0:],columns=bigram_columns[1:])
del bigram_process_dense
data=pd.Series(bigram_bytefile)
del bigram_bytefile
bigram_process_vect_df.insert(loc=0,column='ID',value=data)
del data
```

In [11]:

```
bigram_process_vect_df['ID'] = bigram_process_vect_df['ID'].map(lambda x: x.rstrip('.txt'))
bow_df=pd.read_csv("result_unigrm_byte.csv")
bow_df.drop(columns=['Unnamed: 0','Unnamed: 0.1'],axis=1,inplace=True)
tot_feat = pd.merge(left=bigram_process_vect_df, right=bow_df[['size','Class','ID']],how='l
del bigram_process_vect_df
del bow_df
```

In [13]:

```
tot_feat.head()
```

Out[13]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 (
0	D2FbBoJAweiY8X0pxTOn	0.005062	0.017984	0.002374	0.003711	0.004159	0.001102	0.0007
1	ATVo94avrEyjnexXU8W7	0.000974	0.000443	0.000343	0.001211	0.000205	0.000962	0.0011
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
3	0GUIi7xAlODwZ4YBenNM	0.002237	0.001810	0.001739	0.001693	0.001989	0.000210	0.0000
4	CdeQsaIqAiMVRjT2Pun6	0.002025	0.006216	0.002031	0.002702	0.007431	0.010027	0.0088

5 rows × 66052 columns

In [ ]:

```
tot_feat
```

In [68]:

```
final_y=tot_feat['Class']
```

In [ ]:

```
import joblib
file_name="bigram_topindez.pkl"
joblib.dump(indez,file_name)
```

In [186]:

```
tot_feat.head()
```

Out[186]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 (
0	D2FbBoJAweiY8X0pxTOn	0.005062	0.017984	0.002374	0.003711	0.004159	0.001102	0.0007
1	ATVo94avrEyjnexXU8W7	0.000974	0.000443	0.000343	0.001211	0.000205	0.000962	0.0011
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
3	0GUIi7xAlODwZ4YBenNM	0.002237	0.001810	0.001739	0.001693	0.001989	0.000210	0.0000
4	CdeQsaIqAiMVRjT2Pun6	0.002025	0.006216	0.002031	0.002702	0.007431	0.010027	0.0088

5 rows × 66052 columns

In [ ]:

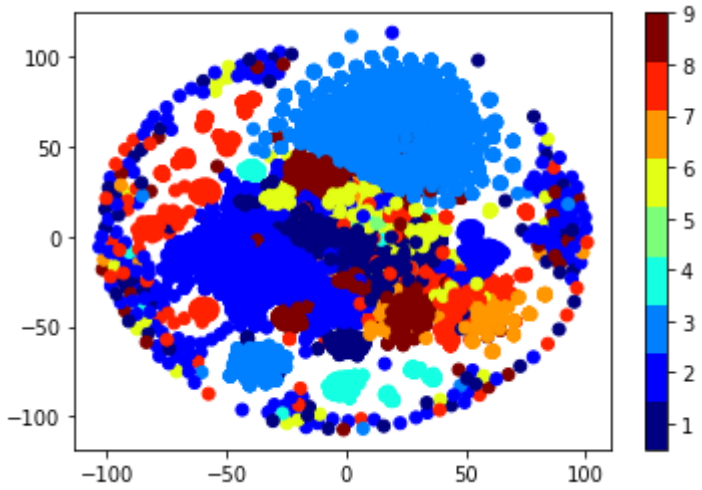
```

```

In [ ]:

```
final_y=final_y.replace(np.nan, 3.0)
```

In [ ]:

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



In [ ]:

```
tot_feat.head(6)
```

Out[13]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	
0	0.005062	0.017984	0.002374	0.003711	0.004159	0.001102	0.000796	0.002941	0.001796	0
1	0.000974	0.000443	0.000343	0.001211	0.000205	0.000962	0.001186	0.000683	0.000322	0
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
3	0.002237	0.001810	0.001739	0.001693	0.001989	0.000210	0.000075	0.000436	0.001781	0
4	0.002025	0.006216	0.002031	0.002702	0.007431	0.010027	0.008846	0.003244	0.008995	0
5	0.000473	0.000091	0.000013	0.001832	0.000039	0.000017	0.000030	0.000019	0.000046	0

6 rows × 66050 columns

In [ ]:

```
y=final_y.replace(np.nan, 3.0)
```

In [ ]:

```
tot_feat.isnull().sum
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(tot_feat, final_y,stratify=final_y,test
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

```

```

In [ ]:

```
del tot_feat
del final_y
```

In [ ]:

```
X_train.shape
```

Out[19]:

```
(6955, 66050)
```

In [ ]:

```

```

In [ ]:

```
# ---------------------------------
# default parame  ters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=Non
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# class_weight=None)

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

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

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

alpha=[50,500,1000,2000]
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,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train )
print ('log loss for train data',(log_loss(y_train , predict_y, labels=sig_clf.classes_, ep
predict_y = sig_clf.predict_proba(X_cv )
print ('log loss for cv data',(log_loss(y_cv , predict_y, labels=sig_clf.classes_, eps=1e-1
predict_y = sig_clf.predict_proba(X_test )
print ('log loss for test data',(log_loss(y_test , predict_y, labels=sig_clf.classes_, eps=
plot_confusion_matrix(y_test ,sig_clf.predict(X_test ))
```

In [ ]:

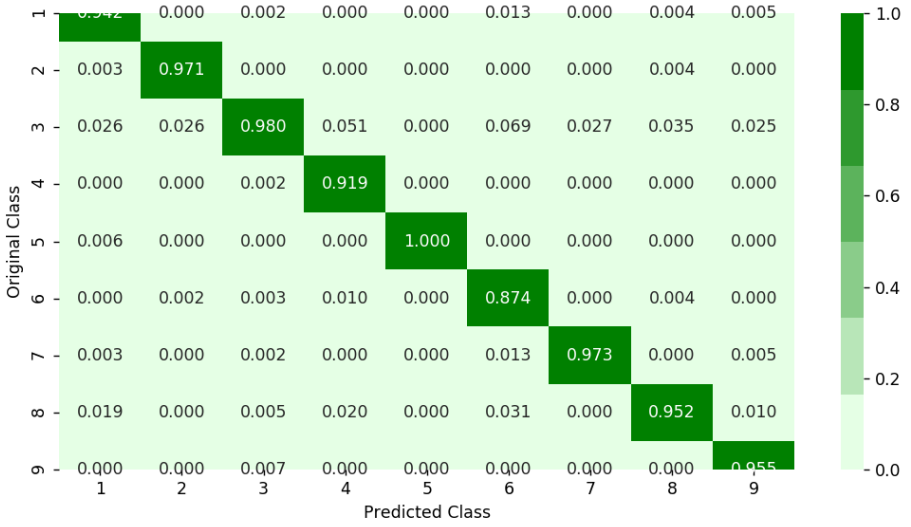
```
plot_confusion_matrix(y_test ,sig_clf.predict(X_test ))
```

```
Number of misclassified points  4.323827046918123
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>


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

In [ ]:

```
r_cfl=RandomForestClassifier(n_estimators=1000,random_state=42,n_jobs=2)
r_cfl.fit(X_train,y_train)
```

Out[62]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=2, oob_score=False, random_state=42, verbose=
0,
                       warm_start=False)
```

In [ ]:

```
len(r_cfl.feature_importances_)
```

Out[21]:

```
66050
```

In [ ]:

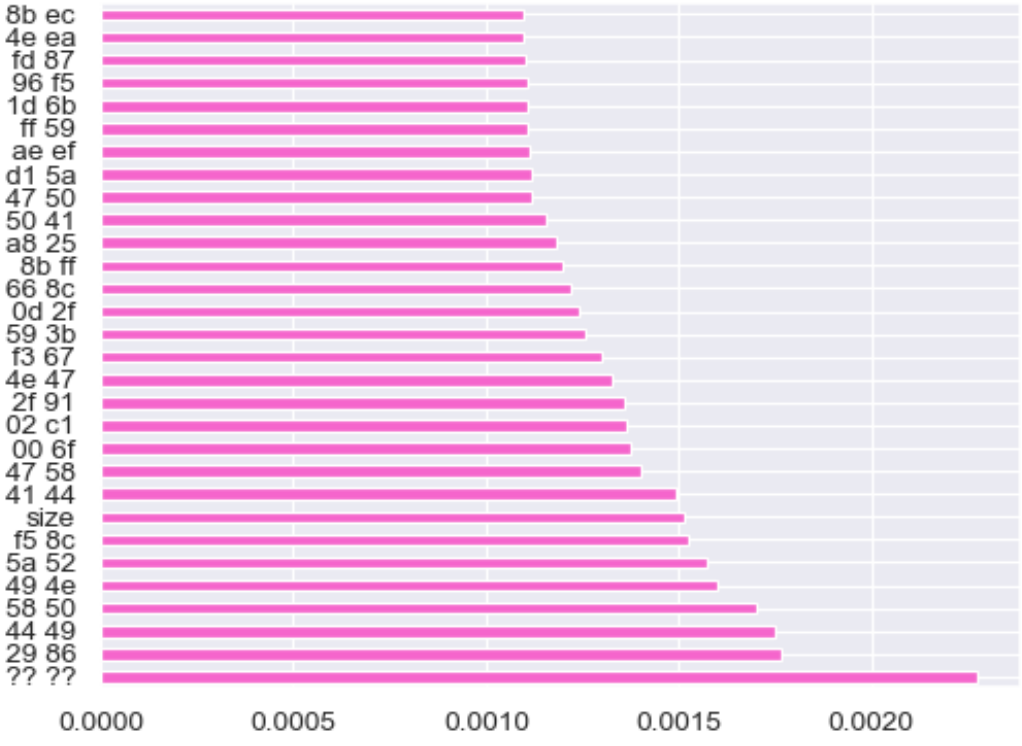
```
for name, importance in zip(X_train.columns, r_cfl.feature_importances_):
...     print(name, "=", importance)
```

```
00 00 = 0.0005037027750615231
00 01 = 0.0007661155656797805
00 02 = 0.000535823856179823
00 03 = 0.0003260130909661878
00 04 = 0.0006355139413726839
00 05 = 0.0005457475690566932
00 06 = 0.00022018804698705792
00 07 = 0.0002686860524460420174
00 08 = 0.0004062578283268711
00 09 = 0.00028740359326657404
00 0a = 0.00017075905924766413
00 0b = 0.00015539892148379688
00 0c = 0.0003617479458403887
00 0d = 0.00012300189509819063
00 0e = 0.00019195616010525736
00 0f = 0.0004927593694535286
00 10 = 0.00045069601388933217
00 11 = 0.00010772483569769189
00 12 = 0.000735268943817739
```

In [ ]:

```
feat_importances = pd.Series(r_cfl.feature_importances_, index=X_train.columns)
feat_importances.nlargest(30).plot(kind='barh')
plt.show()
```

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



In [ ]:

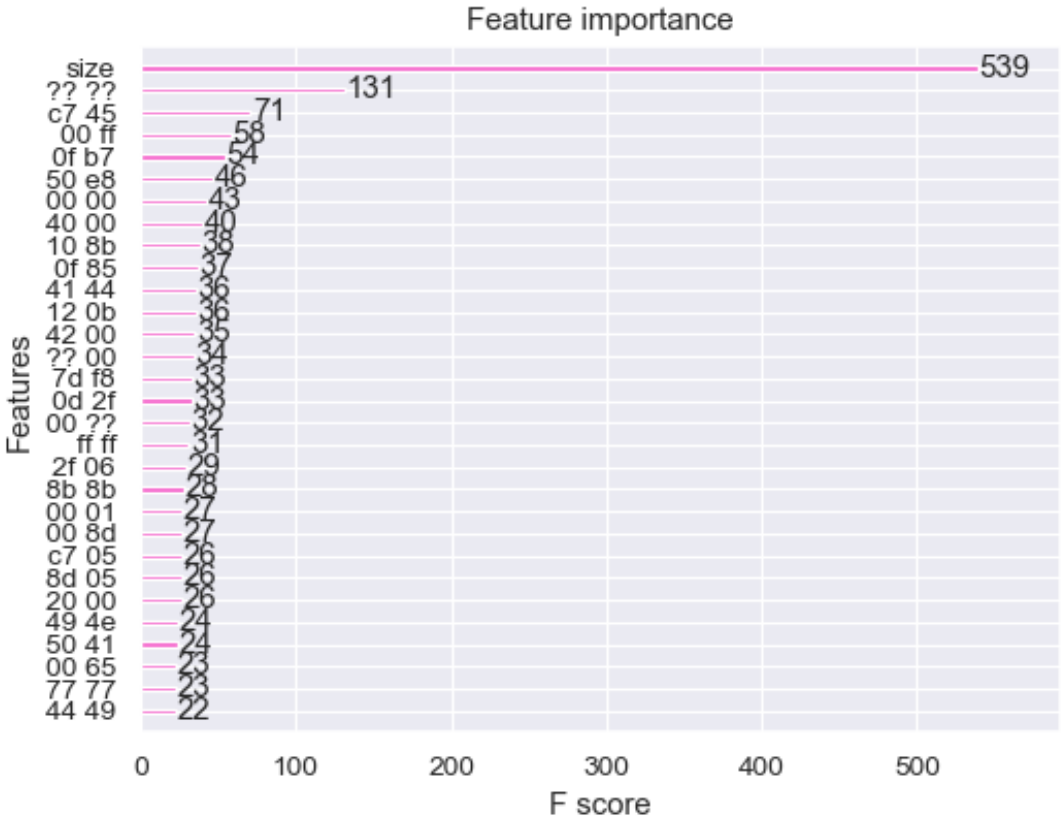
```
imp_feature_indx = np.argsort(r_cfl.feature_importances_)[::-1]
imp_value = np.take(r_cfl.feature_importances_, imp_feature_indx[:40])
imp_feature_name = np.take(features, imp_feature_indx[:40])
sns.set()
plt.figure(figsize = (10, 5))
ax = sns.barplot(x = imp_feature_name, y = imp_value)
ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
plt.title('Important Features')
plt.xlabel('Feature Names')
plt.ylabel('Importance')
plt.show()
```

<IPython.core.display.Javascript object>

In [ ]:

```
xgb.plot_importance(xgb_all_models,max_num_features=30)
plt.show()
```

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



In [ ]:

```
 0.0309563
    0.5519779208831647
    byte_bigram + pxl features



    asm_bow+800_pxl for asm
    0.0356
    0.64397

    10k_byte_bigarm, bow_byte, size of bye
     0.0500170
        1.0119595
```

In [ ]:

```
best_alpha = np.argmin(cv_log_error_array)
```

In [ ]:

```
best_alpha
```

Out[98]:

```
3
```

In [ ]:

```
len(X_train)
```

Out[18]:

```
6955
```

In [ ]:

```
import lightgbm as lgb
```

In [ ]:

```
x_cfl=XGBClassifier()

prams={'learning_rate':[0.01,0.05,0.1,0.25],
     'n_estimators':[100,200,1000,2000],
     'max_depth'    :  [3,5,11],
    'colsample_bytree' :[0.1,0.3,0.5,1]}

cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=0,n_jobs=2,cv=2)

cfl.fit(X_train, y_train)
```

Out[15]:

```
RandomizedSearchCV(cv=2, error_score=nan,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=Non
e,
                                           gpu_id=None, importance_type='gai
n',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=No
ne,
                                           min_child_weight=None, missing=na
n,
                                           monotone_constraints=None,
                                           n...
                                           subsample=None, tree_method=None,
                                           validate_parameters=False,
                                           verbosity=None),
                   iid='deprecated', n_iter=10, n_jobs=2,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5,
1],
                                        'learning_rate': [0.01, 0.05, 0.1,
                                                          0.25],
                                        'max_depth': [3, 5, 11],
                                        'n_estimators': [100, 200, 1000, 200
0]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [ ]:

```
cfl.best_params_
```

Out[18]:

```
{'n_estimators': 100,
 'max_depth': 3,
 'learning_rate': 0.1,
 'colsample_bytree': 1}
```

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

In [ ]:

```
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
The train log loss is: 0.015141884467139094
The cross validation log loss is: 0.03725102829333129
The test log loss is: 0.04182744699729518
```

In [ ]:

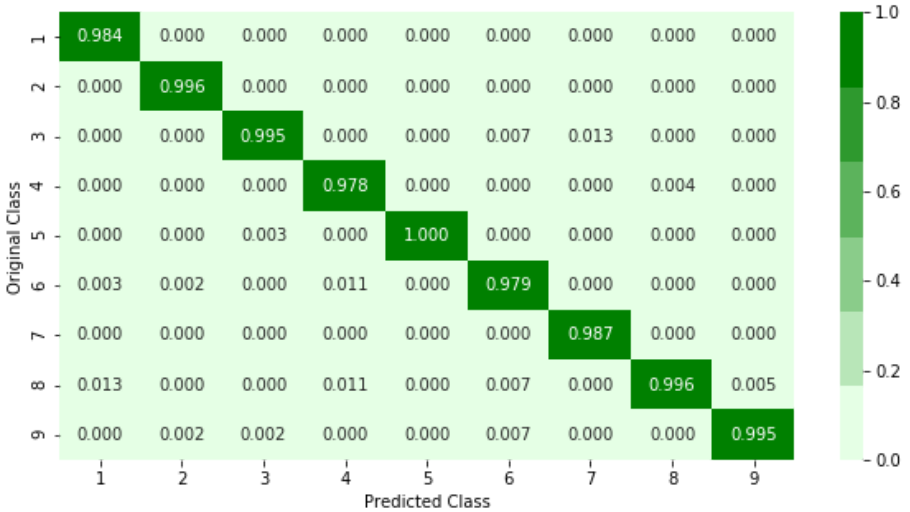
```
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
Number of misclassified points  0.8279668813247469
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>

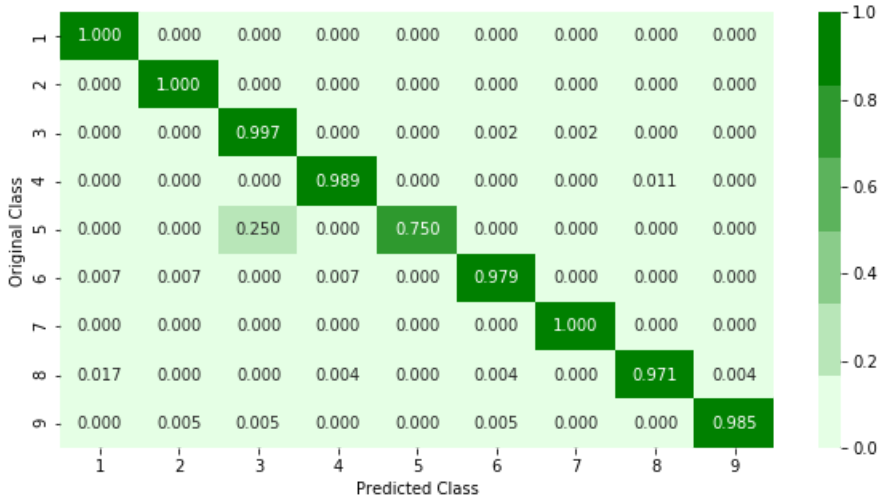

```
---------------------------------------------------- Precision matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

In [ ]:

```
import xgboost as xgb
import joblib
```

In [ ]:

```
xgb_all_models = xgb.XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=2)
model=xgb_all_models.fit(X_train, y_train)
file_name="xgb_model.pkl"
xgb_model=joblib.dump(model,file_name)
```

In [ ]:

```
model.feature_importances_
```

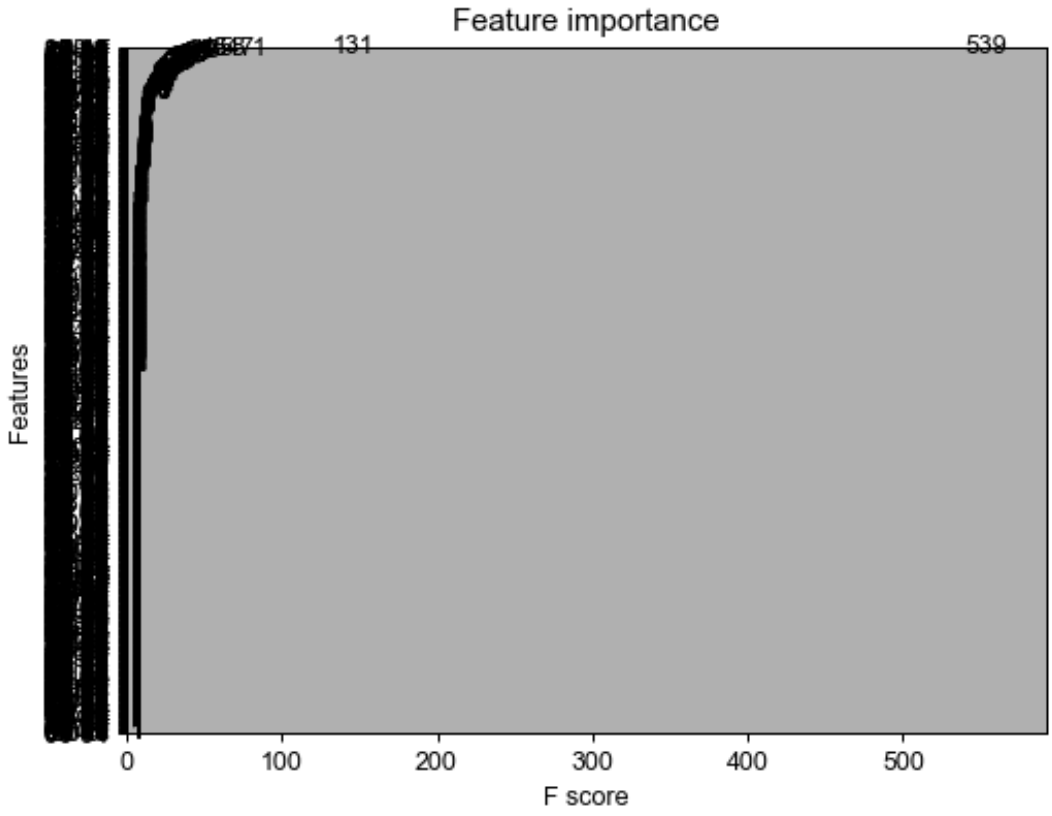
Out[12]:

```
array([0.00053658, 0.00142256, 0.00011132, ..., 0.        , 0.01384808,
       0.00137346], dtype=float32)
```

In [ ]:

```
xgb_all_models = xgb.XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=2)
xgb_all_models.fit(X_train, y_train)

xgb.plot_importance(xgb_all_models)
plt.show()
```

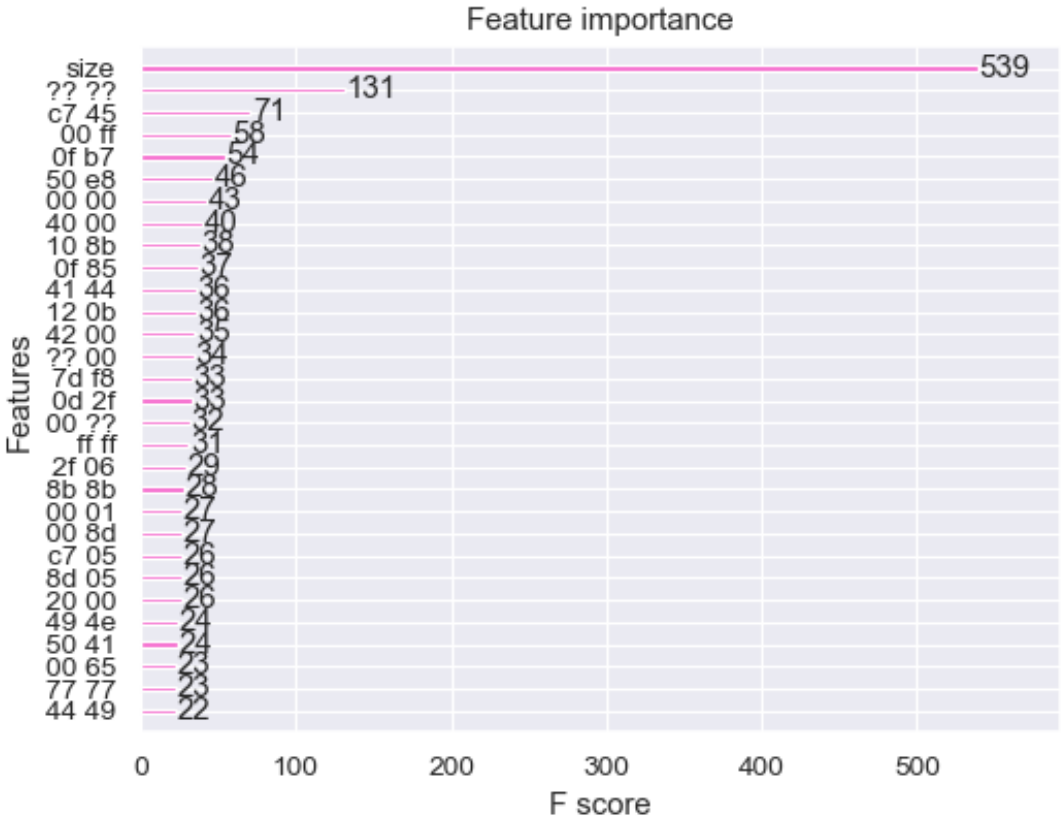
```
<IPython.core.display.Javascript object>
```



In [ ]:

```
xgb.plot_importance(xgb_all_models,max_num_features=30)
plt.show()
```

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



selecting top 10000 features

In [ ]:

```
for name, importance in zip(X_train.columns, model.feature_importances_):
...     print(name, "=", importance)
```

```
00 00 = 0.0005365821
00 01 = 0.0014225618
00 02 = 0.00011131508
00 03 = 0.0001882126
00 04 = 0.0011042665
00 05 = 0.0
00 06 = 2.4585192e-05
00 07 = 0.0
00 08 = 3.6892357e-05
00 09 = 0.00033135313
00 0a = 0.0
00 0b = 4.7360074e-05
00 0c = 4.4197856e-07
00 0d = 0.0
00 0e = 0.0
00 0f = 5.1317843e-06
00 10 = 1.9809162e-05
00 11 = 0.0
00 12 = 2.381652e-05
```

In [ ]:

```
feat_importances = pd.Series(model.feature_importances_, index=X_train.columns)
```

In [ ]:

```
idx=np.argsort(feat_importances)[::-1][0:10000]
```

In [ ]:

```
idx.index
```

Out[42]:

```
Index(['size', '?? ??', '?? ff', '?? fe', '?? fd', '?? fc', '?? fb', '?? f
a',
       '?? f9', '?? f8',
       ...
       'da 21', 'da 20', 'da 1f', 'da 1e', 'da 1d', 'da 1c', 'da 1b', 'da 1
a',
       'da 19', 'da 18'],
      dtype='object', length=10000)
```

In [ ]:

```
import joblib
file_name="top_2000.pkl"
top_2k=joblib.load(file_name)
```

In [ ]:

```
file_name="top_2000.pkl"
joblib.dump(idx.index,file_name)
```

Out[24]:

```
['top_2000.pkl']
```

In [ ]:

```
top_2k
```

Out[17]:

```
Index(['size', '?? ??', '?? ff', '?? fe', '?? fd', '?? fc', '?? fb', '?? f
a',
       '?? f9', '?? f8',
       ...
       'f9 42', 'f9 41', 'f9 40', 'f9 3f', 'f9 3e', 'f9 3d', 'f9 3c', 'f9 3
b',
       'f9 3a', 'f9 39'],
      dtype='object', length=2000)
```

In [ ]:

```
file_name="top_10000.pkl"
joblib.dump(idx.index,file_name)
```

Out[43]:

```
['top_10000.pkl']
```

In [ ]:

```
top_2k_feat=tot_feat[top_2k]
```

In [ ]:

```
top_2k_feat.head()
```

Out[12]:

	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	?? f9	?? f8	...	f9 42	f9 41	f9 40	
0	0.019011	0.000143	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000114	0.002319	0.000532	(
1	0.150520	0.016392	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000285	0.013141	0.005323	(
2	0.042438	0.007489	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	(
3	0.031523	0.002266	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000532	(
4	0.084499	0.000005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.001546	0.003194	(

5 rows × 2000 columns

In [ ]:

```
file_name="top_10000.pkl"
top_10k_feat=joblib.load(file_name)
```

In [ ]:

```
top_10k_feat
```

Out[47]:

```
Index(['size', '?? ??', '?? ff', '?? fe', '?? fd', '?? fc', '?? fb', '?? f
a',
       '?? f9', '?? f8',
       ...
       'da 21', 'da 20', 'da 1f', 'da 1e', 'da 1d', 'da 1c', 'da 1b', 'da 1
a',
       'da 19', 'da 18'],
      dtype='object', length=10000)
```

In [ ]:

```
X_train[top_10k_feat].head()
```

Out[48]:

	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	?? f9	?? f8	...	da 21	da 20	da
9121	0.201100	0.025401	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000800	0.001015	0.0034
93	0.201100	0.016650	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.003200	0.002030	0.0076
10783	0.003039	0.000136	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0003
6970	0.150254	0.016314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000978	0.001327	0.0041
1466	0.201100	0.016617	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.001600	0.002030	0.0069

5 rows × 10000 columns

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=2)
r_cfl.fit(X_train[top_10k_feat] ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train[top_10k_feat] , y_train )
predict_y = sig_clf.predict_proba(X_train[top_10k_feat])
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv[top_10k_feat])
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test[top_10k_feat])
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test[top_10k_feat]))
```

```
The train log loss is: 0.019128924687167315
The cross validation log loss is: 0.055395759369323457
The test log loss is: 0.04921533073112533
Number of misclassified points  0.8739650413983441
---------------------------------------------------- Confusion matrix ------
-----------------------------------------

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

In [ ]:

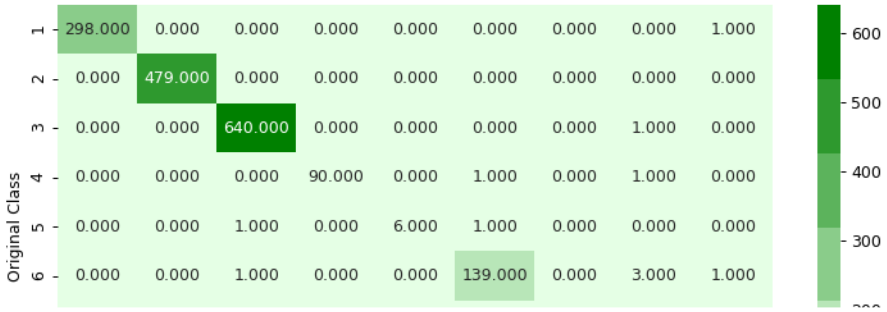
In [ ]:

In [ ]:

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=2)
r_cfl.fit(X_train[top_5k_feat] ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train[top_5k_feat] , y_train )
predict_y = sig_clf.predict_proba(X_train[top_5k_feat])
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv[top_5k_feat])
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test[top_5k_feat])
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test[top_5k_feat]))
```

```
The train log loss is: 0.018902355937181606
The cross validation log loss is: 0.05766979021169468
The test log loss is: 0.05277522468064262
Number of misclassified points  1.1039558417663293
---------------------------------------------------- Confusion matrix ------
-----------------------------------------

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



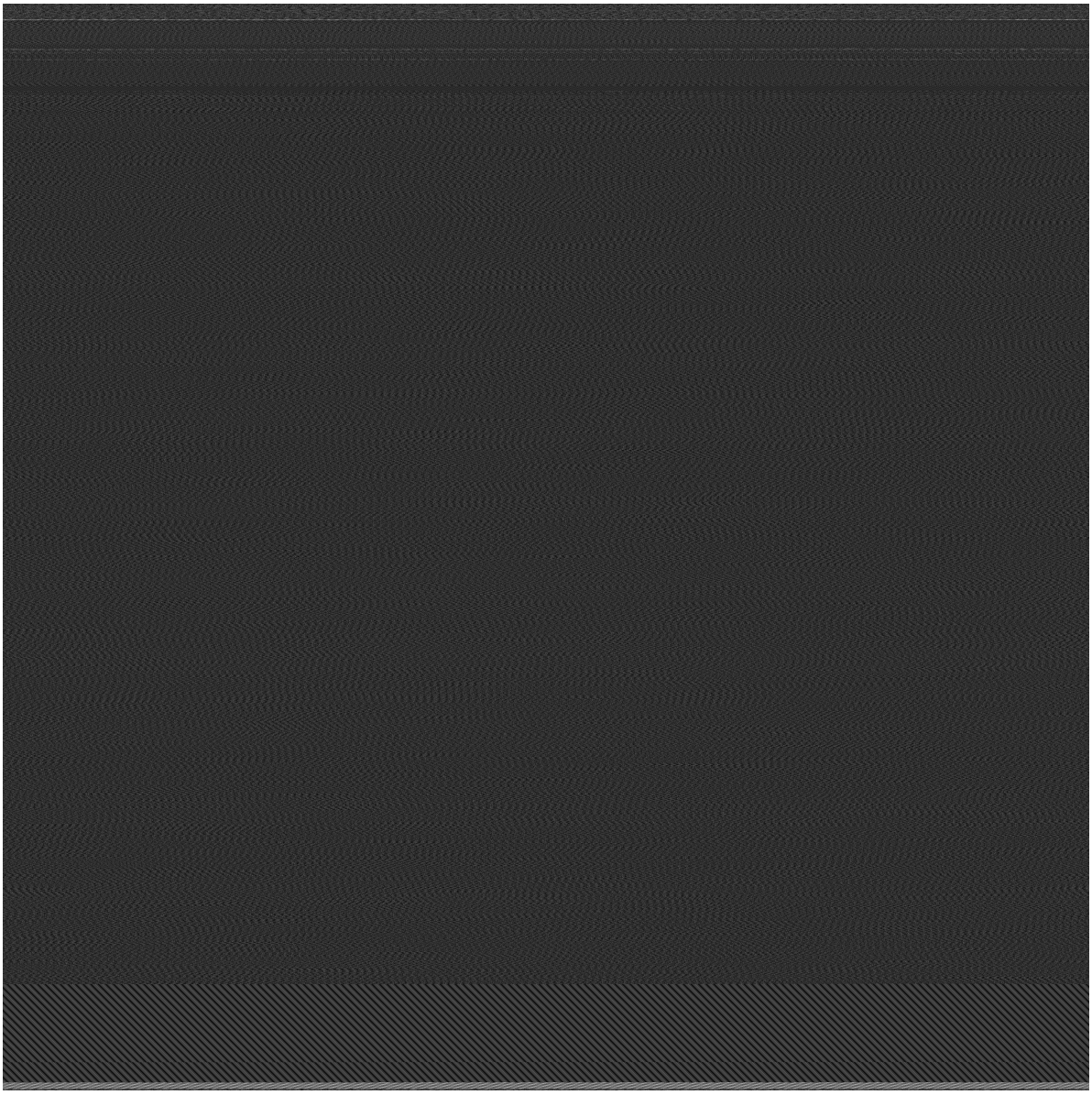
In [ ]:

```
import array
import cv2
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)
        cv2.imwrite('asm_images/' + filename + '.png',reshaped)
```

In [ ]:

```
from IPython.display import Image
Image(filename='asm_images/18A9fzqbeyripCSuLKo4.png')
```

Out[17]:



In [ ]:

```
from tqdm.notebook import tqdm
```

In [ ]:

```
import cv2
pixel_intensity = np.zeros((10868, 800))
```

In [ ]:

```
import cv2
pixel_intensity = np.zeros((10868, 800))
img_file=[]
for i, imag in tqdm(enumerate(os.listdir("asm_images"))):
    img_file.append(imag.split('.')[0])
    img = cv2.imread("asm_images/" + imag)
    pxl_arr = img.flatten()[:800]
    pixel_intensity[i, :] += pxl_arr
pxl_column = []
for i in range(800):
    pxl_column.append('pxl' + str(i))
pxl_df = pd.DataFrame(normalize(pixel_intensity , axis = 0), columns = pxl_column)
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(val
ue='')))
```

In [ ]:

```
pxl_column = []
for i in range(800):
    pxl_column.append('pxl' + str(i))
pxl_df = pd.DataFrame(normalize(pixel_intensity , axis = 0), columns = pxl_column)
```

In [ ]:

```
asm_id=pd.DataFrame(img_file)
```

In [ ]:

```
import joblib
pxl_df=joblib.load('pxl_df.pkl')
```

In [ ]:

```
pxl_df.head()
```

Out[3]:

	ID	pxl0	pxl1	pxl2	pxl3	pxl4	pxl5	pxl
0	jxuKaQ3YtEZANs06iWUf	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
1	GBQtWjTxb4v21NCDAaoI	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
2	ilogAd4QsU38IFBcuwDp	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
3	clkRCKEyz7Irxd2Go9A0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
4	1NpWfIMxU5ebEJHBAv2G	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083

5 rows × 801 columns

In [ ]:

```
asm_df=pd.read_csv("result_with_size.csv")
```

In [ ]:

```
asm_df.head()
```

Out[53]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...
1	1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	...
2	2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	...
3	3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	...
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	...

5 rows × 261 columns

In [ ]:

```
asm_df.drop(columns='Unnamed: 0',inplace=True)
```

In [ ]:

```
asm_bow_size=pd.read_csv("asm_bow_size.csv")
```

top 10k features data set with 10k_byte_bigarm, bow_byte, size of bye

In [ ]:

```
import joblib
file_name="top_10000.pkl"
top_10k_feat=joblib.load(file_name)
```

In [ ]:

```
top_2k_feat=tot_feat[top_2k]
```

In [ ]:

```
top_2k=top_2k.insert(0,'ID')
top_2k=top_2k.insert(1,'Class')
```

In [ ]:

```
top_2k_feat.head()
```

Out[22]:

	ID	Class	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	...	f9 42
0	D2FbBoJAweiY8X0pxTOn	1.0	0.019011	0.000143	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000114
1	ATVo94avrEyjnexXU8W7	3.0	0.150520	0.016392	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000285
2	cf4nzsoCmudt1kwleOTI	1.0	0.042438	0.007489	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000
3	0GUIi7xAlODwZ4YBenNM	2.0	0.031523	0.002266	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000
4	CdeQsaIqAiMVRjT2Pun6	9.0	0.084499	0.000005	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000

5 rows × 2002 columns

In [ ]:

```
top10k_feat=tot_feat[top_10k_feat]
```

In [ ]:

```
del tot_feat
```

In [ ]:

```
top10k_feat.head()
```

Out[11]:

	ID	Class	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	...	da 21
0	D2FbBoJAweiY8X0pxTOn	1.0	0.019011	0.000143	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000178
1	ATVo94avrEyjnexXU8W7	3.0	0.150520	0.016392	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000711
2	cf4nzsoCmudt1kwleOTI	1.0	0.042438	0.007489	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000
3	0GUIi7xAlODwZ4YBenNM	2.0	0.031523	0.002266	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000
4	CdeQsaIqAiMVRjT2Pun6	9.0	0.084499	0.000005	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000267

5 rows × 10002 columns

In [ ]:

```

```

In [ ]:

```
final_y=top10k_feat['Class']
```

In [ ]:

```
top10k_feat = pd.merge(left=top10k_feat, right=bow_df.drop(columns=['size','Class']),how='l
```

In [ ]:

```
top10k_feat.drop(columns=['ID','Class'],inplace=True)
```

In [ ]:

```
tot_feat[top_2k]
```

In [ ]:

```
final_y=top10k_feat['Class']
```

In [ ]:

```
top10k_feat.drop(columns=['size','Class'])
```

In [ ]:

```
top10k_feat.head()
```

Out[20]:

	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	?? f9	?? f8	...	f7	f8	f9	
0	0.019011	0.000143	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.002953	0.004279	0.002051	(
1	0.150520	0.016392	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.014340	0.017158	0.014111	(
2	0.042438	0.007489	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	(
3	0.031523	0.002266	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000680	0.025975	0.000232	(
4	0.084499	0.000005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.012085	0.014017	0.011317	(

5 rows × 10257 columns

In [ ]:

```
top10k_feat.fillna(top10k_feat.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(top10k_feat, final_y,stratify=final_y,t
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

```

```

Out[35]:

```
4.123105625617661
```

In [ ]:

```
del tot_feat
del bow_df
```

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )

file_name="xgb_model.pkl"
xgb_model=joblib.dump(model,file_name)
```

In [ ]:

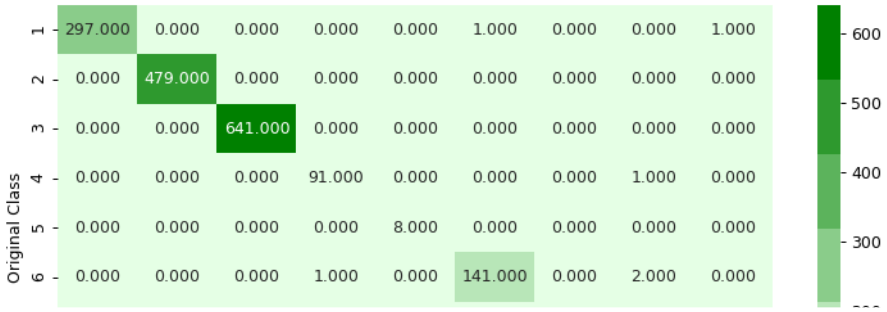
```
 0.0309563
    0.5519779208831647
    byte_bigram + pxl features



    asm_bow+800_pxl for asm
    0.0356
    0.64397

    10k_byte_bigarm, bow_byte, size of bye
     0.0500170
        1.0119595
```

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=100,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=2)
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 ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
The train log loss is: 0.01818155426915338
The cross validation log loss is: 0.043347840587675095
The test log loss is: 0.05001703258426704
Number of misclassified points  1.011959521619135
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```

<IPython.core.display.Javascript object>



In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=2)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-30-32d3ee562fed> in <module>
     22 for i in alpha:
     23     x_cfl=XGBClassifier(n_estimators=i,n_jobs=2)
---> 24     x_cfl.fit(X_train,y_train)
     25     sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
     26     sig_clf.fit(X_train, y_train)

~\Anaconda3\lib\site-packages\xgboost\sklearn.py in fit(self, X, y, sample_w
eight, base_margin, eval_set, eval_metric, early_stopping_rounds, verbose, x
gb_model, sample_weight_eval_set, callbacks)
    821                                 evals_result=evals_result, obj=obj, fe
val=feval,
    822                                 verbose_eval=verbose, xgb_model=xgb_mo
del,
--> 823                                 callbacks=callbacks)
    824
    825         self.objective = xgb_options["objective"]

~\Anaconda3\lib\site-packages\xgboost\training.py in train(params, dtrain, n
um_boost_round, evals, obj, feval, maximize, early_stopping_rounds, evals_re
sult, verbose_eval, xgb_model, callbacks)
    207                             evals=evals,
    208                             obj=obj, feval=feval,
--> 209                             xgb_model=xgb_model, callbacks=callbacks)
    210
    211

~\Anaconda3\lib\site-packages\xgboost\training.py in _train_internal(params,
dtrain, num_boost_round, evals, obj, feval, xgb_model, callbacks)
     72         # Skip the first update if it is a recovery step.
     73         if version % 2 == 0:
---> 74             bst.update(dtrain, i, obj)
     75             bst.save_rabit_checkpoint()
     76             version += 1

~\Anaconda3\lib\site-packages\xgboost\core.py in update(self, dtrain, iterat
ion, fobj)
   1247             _check_call(_LIB.XGBoosterUpdateOneIter(self.handle,
   1248                                                     ctypes.c_int(ite
ration),
-> 1249                                                     dtrain.handle))
   1250         else:
   1251             pred = self.predict(dtrain, training=True)

KeyboardInterrupt:
```

In [ ]:

```
 0.0309563
    0.5519779208831647
    byte_bigram + pxl features



    asm_bow+800_pxl for asm
    0.0356
    0.64397
```

asm_bow+800_pxl for asm

In [ ]:

```
asm_bow=pd.read_csv("asm_bow_size.csv")
```

In [ ]:

```
import joblib
file_name="pxl_df.pkl"
pxl_df=joblib.load(file_name)
```

In [ ]:

```
pxl_df.head()
```

Out[39]:

	ID	pxl0	pxl1	pxl2	pxl3	pxl4	pxl5	pxl
0	jxuKaQ3YtEZANs06iWUf	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
1	GBQtWjTxb4v21NCDAaoI	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
2	ilogAd4QsU38IFBcuwDp	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
3	clkRCKEyz7Irxd2Go9A0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083
4	1NpWfIMxU5ebEJHBAv2G	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.0083

5 rows × 801 columns

In [ ]:

```
asm_bow.head()
```

Out[40]:

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

5 rows × 55 columns

In [176]:

```
asm_tot = pd.merge(left=asm_bow, right=pxl_df,how='inner', left_on='ID', right_on='ID')
```

In [ ]:

```
asm_tot.to_csv("asm_tot_feat.csv")
```

In [ ]:

```
asm_tot=pd.read_csv("asm_tot_feat.csv")
```

In [177]:

```
asm_tot.head()
```

Out[177]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata
0	01kcpwa9k2boxqes5rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.(
1	1e93cpp60rhfnit5qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.(
2	3ekvow2ajzhbtnbcsdfx	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.(
3	3x2ny7iqapbiwdrazqje	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.(
4	46ozzdsskdcfv8h7xwxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.(

5 rows × 554 columns

In [ ]:

```
asm_y=asm_tot["Class"]
```

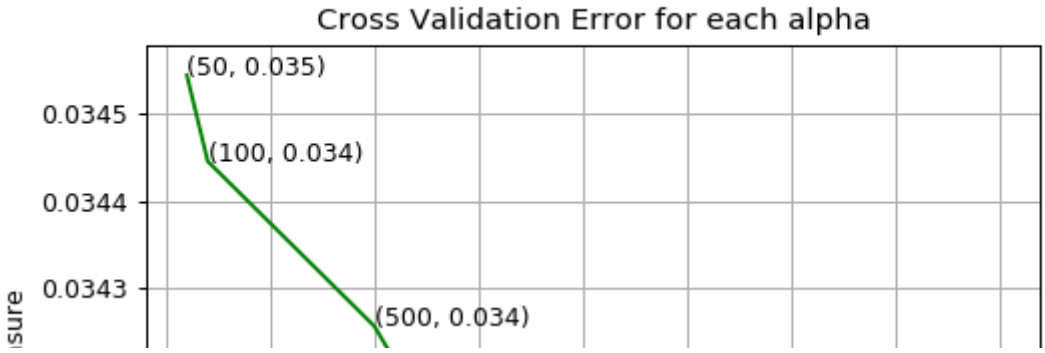
In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(asm_tot.drop(columns=["Class","ID"]), a
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=2)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
log_loss for c =  50 is 0.034545092836676075
log_loss for c =  100 is 0.034445704839329176
log_loss for c =  500 is 0.0342550850414802
log_loss for c =  1000 is 0.033845071278490395
log_loss for c =  2000 is 0.03390032808461197

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



byte_10kbigram+byte_singlegram_size+asm_bow+800_p for asm

In [ ]:

```
asm_tot=pd.read_csv("asm_tot_feat.csv")
```

In [ ]:

```
asm_tot.head()
```

Out[83]:

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

5 rows × 855 columns

In [ ]:

```
top10k_feat.head()
```

Out[21]:

	ID	Class	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	...	f7
0	D2FbBoJAweiY8X0pxTOn	1.0	0.019011	0.000143	0.0	0.0	0.0	0.0	0.0	0.0	...	0.002953
1	ATVo94avrEyjnexXU8W7	3.0	0.150520	0.016392	0.0	0.0	0.0	0.0	0.0	0.0	...	0.014340
2	cf4nzsoCmudt1kwleOTI	1.0	0.042438	0.007489	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000
3	0GUIi7xAlODwZ4YBenNM	2.0	0.031523	0.002266	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000680
4	CdeQsaIqAiMVRjT2Pun6	9.0	0.084499	0.000005	0.0	0.0	0.0	0.0	0.0	0.0	...	0.012085

5 rows × 10259 columns

In [ ]:

```

```

In [ ]:

```
top_asm_byte = pd.merge(left=top_2k_feat, right=asm_tot.drop(columns=['Unnamed: 0']),how='l
```

In [ ]:

```
top_asm_byte.head()
```

Out[27]:

	ID	Class_x	size	?? ??	?? ff	?? fe	?? fd	?? fc	?? fb	?? fa	...	pxl7
0	D2FbBoJAweiY8X0pxTOn	1.0	0.019011	0.000143	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0107
1	ATVo94avrEyjnexXU8W7	3.0	0.150520	0.016392	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0107
2	cf4nzsoCmudt1kwleOTI	1.0	0.042438	0.007489	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0030
3	0GUIi7xAlODwZ4YBenNM	2.0	0.031523	0.002266	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0030
4	CdeQsaIqAiMVRjT2Pun6	9.0	0.084499	0.000005	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0107

5 rows × 2855 columns

In [ ]:

```
top_asm_byte.to_csv('top2k_asm_byte.csv')
```

In [ ]:

```

```

In [ ]:

```
del top_2k_feat
del asm_tot
```

In [ ]:

```
final_y=top_asm_byte['Class_x']
```

In [ ]:

```
top_asm_byte.to_csv('top_asm_byte.csv')
```

In [ ]:

```
top_asm_byte.drop(columns=['Class_x','Class_y','ID'],inplace=True)
```

In [ ]:

```
top_asm_byte.fillna(top_asm_byte.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(top_asm_byte, final_y,stratify=final_y,
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

```
print(X_train.shape)
print(X_cv.shape)
X_test.shape
```

```
(6955, 2852)
(1739, 2852)
```

Out[31]:

```
(2174, 2852)
```

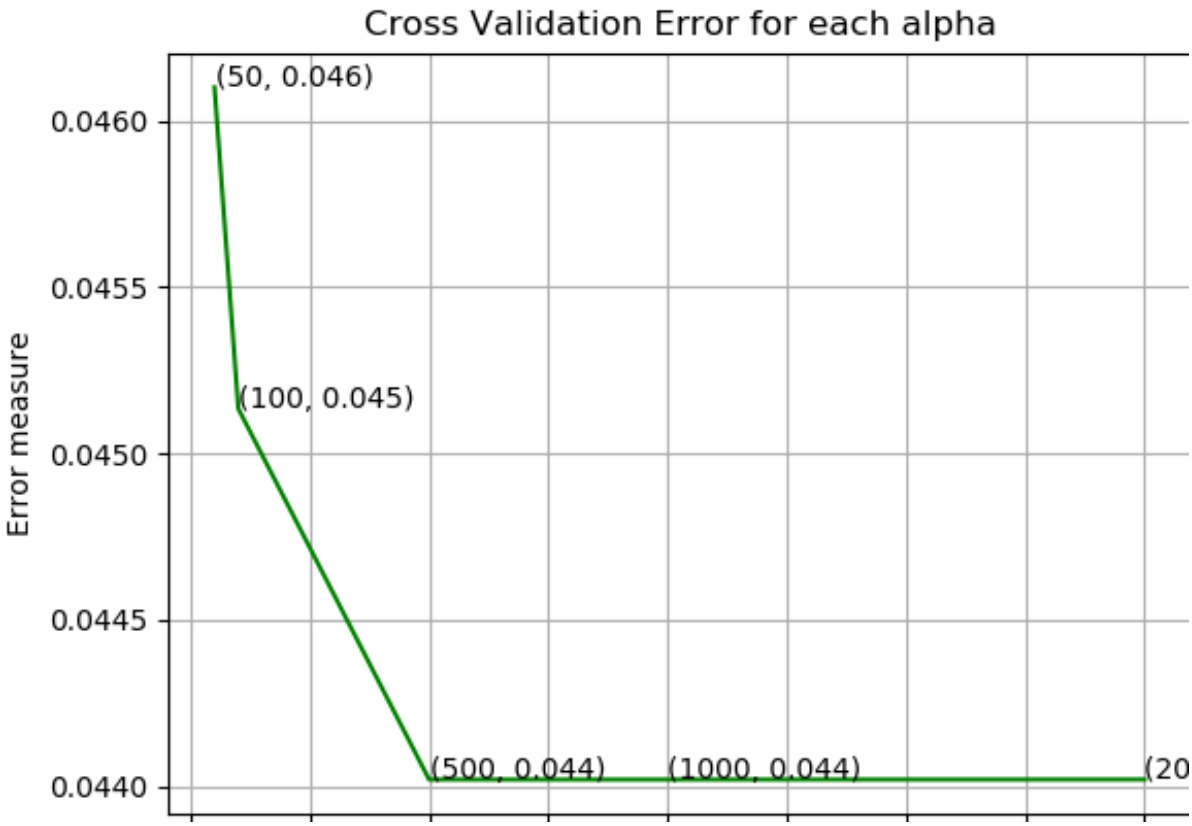
In [ ]:

```
del top_asm_byte
```

In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=2)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
log_loss for c =  50 is 0.046100709122627107
log_loss for c =  100 is 0.04513463510694112
log_loss for c =  500 is 0.044020311659167116
log_loss for c =  1000 is 0.044020295491040445
log_loss for c =  2000 is 0.04402049244640227

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




```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-45-32d3ee562fed> in <module>
     47 x_cfl.fit(X_train,y_train)
     48 sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
---> 49 sig_clf.fit(X_train, y_train)
     50
     51 predict_y = sig_clf.predict_proba(X_train)

~\Anaconda3\lib\site-packages\sklearn\calibration.py in fit(self, X, y, samp
le_weight)
```

```
    187                         sample_weight=base_estimator_sample_weight[t
rain])
    188                 else:
--> 189                     this_estimator.fit(X[train], y[train])
    190
    191                 calibrated_classifier = _CalibratedClassifier(

~\Anaconda3\lib\site-packages\xgboost\sklearn.py in fit(self, X, y, sample_w
eight, base_margin, eval_set, eval_metric, early_stopping_rounds, verbose, x
gb_model, sample_weight_eval_set, callbacks)
    821                                 evals_result=evals_result, obj=obj, fe
val=feval,
    822                                 verbose_eval=verbose, xgb_model=xgb_mo
del,
--> 823                                 callbacks=callbacks)
    824
    825         self.objective = xgb_options["objective"]

~\Anaconda3\lib\site-packages\xgboost\training.py in train(params, dtrain, n
um_boost_round, evals, obj, feval, maximize, early_stopping_rounds, evals_re
sult, verbose_eval, xgb_model, callbacks)
    207                             evals=evals,
    208                             obj=obj, feval=feval,
--> 209                             xgb_model=xgb_model, callbacks=callbacks)
    210
    211

~\Anaconda3\lib\site-packages\xgboost\training.py in _train_internal(params,
dtrain, num_boost_round, evals, obj, feval, xgb_model, callbacks)
     72         # Skip the first update if it is a recovery step.
     73         if version % 2 == 0:
---> 74             bst.update(dtrain, i, obj)
     75             bst.save_rabit_checkpoint()
     76             version += 1

~\Anaconda3\lib\site-packages\xgboost\core.py in update(self, dtrain, iterat
ion, fobj)
   1247             _check_call(_LIB.XGBoosterUpdateOneIter(self.handle,
   1248                                                     ctypes.c_int(ite
ration),
-> 1249                                                     dtrain.handle))
   1250         else:
   1251             pred = self.predict(dtrain, training=True)

KeyboardInterrupt:
```

In [ ]:

```
```

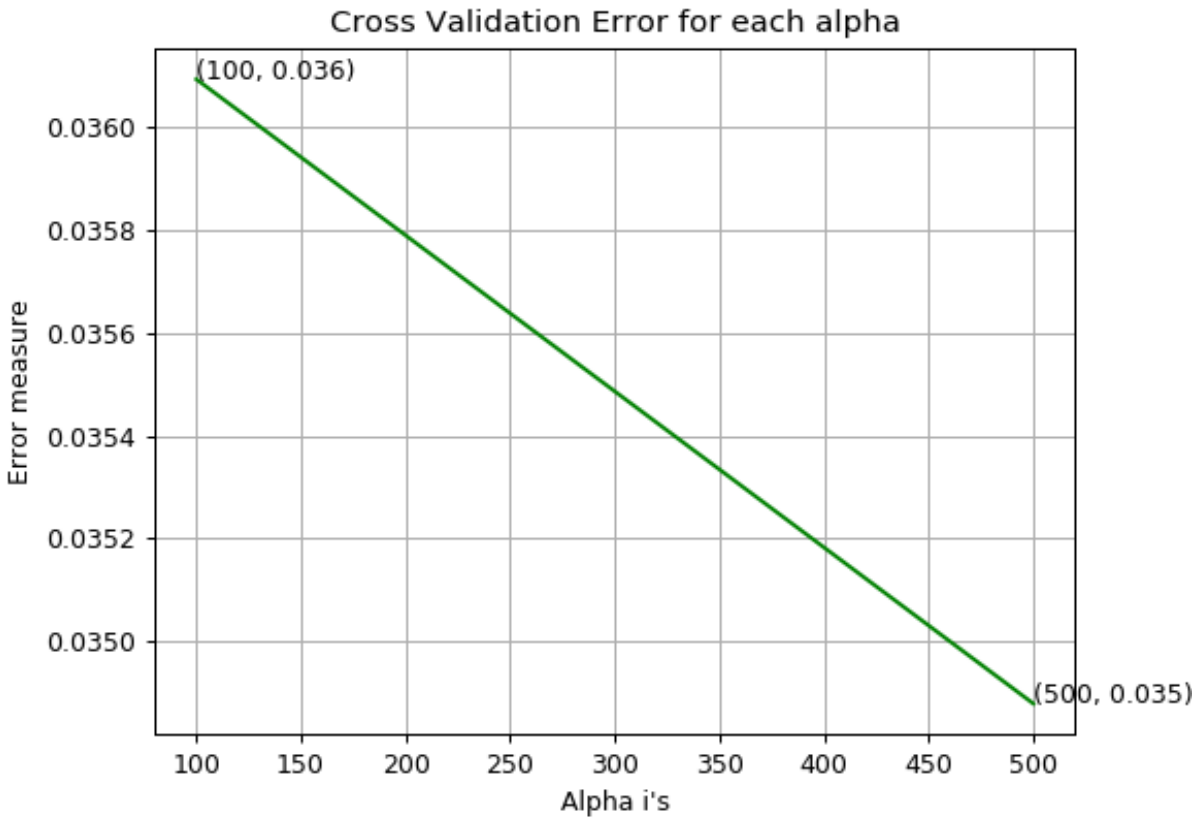
In [ ]:

```
from tqdm import tqdm_notebook as tqdm
```

In [ ]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[100,500]
cv_log_error_array=[]
for i in tqdm (alpha):
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=2)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

```
HBox(children=(FloatProgress(value=0.0, max=2.0), HTML(value='')))


log_loss for c =  100 is 0.03609417788923733
log_loss for c =  500 is 0.034879074752499246

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



In [ ]:

```
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
For values of best alpha =  500 The train log loss is: 0.013480077162740618
For values of best alpha =  500 The cross validation log loss is: 0.03487907
4752499246
For values of best alpha =  500 The test log loss is: 0.030956335637509137

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-f0fddf50fcbc> in <module>
     11 predict_y = sig_clf.predict_proba(X_test)
     12 print('For values of best alpha = ', alpha[best_alpha], "The test lo
g loss is:",log_loss(y_test, predict_y))
---> 13 plot_confusion_matrix(y_test,sig_clf.predict(X_test))

NameError: name 'plot_confusion_matrix' is not defined
```

In [ ]:

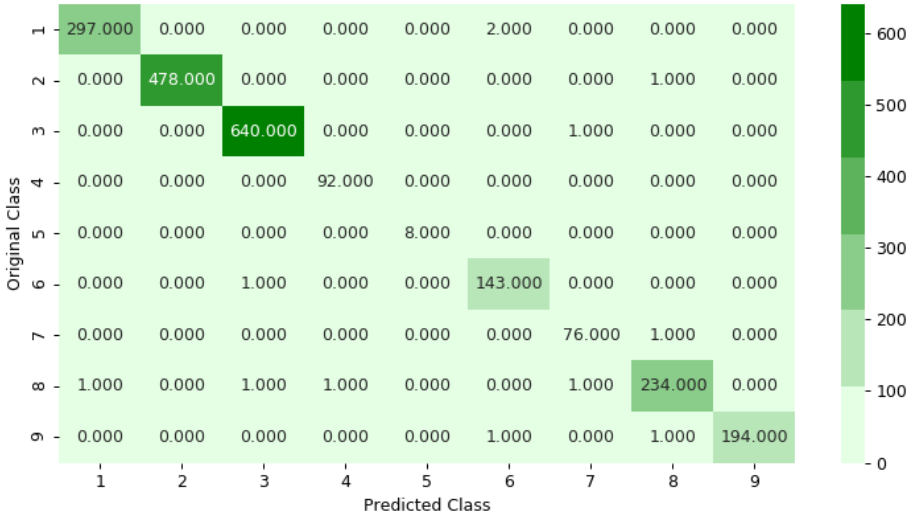
```
0.0309563
   0.5519779208831647
   byte_bigram + pxl features
```

In [ ]:

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

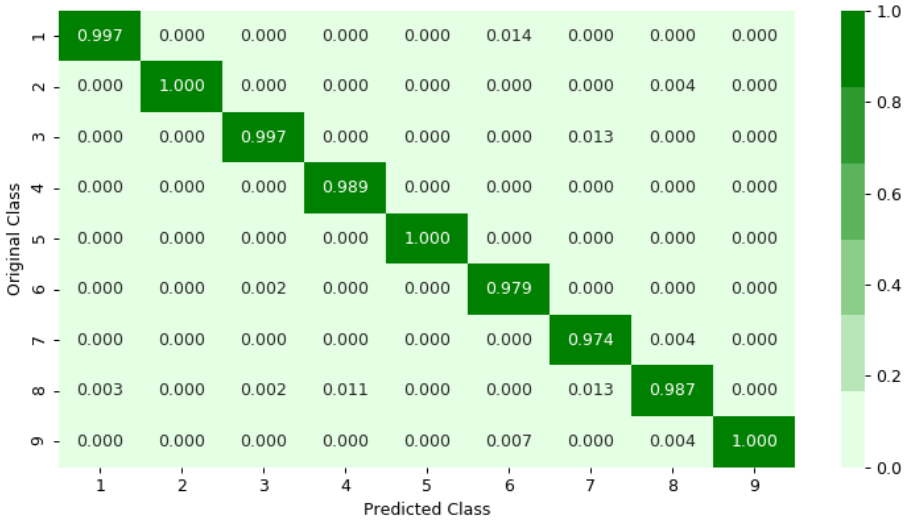
```
Number of misclassified points  0.5519779208831647
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

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



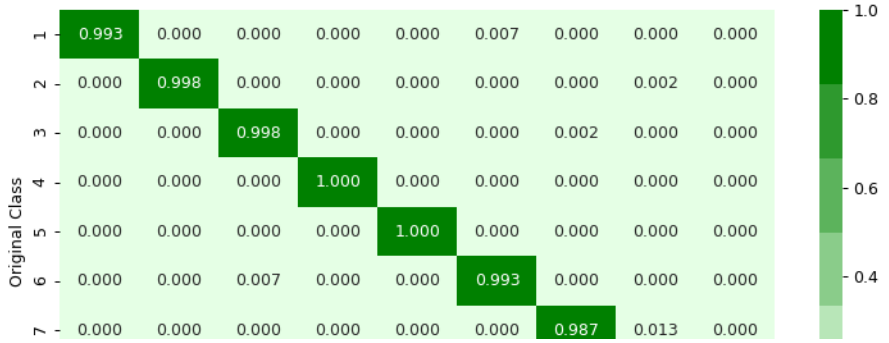

```
---------------------------------------------------- Precision matrix --------
------------------------------------------
```

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




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

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




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

Here we introduced asm bigram + asm_pixel+ asm_gram

In [ ]:

```
from tqdm import tqdm_notebook as tqdm
from nltk.util import ngrams
asm_bigram = ['ID']
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        asm_bigram.append(v + ' ' + opcodes[j])

asm_bigram_file=open('asm_bigram_file.csv','w+')
asm_bigram_file.write(','.join(asm_bigram))
asm_bigram_file.write("\n")

asmFiles=os.listdir("asmFiles")
for file in tqdm(asmFiles):
    temp = dict(zip(asm_bigram, [0]*len(asm_bigram))).copy()
    temp['ID']=str(file)
    all_line=[]
    if(file.endswith(".asm")):
        op_code_str=""
        bi_g=[]
        with codecs.open('asmFiles/'+file,encoding='cp1252',errors ='replace') as asm_file:
            for lines in (asm_file):
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        op_code_str+=li+ " "
            bi_g = [' '.join(x) for x in list(ngrams(op_code_str.split(), 2))]
            for i in bi_g:
                temp[i.lower()]+=1
            asm_features=[]
            asm_features = [str(temp[x]) for x in asm_bigram]
            asm_bigram_file.write(','.join(asm_features))
            asm_bigram_file.write("\n")
            del temp
asm_bigram_file.close()
```

In [ ]:

```
asm_big=pd.read_csv("asm_bigram_file.csv")
```

In [ ]:

```
asm_big.tail()
```

Out[4]:

	ID	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	...	mov c
10863	5C0Ke7rXRtb9smU3Qkho.asm	3	64	0	6	1	43	0	0	0	...	
10864	5mThHDbEpSQAYq3F1xvz.asm	9	140	0	62	2	36	0	0	5	...	
10865	H7CpUha5E4m9YzjxD0Z1.asm	0	54	0	6	0	5	0	0	0	...	
10866	8DxQhtuOKPRZAjGy6Y3E.asm	1	3	0	0	1	1	1	0	1	...	
10867	BMmCz4hZdgYjcbQD5oai.asm	0	0	0	1	0	0	0	0	0	...	

5 rows × 677 columns

In [ ]:

```
asm_big['ID']=asm_big['ID'].map(lambda x : x.rstrip('.asm'))
```

In [ ]:

```
asm_big.columns[1:]
```

Out[23]:

```
Index(['jmp jmp', 'jmp mov', 'jmp retf', 'jmp push', 'jmp pop', 'jmp xor',
       'jmp retn', 'jmp nop', 'jmp sub', 'jmp inc',
       ...
       'movzx cmp', 'movzx call', 'movzx shl', 'movzx ror', 'movzx rol',
       'movzx jnb', 'movzx jz', 'movzx rtn', 'movzx lea', 'movzx movzx'],
      dtype='object', length=676)
```

In [ ]:

```
asm_big.iloc[0:4,85:585]
```

Out[52]:

	push retn	push nop	push sub	push inc	push dec	push add	push imul	push xchg	push or	push shr	...	jz retf	jz push	jz pop	jz xor	i
0	0	0	6	0	0	7	0	0	1	0	...	0	17	0	1	
1	0	5	0	7	6	10	0	0	4	1	...	0	1	4	0	
2	0	0	4	0	0	6	0	0	4	0	...	0	0	0	0	
3	0	0	3	0	0	0	0	0	0	0	...	0	0	0	28	

4 rows × 500 columns

In [ ]:

```
asm_big.columns[1:]
```

Out[74]:

```
Index(['jmp jmp', 'jmp mov', 'jmp retf', 'jmp push', 'jmp pop', 'jmp xor',
       'jmp retn', 'jmp nop', 'jmp sub', 'jmp inc',
       ...
       'movzx cmp', 'movzx call', 'movzx shl', 'movzx ror', 'movzx rol',
       'movzx jnb', 'movzx jz', 'movzx rtn', 'movzx lea', 'movzx movzx'],
      dtype='object', length=676)
```

In [ ]:

```
asm_big.iloc[:,1:]
```

Out[75]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	jmp inc	...	movzx cmp	movzx call	movzx shl	mov
0	0	42	0	5	0	3	0	0	0	0	...	0	0	0	
1	31	8	1	10	3	0	0	2	0	3	...	0	0	0	
2	46	101	0	10	0	0	0	0	0	0	...	14	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	28	9	0	
4	12	398	0	56	2	10	0	11	24	9	...	11	0	4	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10863	3	64	0	6	1	43	0	0	0	0	...	0	0	0	
10864	9	140	0	62	2	36	0	0	5	17	...	5	0	0	
10865	0	54	0	6	0	5	0	0	0	0	...	0	0	0	
10866	1	3	0	0	1	1	1	0	1	1	...	0	0	0	
10867	0	0	0	1	0	0	0	0	0	0	...	0	0	0	

10868 rows × 676 columns

In [ ]:

```
with open("asmbigram_columns.txt","wb") as fp:
    asmbigram_columns=pickle.dump(asm_big.columns[1:],fp)
```

In [ ]:

```

```

In [ ]:

```
from scipy import sparse
dense_matrix=np.array(asm_big.iloc[:,1:]) #pandas data frame to numpy matrix
sparse_matrix=sparse.csr_matrix(dense_matrix) #numpy matrix to sparse
from sklearn.preprocessing import normalize
asmbigram_process_vect= normalize(sparse_matrix, axis = 0)
import pickle
with open("asmbigram_columns.txt","rb") as fp:
    asmbigram_columns=pickle.load(fp)
asmbigram_process_dense=asmbigram_process_vect.todense()
del asmbigram_process_vect
import pandas as pd
asmbigram_process_vect_df=pd.DataFrame(asmbigram_process_dense[0:,0:],columns=asmbigram_col

with open("asmbigram_ID.txt","rb") as fp:
    asmbigram_ID=pickle.load(fp)
asmbigram_process_vect_df.insert(0,column='ID',value=asmbigram_ID)
```

In [204]:

```
asmbigram_process_vect_df=pd.read_csv('asmbigram_process_vect_df.csv')
```

In [205]:

```
asmbigram_process_vect_df.head()
```

Out[205]:

	Unnamed: 0	ID	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp
0	0	8VU3wPKq2lBY5bvnHQMS	0.000000	0.000852	0.000000	0.000350	0.000000	0.00
1	1	ErYMOiwT8h3yqvfkm9Pu	0.002241	0.000162	0.001467	0.000700	0.001123	0.00
2	2	dKNk2oDiqevnRTsgCmG0	0.003326	0.002049	0.000000	0.000700	0.000000	0.00
3	3	dFUABXPZzn3mL06OEjek	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
4	4	EuH31db0lYoVintDaL9v	0.000868	0.008073	0.000000	0.003923	0.000749	0.00

5 rows × 678 columns

asm_bigram+top_800_bytepxl_features

In [ ]:

```
asm_tot.head()
```

Out[107]:

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

5 rows × 855 columns

In [211]:

```
asmbigram_process_vect_df['ID']=asmbigram_process_vect_df['ID'].str.lower()
asm_tot['ID']=asm_tot['ID'].str.lower()
```

In [254]:

```
#asmbigram_process_vect_df.to_csv('asmbigram_process_vect_df.csv')
top_asm_uptobig = pd.merge(left=asmbigram_process_vect_df, right=asm_tot,how='left', left_o
```

In [231]:

```
#asmbigram_process_vect_df.to_csv('asmbigram_process_vect_df.csv')
fin_feat = pd.merge(left=fin_bytebigram,right=top_asm_uptobig ,how='left', left_on='ID', ri
```

In [234]:

```
final_y=fin_feat['Class']
```

In [256]:

```
top_asm_uptobig.drop(columns=['Unnamed: 0','Class','ID'],inplace=True)
```

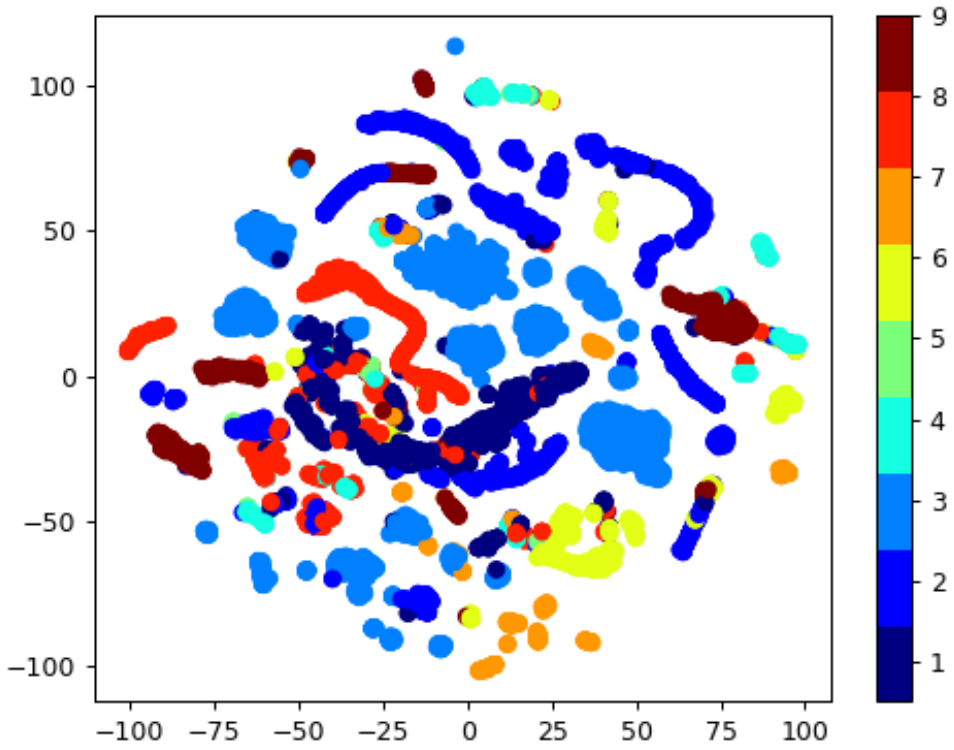
In [246]:

```
fin_feat.fillna(fin_feat.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
```

In [ ]:

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

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



In [250]:

```
fin_feat.fillna(fin_feat.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
X_train, X_test, y_train, y_test = train_test_split(fin_feat, final_y,stratify=final_y,test
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [251]:

```
X_train.shape
```

Out[251]:

```
(6955, 1528)
```

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=500,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

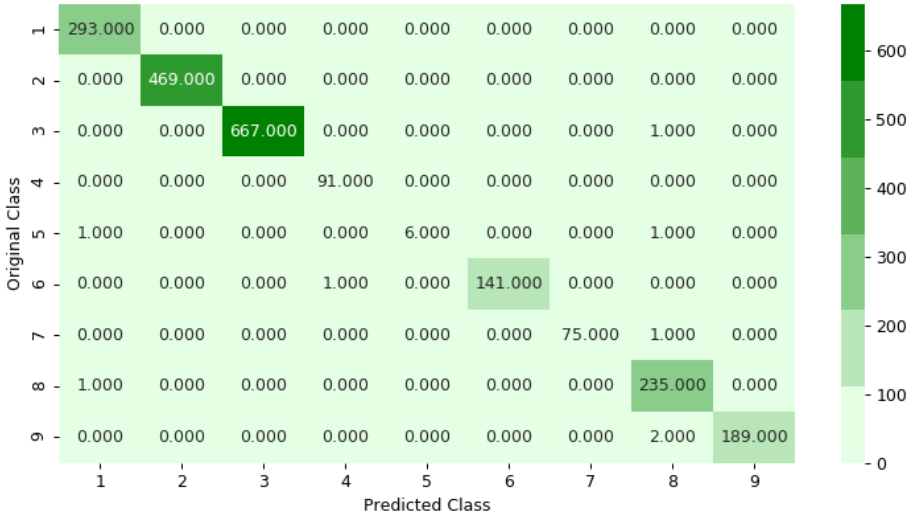
```
The train log loss is: 0.011677288128305258
The cross validation log loss is: 0.029473259079896015
The test log loss is: 0.02063813700116953
```

In [ ]:

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

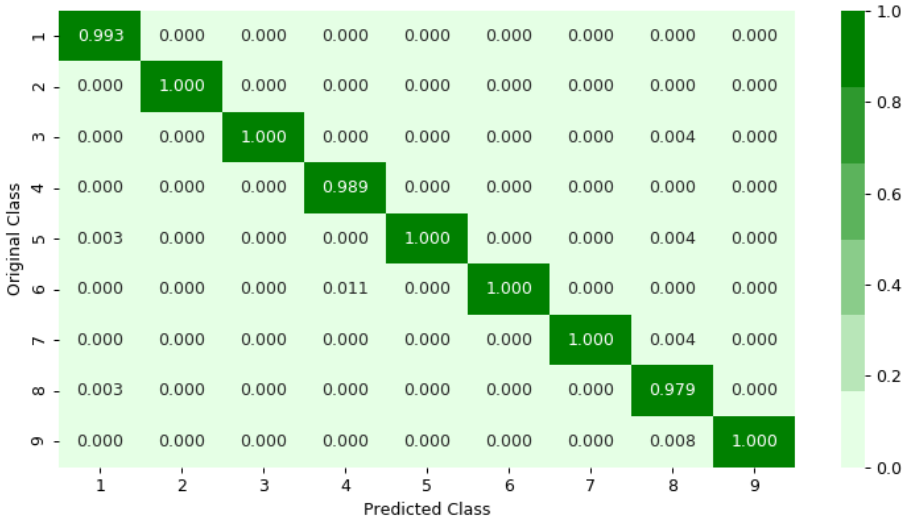
```
Number of misclassified points  0.36798528058877644
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

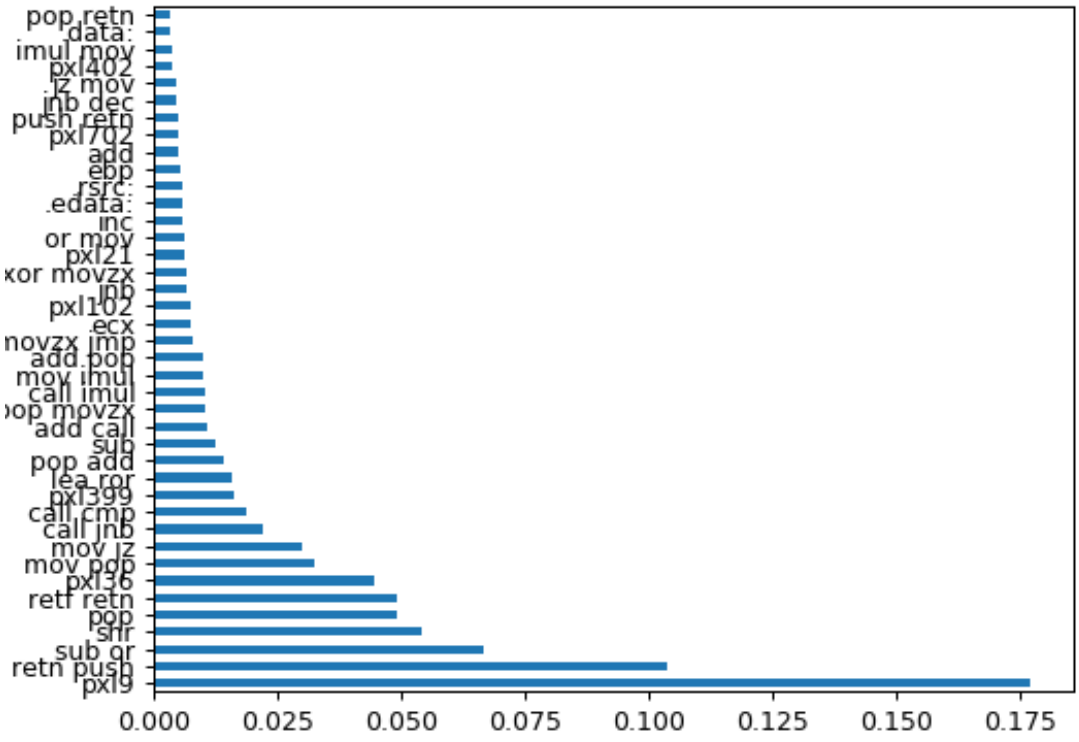
<IPython.core.display.Javascript object>


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

In [ ]:

```
feat_importances = pd.Series(r_cfl.feature_importances_, index=X_train.columns)
feat_importances.nlargest(40).plot(kind='barh')
plt.show()
```

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



we coulde see here pxl_9 and retn_push has sub_gr has greater importance values

now we try to make use of asm_tri_grams here

In [ ]:

```
# due to computation and large files I have run these files on the GCP and downloaded only
from tqdm import tqdm_notebook as tqdm
from nltk.util import ngrams
asm_trigram = ['ID']
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        for k in range(0, len(opcodes)):
            asm_trigram.append(v + ' ' + opcodes[j]+ ' ' + opcodes[k])

asm_trigram_file=open('asm_trigram_file.csv','w+')
asm_trigram_file.write(','.join(asm_trigram))
asm_trigram_file.write("\n")

asmFiles=os.listdir("asmFiles")
for file in tqdm(asmFiles):
    temp = dict(zip(asm_trigram, [0]*len(asm_trigram))).copy()
    temp['ID']=str(file)
    all_line=[]
    if(file.endswith(".asm")):
        op_code_str=""
        tri_g=[]
        with codecs.open('asmFiles/'+file,encoding='cp1252',errors ='replace') as asm_file:
            for lines in (asm_file):
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        op_code_str+=li+ " "
            tri_g = [' '.join(x) for x in list(ngrams(op_code_str.split(), 3))]
            for i in tri_g:
                temp[i.lower()]+=1
            asm_features=[]
            asm_features = [str(temp[x]) for x in asm_trigram]
            asm_trigram_file.write(','.join(asm_features))
            asm_trigram_file.write("\n")
            del temp
asm_trigram_file.close()
```

In [ ]:

```
dense_matrix=np.array(df_bigram.iloc[:,1:]) #pandas data frame to numpy matrix
sparse_matrix=sparse.csr_matrix(dense_matrix) #numpy matrix to sparse
```

In [ ]:

```
asm_tri=pd.read_csv("asm_trigram_file.csv")
```

In [ ]:

```
asmtrigram_columns=asm_tri.columns[1:]
asmtrigram_ID=asm_tri['ID']
```

In [ ]:

```
asm_tri.tail()
```

Out[65]:

	ID	jmp jmp jmp	jmp jmp mov	jmp jmp retf	jmp jmp push	jmp jmp pop	jmp jmp xor	jmp jmp retn	jmp jmp nop	jmp jmp sub	...	mov mov c
10863	5C0Ke7rXRtb9smU3Qkho.asm	0	1	0	1	0	0	0	0	0	...	
10864	5mThHDbEpSQAYq3F1xvz.asm	2	7	0	0	0	0	0	0	0	...	
10865	H7CpUha5E4m9YzjxD0Z1.asm	0	0	0	0	0	0	0	0	0	...	
10866	8DxQhtuOKPRZAjGy6Y3E.asm	0	0	0	0	0	0	0	0	1	...	
10867	BMmCz4hZdgYjcbQD5oai.asm	0	0	0	0	0	0	0	0	0	...	

5 rows × 17577 columns

In [ ]:

```
with open("asmtrigram_columns.txt","wb") as fp:
    pickle.dump(asmtrigram_columns,fp)
with open("asmtrigram_ID.txt","wb") as fp:
    pickle.dump(asmtrigram_ID,fp)
```

In [ ]:

```
from scipy import sparse
dense_matrix=np.array(asm_tri.iloc[:,1:]) #pandas data frame to numpy matrix
sparse_matrix=sparse.csr_matrix(dense_matrix) #numpy matrix to sparse
from sklearn.preprocessing import normalize
asmtrigram_process_vect= normalize(sparse_matrix, axis = 0)
import pickle
with open("asmtrigram_columns.txt","rb") as fp:
    asmtrigram_columns=pickle.load(fp)
asmtrigram_process_dense=asmtrigram_process_vect.todense()
del asmtrigram_process_vect
import pandas as pd
asmtrigram_process_vect_df=pd.DataFrame(asmtrigram_process_dense[0:,0:],columns=asmtrigram_

with open("asmtrigram_ID.txt","rb") as fp:
    asmtrigram_ID=pickle.load(fp)
asmtrigram_process_vect_df.insert(0,column='ID',value=asmtrigram_ID)
```

In [ ]:

```
asmtrigram_process_vect_df['ID']=asmtrigram_process_vect_df['ID'].map(lambda x : x.rstrip('
```

In [ ]:

```

```

In [ ]:

```
top_asm_uptobig = pd.merge(left=asmtrigram_process_vect_df, right=asm_tot.drop(columns=['Un
```

In [ ]:

```
top_asm_uptobig.head()
```

Out[157]:

	ID	jmp jmp jmp	jmp jmp mov	jmp jmp retf	jmp jmp push	jmp jmp pop	jmp jmp xor	jmp jmp retn	jmp jmp nop	j
0	8VU3wPKq2lBY5bvnHQMS	0.000000	0.0000	0.0	0.000000	0.000000	0.000000	0.0	0.0	0
1	ErYMOiwT8h3yqvfkm9Pu	0.001310	0.0018	0.0	0.011823	0.001741	0.000000	0.0	0.0	0
2	dKNk2oDiqevnRTsgCmG0	0.002948	0.0027	0.0	0.004729	0.000000	0.000000	0.0	0.0	0
3	dFUABXPZzn3mL06OEjek	0.000000	0.0000	0.0	0.000000	0.000000	0.000000	0.0	0.0	0
4	EuH31db0lYoVintDaL9v	0.000164	0.0018	0.0	0.009458	0.000000	0.002563	0.0	0.0	0

5 rows × 18430 columns

In [ ]:

```
top_asm_uptobig["Class"].value_counts()
```

Out[156]:

```
3.0    2815
2.0    2346
1.0    1464
8.0    1178
9.0     956
6.0     709
4.0     455
7.0     380
5.0      39
Name: Class, dtype: int64
```

In [ ]:

```
final_y=top_asm_uptobig['Class']
```

In [ ]:

```
top_asm_uptobig.drop(columns=["ID","Class"],inplace=True)
```

In [ ]:

```
top_asm_uptobig.shape
```

Out[160]:

```
(10868, 18428)
```

In [ ]:

```
top_asm_uptobig.fillna(top_asm_uptobig.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
```

In [ ]:

```
final_y.value_counts()
```

Out[170]:

```
3.0    2138
2.0    1501
1.0     937
8.0     754
9.0     612
6.0     454
4.0     291
7.0     243
5.0      25
Name: Class, dtype: int64
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(top_asm_uptobig, final_y,stratify=final
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

```
X_train.shape
```

Out[166]:

```
(6955, 18428)
```

In [ ]:

```
#r_cfl=XGBClassifier(n_estimators=500,max_depth= 3,learning_rate= 0.1,
# colsample_bytree= 1,random_state=42,n_jobs=-1)
#r_cfl.fit(X_train ,y_train )
import joblib
file_name="asm_xgb_model.pkl"
xgb_model=joblib.dump(r_cfl,file_name)
```

In [ ]:

```
file_name="asm_xgb_model.pkl"
r_cfl=joblib.load(file_name)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

In [ ]:

```
y_train.replace(4.018372, 4.0,inplace=True)
```

In [ ]:

```
trigramasm_top400.pkl
pxl_top400.pkl
bytebigram_top300_impfeat.pkl
bigramasm_top250.pkl
asm_bow
```

In [ ]:

```
import joblib
filename = "bigramasm_top250.pkl"
bigramasm_top250=joblib.load(filename)
bigramasm_top250=bigramasm_top250.insert(0,'ID')
```

In [ ]:

```
asm_big=pd.read_csv("asm_procbidf.csv")
```

In [ ]:

```
asmbig250=asm_big[bigramasm_top250]
```

In [ ]:

```
asmbig250.head()
```

Out[72]:

	ID	retn push	mov mov	mov pop	add pop	call cmp	cmp jnb	mov j
0	8VU3wPKq2lBY5bvnHQMS	0.000958	0.001142	0.001792	0.000183	0.000598	0.003909	0.001(
1	ErYMOiwT8h3yqvfkm9Pu	0.000030	0.000011	0.000045	0.000457	0.000000	0.000000	0.000
2	dKNk2oDiqevnRTsgCmG0	0.001107	0.001101	0.001021	0.000000	0.000435	0.003909	0.001(
3	dFUABXPZzn3mL06OEjek	0.001077	0.000000	0.000000	0.000000	0.000544	0.000000	0.000(
4	EuH31db0IYoVintDaL9v	0.002753	0.007132	0.001474	0.005570	0.001522	0.007166	0.010(

5 rows × 251 columns

In [ ]:

```
import joblib
filename = "pxl_df.pkl"
pxl_df=joblib.load(filename)
```

In [ ]:

```
import joblib
filename = "pxl_top400.pkl"
pxl_top400=joblib.load(filename)
pxl_top400=pxl_top400.insert(0,'ID')
```

In [ ]:

```

```

In [ ]:

```
pxl_400=pxl_df[pxl_top400]
```

In [ ]:

```
pxl_400.head()
```

Out[13]:

	ID	pxl693	pxl97	pxl0	pxl2	pxl9	pxl13	pxl
0	jxuKaQ3YtEZANs06iWUf	0.009472	0.007751	0.010268	0.010268	0.007913	0.008107	0.0081
1	GBQtWjTxb4v21NCDAaoI	0.009472	0.007155	0.010268	0.010268	0.007913	0.008107	0.0081
2	ilogAd4QsU38IFBcuwDp	0.009472	0.008049	0.010268	0.010268	0.007913	0.008107	0.0081
3	clkRCKEyz7Irxd2Go9A0	0.009472	0.007155	0.010268	0.010268	0.007913	0.008107	0.0081
4	1NpWfIMxU5ebEJHBAv2G	0.009472	0.007155	0.010268	0.010268	0.007913	0.008107	0.0081

5 rows × 401 columns

In [ ]:

```
bytebigram_top300_impfeat.pkl
```

In [83]:

```
import joblib
filename = "bytebigram_top300_impfeat.pkl"
bytebigram_top300=joblib.load(filename)
bytebigram_top300=bytebigram_top300.insert(0,'ID')
```

In [89]:

```
np.array(bytebigram_top300)
```

Out[89]:

```
array(['ID', '52 a1', '00 da', '41 44', '59 85', 'c4 0c', '00 ad',
       '8a 46', '85 c0', '02 c3', '24 8d', 'ff 55', '00 4a', '56 e8',
       '?? ??', 'c2 0c', '6e 72', '8b fe', '81 bd', '81 7d', '40 00',
       '74 0c', '14 8d', '83 c4', 'f6 83', '53 8b', '33 32', '2b c7',
       '4e 47', '00 65', '50 e8', 'e5 81', '00 ??', '42 69', '?? 6c',
       '38 00', 'c7 41', '77 77', '0f 84', '61 6f', '4a 00', '50 41',
       '01 3b', '4a ff', '0f 8b', 'f6 59', 'f1 35', '42 00', '0f f0',
       '2c 01', '0f b7', '81 25', '3c 04', '57 bb', '04 56', '8a 06',
       'e2 73', '81 f9', '49 b3', '85 44', 'e4 04', '68 03', '5d c3',
       'ee 8f', 'c7 45', '73 6e', '27 d1', 'c4 64', '16 1c', '69 00',
       '72 20', '8d 89', '44 ??', '00 ff', '74 03', '06 b7', '00 eb',
       '01 10', '49 4e', '00 97', 'c1 53', '67 6c', '00 89', '2d 63',
       '0f 85', '01 8b', '53 79', '50 6a', '00 01', '68 2e', '00 13',
       '83 f6', 'size', '11 c8', 'ff 6a', '4e 65', '6a c7', '6a b0',
       '16 41', 'ef b9', '10 89', '06 b1', '8a 51', '64 a1', '57 77',
       '00 17', '89 48', '58 ??', '81 05', '61 65', '00 ec', '00 04',
       '77 70', '24 eb', '35 f1', '76 20', '03 89', 'c3 00', 'a5 0f',
       '42 bb', '27 b9', 'ca 19', '6d 62', '88 04', 'e7 79', '06 01',
       '36 e6', '56 8b', 'c7 05', '95 88', '8b c0', '77 07', '3e 0d',
       'fc 35', '65 37', '6a 99', 'ff 95', '81 3d', '31 79', '3e f1',
       '58 c9', 'ff 84', 'cb 00', '17 de', '66 ab', '31 c1', '20 6f',
       '0d 2f', '89 ff', '00 59', '04 89', '6b c8', '8b 00', '74 63',
       '39 d1', 'fa 15', '4c 00', '89 71', 'a9 1f', '56 1c', 'f6 6d',
       '52 00', 'ff 25', '7d 70', '18 52', '5a 41', '4e 6f', 'e2 4e',
       '61 70', '73 63', '50 72', '64 73', '83 8d', '96 b7', '1c 11',
       '7a c0', 'ff ac', '24 04', '6d f9', 'c3 8b', 'cf fc', '4f 00',
       '9d ab', 'ba 8d', '66 89', '5f 5e', '80 7d', '59 af', 'c7 85',
       '8d 00', '00 00', 'b5 05', '8b ff', 'ff 00', '00 8d', 'f5 35',
       '3d 22', '56 a6', '4a 3c', '07 70', 'a5 a5', '75 61', '2f 06',
       '8d 05', '5a 78', '00 6a', '61 75', '8b 8b', '82 a3', '63 65',
       '6a 59', '2b 4a', '56 6a', '72 6c', 'db 8d', 'ae 49', 'ad de',
       '05 53', '20 4a', '00 57', '?? 00', '10 8b', 'b7 25', 'fc 70',
       '2b d4', '39 5d', 'f0 64', '52 89', '77 c8', '46 75', 'f1 6f',
       '75 db', 'e5 5d', 'ff ff', '00 c9', 'fb d1', 'fb f8', '12 ce',
       '44 34', '93 2f', '6f 63', '41 00', '00 6f', 'fd ff', '7c fc',
       '52 e8', '08 c4', '83 7d', '41 90', '27 91', 'd3 de', '8a 43',
       '48 ec', '81 4d', '6a c4', '69 6f', '02 8b', 'ff 8b', '83 c3',
       '8e 47', '2e 65', 'ee fc', '44 49', '6a ac', '00 09', '65 d3',
       '6a 6a', '8b 2d', '30 22', '00 16', 'f9 cb', '50 c5', '8b c1',
       'ff 72', '49 6e', '4e c0', '8f 05', '80 87', '6b 68', 'f4 3a',
       '12 39', '83 ea', '8b 90', 'ff 80', '00 68', 'cf 91', '01 00',
       '83 ff', '05 fc', '48 27', 'c6 45', '15 81', '8d 94', '01 88',
       '1e 00', '15 5e', '9e ab', '66 81', 'c4 38', '2b 0d', 'a2 f0'],
      dtype=object)
```

In [92]:

```
tot_feat.head()
```

Out[92]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 (
0	D2FbBoJAweiY8X0pxTOn	0.005062	0.017984	0.002374	0.003711	0.004159	0.001102	0.0007
1	ATVo94avrEyjnexXU8W7	0.000974	0.000443	0.000343	0.001211	0.000205	0.000962	0.0011
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
3	0GUli7xAlODwZ4YBenNM	0.002237	0.001810	0.001739	0.001693	0.001989	0.000210	0.0000
4	CdeQsaIqAiMVRjT2Pun6	0.002025	0.006216	0.002031	0.002702	0.007431	0.010027	0.0088

5 rows × 66052 columns

In [ ]:

```
byte_top300[[c for c in df.columns if c in lst]]
```

In [ ]:

```
byte_top300.head()
```

In [ ]:

```
import joblib
filename = "trigramasm_top400.pkl"
trigramasm_top400=joblib.load(filename)
trigramasm_top400=trigramasm_top400.insert(0,'ID')
```

In [ ]:

```
trigramasm_400df=asmtrigram_process_vect_df[trigramasm_top400].head()
```

In [ ]:

```
trigramasm_400df+asmbig250
```

Out[69]:

	ID	add pop push	mov mov add	pop add pop	mov sub lea	mov mov mov	call add pop	retn push mov
0	8VU3wPKq2lBY5bvnHQMS	0.000000	0.001894	0.0	0.001504	0.000773	0.000125	0.000723
1	ErYMOiwT8h3yqvfkm9Pu	0.000883	0.000000	0.0	0.000000	0.000000	0.000000	0.000000
2	dKNk2oDiqevnRTsgCmG0	0.000000	0.002907	0.0	0.000000	0.000980	0.000000	0.001672
3	dFUABXPZzn3mL06OEjek	0.000000	0.000000	0.0	0.002506	0.000000	0.000000	0.001626
4	EuH31db0IYoVintDaL9v	0.000000	0.007840	0.0	0.005514	0.006773	0.001756	0.003388

5 rows × 401 columns

In [ ]:

```
asmbig250.head()
```

Out[73]:

	ID	retn push	mov mov	mov pop	add pop	call cmp	cmp jnb	mov j
0	8VU3wPKq2lBY5bvnHQMS	0.000958	0.001142	0.001792	0.000183	0.000598	0.003909	0.001(
1	ErYMOiwT8h3yqvfkm9Pu	0.000030	0.000011	0.000045	0.000457	0.000000	0.000000	0.000
2	dKNk2oDiqevnRTsgCmG0	0.001107	0.001101	0.001021	0.000000	0.000435	0.003909	0.001(
3	dFUABXPZzn3mL06OEjek	0.001077	0.000000	0.000000	0.000000	0.000544	0.000000	0.000(
4	EuH31db0IYoVintDaL9v	0.002753	0.007132	0.001474	0.005570	0.001522	0.007166	0.010(

5 rows × 251 columns

In [ ]:

```
pxl_400+trigramasm_400df+asmbig250
```

Out[44]:

	ID	pxl693	pxl97	pxl0	pxl2	pxl9	pxl13	pxl
0	jxuKaQ3YtEZANs06iWUf	0.009472	0.007751	0.010268	0.010268	0.007913	0.008107	0.0081
1	GBQtWjTxb4v21NCDAaoI	0.009472	0.007155	0.010268	0.010268	0.007913	0.008107	0.0081
2	ilogAd4QsU38IFBcuwDp	0.009472	0.008049	0.010268	0.010268	0.007913	0.008107	0.0081
3	clkRCKEyz7Irxd2Go9A0	0.009472	0.007155	0.010268	0.010268	0.007913	0.008107	0.0081
4	1NpWfIMxU5ebEJHBAv2G	0.009472	0.007155	0.010268	0.010268	0.007913	0.008107	0.0081

5 rows × 401 columns

In [ ]:

```
byte_top300+pxl_400+trigramasm_400df+asmbig250
```

Out[45]:

	ID	52 a1	00 da	41 44	59 85	c4 0c	00 ad	8a
0	D2FbBoJAweiY8X0pxTOn	0.000489	0.000157	0.003928	0.000053	0.001041	0.000152	0.0000
1	ATVo94avrEyjnexXU8W7	0.002935	0.000295	0.000680	0.000747	0.000946	0.000836	0.0067
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
3	0GUIi7xAlODwZ4YBenNM	0.000000	0.000039	0.000151	0.006298	0.005630	0.000076	0.0132
4	CdeQsaIqAiMVRjT2Pun6	0.001467	0.003286	0.000906	0.005444	0.000520	0.006159	0.0023

5 rows × 301 columns

In [ ]:

```
asm_bow=pd.read_csv('asm_bowdf.csv')
```

In [ ]:

```
asm_bow.head()
```

Out[48]:

	Unnamed: 0	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:
0	0	8VU3wPKq2lBY5bvnHQMS	0.011483	0.000119	0.0	0.000680	1.526236e-06	0.0
1	1	ErYMOiwT8h3yqvfkm9Pu	0.010274	0.000133	0.0	0.000551	1.312028e-06	0.0
2	2	dKNk2oDiqevnRTsgCmG0	0.010274	0.000068	0.0	0.000268	1.151371e-06	0.0
3	3	dFUABXPZzn3mL06OEjek	0.010274	0.000036	0.0	0.000230	5.087454e-07	0.0
4	4	EuH31db0IYoVintDaL9v	0.010274	0.000064	0.0	0.000316	4.551933e-06	0.0

5 rows × 55 columns

In [ ]:

```
top_asm = pd.merge(left=asmbig250, right=trigramasm_400df,how='left', left_on='ID', right_o
```

In [ ]:

```
top_asm.head()
```

Out[75]:

	ID	retn push	mov mov	mov pop	add pop	call cmp	cmp jnb	mov j
0	8VU3wPKq2lBY5bvnHQMS	0.000958	0.001142	0.001792	0.000183	0.000598	0.003909	0.001(
1	ErYMOiwT8h3yqvfkm9Pu	0.000030	0.000011	0.000045	0.000457	0.000000	0.000000	0.000
2	dKNk2oDiqevnRTsgCmG0	0.001107	0.001101	0.001021	0.000000	0.000435	0.003909	0.001(
3	dFUABXPZzn3mL06OEjek	0.001077	0.000000	0.000000	0.000000	0.000544	0.000000	0.000(
4	EuH31db0lYoVintDaL9v	0.002753	0.007132	0.001474	0.005570	0.001522	0.007166	0.010(

5 rows × 651 columns

In [ ]:

```
top_asm = pd.merge(left=top_asm, right=asm_bow.drop(columns=['Unnamed: 0']),how='left', lef
```

In [ ]:

```
top_asm.head()
```

Out[77]:

	ID	retn push	mov mov	mov pop	add pop	call cmp	cmp jnb	mov j
0	8VU3wPKq2lBY5bvnHQMS	0.000958	0.001142	0.001792	0.000183	0.000598	0.003909	0.001(
1	ErYMOiwT8h3yqvfkm9Pu	0.000030	0.000011	0.000045	0.000457	0.000000	0.000000	0.000
2	dKNk2oDiqevnRTsgCmG0	0.001107	0.001101	0.001021	0.000000	0.000435	0.003909	0.001(
3	dFUABXPZzn3mL06OEjek	0.001077	0.000000	0.000000	0.000000	0.000544	0.000000	0.000(
4	EuH31db0lYoVintDaL9v	0.002753	0.007132	0.001474	0.005570	0.001522	0.007166	0.010(

5 rows × 704 columns

In [ ]:

```
byte_top300
pxl_400
```

In [ ]:

```
top_byte = pd.merge(left=byte_top300, right=pxl_400,how='left', left_on='ID', right_on='ID'
```

In [ ]:

```
top_byte.to_csv("top_byte.csv")
```

In [ ]:

```
top_asm.to_csv("top_asm.csv")
```

In [ ]:

```
merge_asm_byte=pd.merge(left=top_asm, right=top_byte,how='left', left_on='ID', right_on='ID
```

In [ ]:

```
merge_asm_byte.shape
```

Out[94]:

```
(10868, 1404)
```

In [ ]:

```
merge_asm_byte
```

Out[95]:

	ID	retn push	mov mov	mov pop	add pop	call cmp	cmp jnb	n
0	8VU3wPKq2lBY5bvnHQMS	0.000958	0.001142	0.001792	0.000183	0.000598	0.003909	0
1	ErYMOiwT8h3yqvfkm9Pu	0.000030	0.000011	0.000045	0.000457	0.000000	0.000000	0
2	dKNk2oDiqevnRTsgCmG0	0.001107	0.001101	0.001021	0.000000	0.000435	0.003909	0
3	dFUABXPZzn3mL06OEjek	0.001077	0.000000	0.000000	0.000000	0.000544	0.000000	0
4	EuH31db0lYoVintDaL9v	0.002753	0.007132	0.001474	0.005570	0.001522	0.007166	0
...	...	...	...	...	...	...	...	
10863	5C0Ke7rXRtb9smU3Qkho	0.001706	0.001239	0.002812	0.000913	0.000761	0.006352	0
10864	5mThHDbEpSQAYq3F1xvz	0.005057	0.002007	0.002517	0.001644	0.003316	0.006026	0
10865	H7CpUha5E4m9YzjxD0Z1	0.001287	0.001068	0.001882	0.000183	0.000435	0.004723	0
10866	8DxQhtuOKPRZAjGy6Y3E	0.000000	0.000005	0.000000	0.000000	0.000000	0.000000	0
10867	BMmCz4hZdgYjcbQD5oai	0.000030	0.000014	0.000045	0.008126	0.000000	0.000000	0

10868 rows × 1404 columns

In [ ]:

```

```

In [ ]:

```
final_y=merge_asm_byte["Class"]
```

In [ ]:

```
merge_asm_byte.fillna(merge_asm_byte.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
```

In [ ]:

```
merge_asm_byte.drop(columns=['Class','ID'],inplace=True)
```

In [ ]:

```
merge_asm_byte.shape
```

Out[108]:

```
(10868, 1402)
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(merge_asm_byte, final_y,stratify=final_
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

```
X_train.shape
```

Out[110]:

```
(6955, 1402)
```

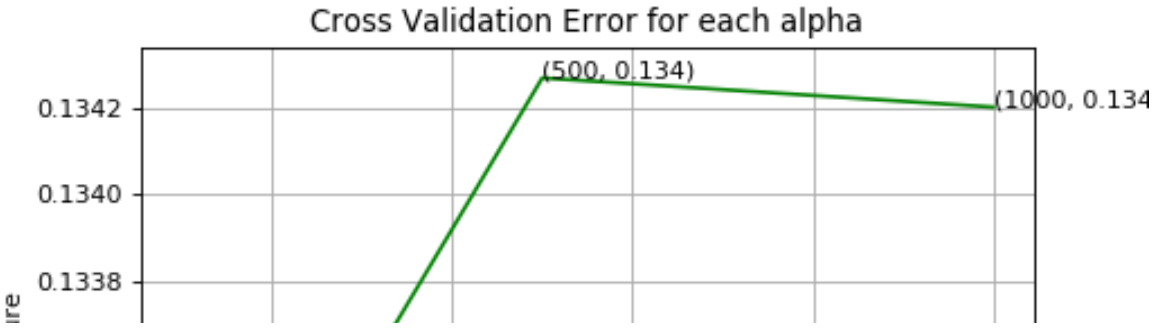
In [ ]:

```
from tqdm import tqdm_notebook as tqdm
```

In [ ]:

```
from tqdm import tqdm_notebook as tqdm
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[100,500,1000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=3)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

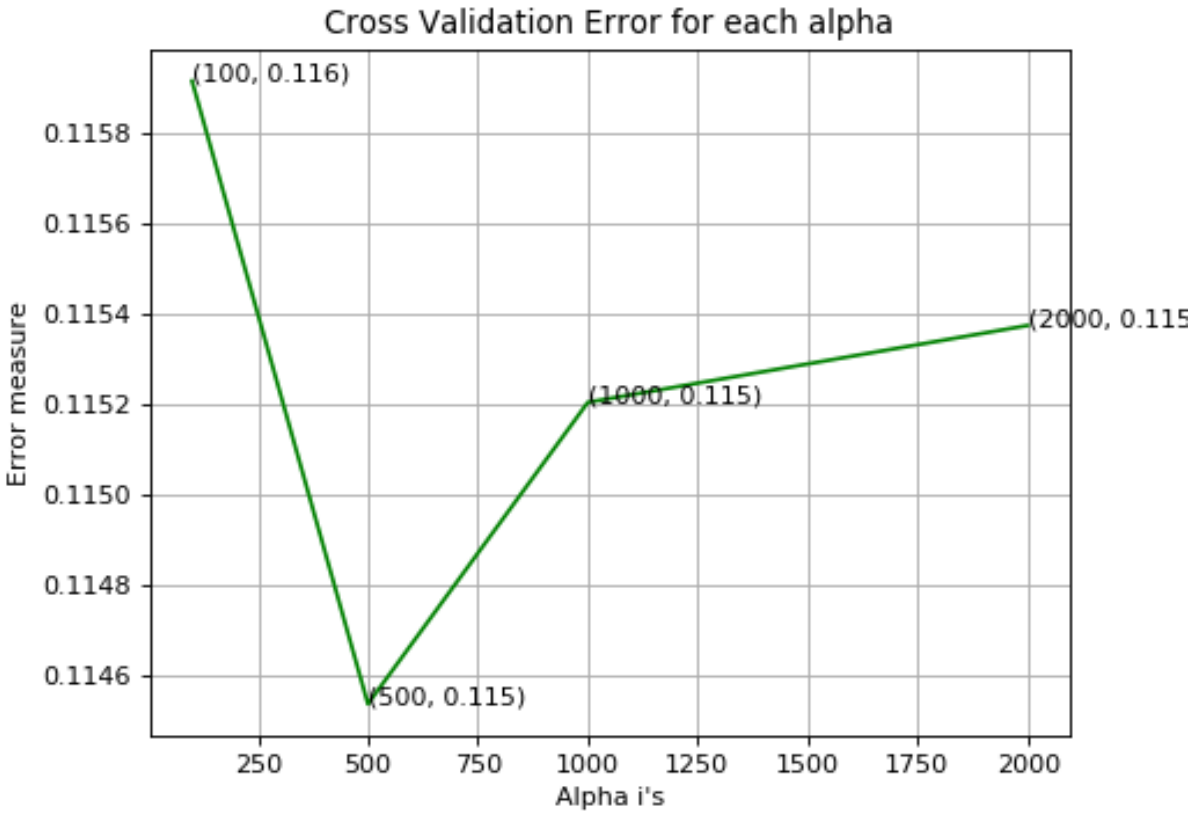
```
HBox(children=(FloatProgress(value=0.0, max=3.0), HTML(value='')))


log_loss for c =  100 is 0.1328755802035313
log_loss for c =  500 is 0.13426874317226414
log_loss for c =  1000 is 0.1342014664649063

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



In [ ]:

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

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

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

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

alpha=[100,500,1000,2000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=2)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

```
HBox(children=(FloatProgress(value=0.0, max=4.0), HTML(value='')))


log_loss for c =  100 is 0.11591611087067548
log_loss for c =  500 is 0.11453614532757751
log_loss for c =  1000 is 0.11520417743161893
log_loss for c =  2000 is 0.11537446230803011

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



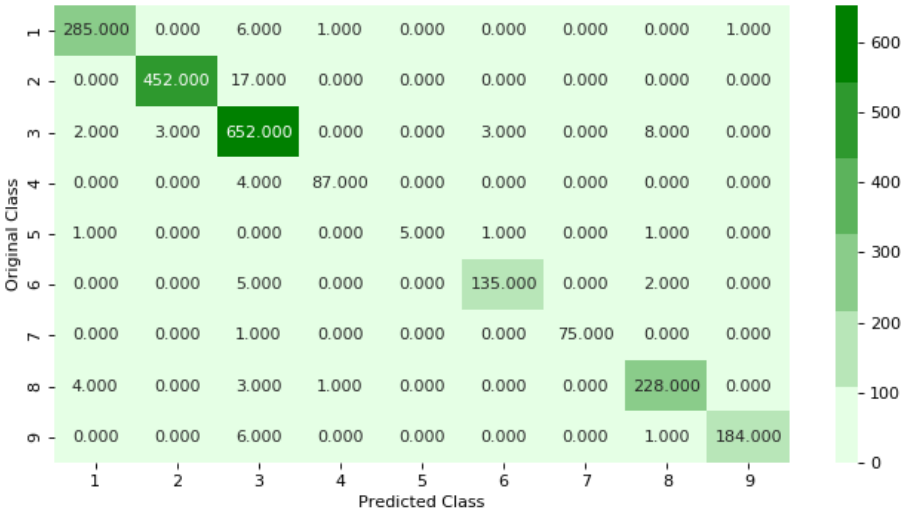

```
For values of best alpha =  500 The train log loss is: 0.033062678985337296
For values of best alpha =  500 The cross validation log loss is: 0.11453614
532757751
For values of best alpha =  500 The test log loss is: 0.10357779716836032
```

In [ ]:

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

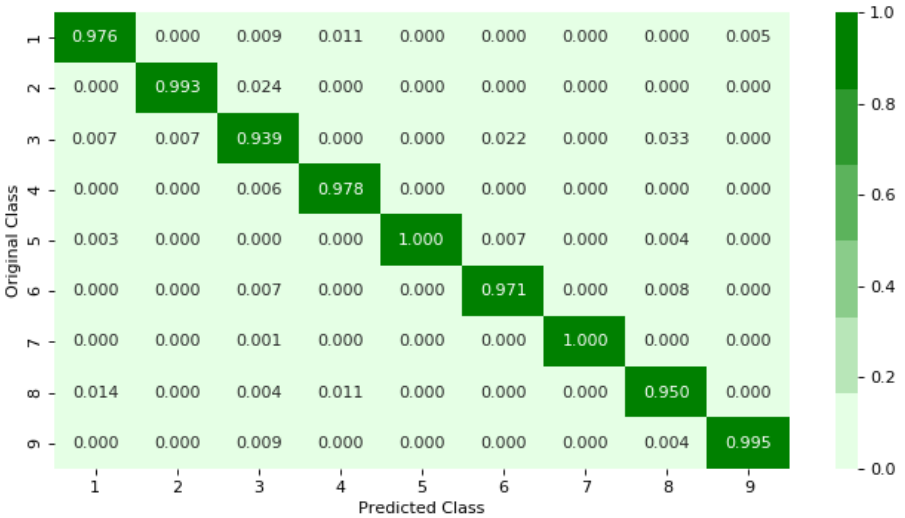
```
Number of misclassified points  3.265869365225391
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>



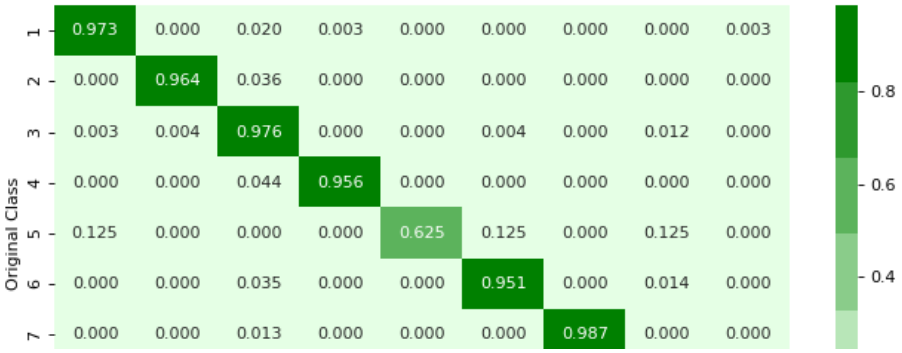

```
---------------------------------------------------- Precision matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

In [266]:

```
pip install prettytable
```

```
Requirement already satisfied: prettytable in /opt/conda/lib/python3.7/site-
packages (0.7.2)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [267]:

```
from prettytable import PrettyTable
```

In [ ]:

```
x=PrettyTable()
x.field_names = ["model", "featuriazation", "test_log_loss", "Miss-classfied_points"]

x.add_row(["Xg_boost","10k_byte_bigarm+bow_byte+size of bye",  0.0500170, 1.0119595])
x.add_row(["xg_boost", "asm_bow+800_pxl_byte", 0.0356, 0.64397])
x.add_row(["random_forest", "byte_top300+pxl_400+trigramasm_400df+asmbig250", 0.1035,3.2658
x.add_row(["Xg_boost", "byte_top300+pxl_400+trigramasm_400df+asmbig250",0.11663, 3.0358])
x.add_row(["Xgboost","byte_bigram + 800_bytepxl features" ,  0.0309563, 0.64397])
x.add_row(["Xg_boost","asm_bigram+800_bytepxl_features", 0.02063813,0.55197])
print(x)
```

```
+---------------+-----------------------------------------------------+----------
-----+-----------------------+
|     model     |                 featuriazation                 | test_log_
loss | Miss-classfied points |
+---------------+-----------------------------------------------------+----------
-----+-----------------------+
|    Xg_boost   |      10k_byte_bigarm+bow_byte+size of bye      |    0.0500
17   |       1.0119595       |
|    xg_boost   |              asm_bow+800_pxl_byte              |     0.035
6    |        0.64397        |
| random_forest | byte_top300+pxl_400+trigramasm_400df+asmbig250 |     0.103
5    |        3.26586        |
|    Xg_boost   | byte_top300+pxl_400+trigramasm_400df+asmbig250 |    0.1166
3    |         3.0358        |
|    Xgboost    |       byte_bigram + 800_bytepxl features       |   0.03095
63   |        0.64397        |
|    Xg_boost   |        asm_bigram+800_bytepxl_features         |   0.02063
813  |        0.55197        |
+---------------+-----------------------------------------------------+----------
-----+-----------------------+
```

summarise part: this assignemnt data was huge As I am unable to download and preprocess I have tried in notebook so, then by seeing the coments I swithced to google colab.

-> first I strated by taking bigram for bytes_file which constitute 66050 features then I also tried with 255 bag of words. with Xg_boost model and got the logloss of 0.05

-> As we knew the decison tree will work well for less features so we take top features only this time tried with asm_bow+800_pxl_byte we got this pxl_data through byte_file we converted byte files into pxl_data achieved log loss off 0.03456

-> To increase the variance of features we tried to use various features like byte_top300+pxl_400+trigramasm_400df+asmbig250 using random forest_model we achieved log_loss of 0.1035

-> we then tried with xg_boost model with the asm_byte and only 800_pxl_byte file data with which we achieved 0.0206,which is the less log loss I could reduce with having 1528 features only.

In [391]:

```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
```

In [ ]:

```
from tqdm import tqdm_notebook as tqdm
from nltk.util import ngrams
asm_trigram = ['ID']
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        for k in range(0, len(opcodes)):
            asm_trigram.append(v + ' ' + opcodes[j]+ ' ' + opcodes[k])

asm_trigram_file=open('asm_trigram_file.csv','w+')
asm_trigram_file.write(','.join(asm_trigram))
asm_trigram_file.write("\n")

asmFiles=os.listdir("asmFiles")
for file in tqdm(asmFiles):
    temp = dict(zip(asm_trigram, [0]*len(asm_trigram))).copy()
    temp['ID']=str(file)
    all_line=[]
    if(file.endswith(".asm")):
        op_code_str=""
        tri_g=[]
        with codecs.open('asmFiles/'+file,encoding='cp1252',errors ='replace') as asm_file:
            for lines in (asm_file):
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        op_code_str+=li+ " "
            tri_g = [' '.join(x) for x in list(ngrams(op_code_str.split(), 3))]
            for i in tri_g:
                temp[i.lower()]+=1
            asm_features=[]
            asm_features = [str(temp[x]) for x in asm_trigram]
            asm_trigram_file.write(','.join(asm_features))
            asm_trigram_file.write("\n")
            del temp
asm_trigram_file.close()
```

```
HBox(children=(FloatProgress(value=0.0, max=10868.0), HTML(value='')))
```

In [393]:

```
from tqdm import tqdm_notebook as tqdm
from nltk.util import ngrams
asm_tetagram = ['ID']
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)):
                asm_tetagram.append(v + ' ' + opcodes[j]+ ' ' + opcodes[k]+' ' + opcodes[l]

asm_tetagram_file=open('asm_tetragram_file.csv','w+')
asm_tetagram_file.write(','.join(asm_tetagram))
asm_tetagram_file.write("\n")

asmFiles=os.listdir("asmFiles")
for file in tqdm(asmFiles):
    temp = dict(zip(asm_tetagram, [0]*len(asm_tetagram))).copy()
    temp['ID']=str(file)
    all_line=[]
    if(file.endswith(".asm")):
        op_code_str=""
        teta_g=[]
        with codecs.open('asmFiles/'+file,encoding='cp1252',errors ='replace') as asm_file:
            for lines in (asm_file):
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        op_code_str+=li+ " "
            teta_g = [' '.join(x) for x in list(ngrams(op_code_str.split(),4))]
            for i in teta_g:
                temp[i.lower()]+=1
            asm_features=[]
            asm_features = [str(temp[x]) for x in asm_tetagram]
            asm_tetagram_file.write(','.join(asm_features))
            asm_tetagram_file.write("\n")
            del temp
asm_tetagram_file.close()
```

```
HBox(children=(FloatProgress(value=0.0, max=10868.0), HTML(value='')))

IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [ ]:

```
asm_tetra=pd.read_csv("asm_tetragram_file.csv")
```

In [ ]:

```
asm_teta.head()
```

In [5]:

```
asm_tri.head()
```

Out[5]:

	ID	jmp jmp jmp	jmp jmp mov	jmp jmp retf	jmp jmp push	jmp jmp pop	jmp jmp xor	jmp jmp retn	jmp jmp nop	jmp jmp sub	...	movzx movzx cmp
0	8VU3wPKq2lBY5bvnHQMS.asm	0	0	0	0	0	0	0	0	0	...	0
1	ErYMOiwT8h3yqvfkm9Pu.asm	16	4	0	5	3	0	0	0	0	...	0
2	dKNk2oDiqevnRTsgCmG0.asm	36	6	0	2	0	0	0	0	0	...	2
3	dFUABXPZzn3mL06OEjek.asm	0	0	0	0	0	0	0	0	0	...	6
4	EuH31db0IYoVintDaL9v.asm	2	4	0	4	0	1	0	0	1	...	5

5 rows × 17577 columns

In [6]:

```
pixel_intensity = np.zeros((10868, 800))
```

In [ ]:

```
"asmFiles/" + asmfile, 'rb'
```

In [115]:

```
print("hello")
```

```
hello
```

In [134]:

```
import array
def read_image(filename):
    f=open("asm_images/" + filename,'rb')
    ln=os.path.getsize("asm_images/" + filename)
    width=256
    rem=ln%width
    a=array.array("B")
    a.fromfile(f,ln-rem)
    f.close()
    g=np.reshape(a,(int(len(a)/width),width))
    g=np.uint8(g)
    b=g.copy()
    b.resize((500,))
    return list(b)
```

In [ ]:

```
```

In [136]:

```
from tqdm import tqdm_notebook as tqdm

import cv2
pixel_intensity = np.zeros((10868, 500))
img_file=[]
for i, imag in tqdm(enumerate(os.listdir("asm_images"))):
    img_file.append(imag.split('.')[0])
    #img = cv2.imread("asm_images/" + imag)
    pixel_intensity[i, :] += read_image(imag)
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(val
ue='')))
```

In [138]:

```
pxl_df.head()
```

Out[138]:

	ID	pxl0	pxl1	pxl2	pxl3	pxl4	pxl5	px
0	jxuKaQ3YtEZANs06iWUf	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
1	GBQtWjTxb4v21NCDAaoI	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
2	ilogAd4QsU38IFBcuwDp	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
3	clkRCKEyz7Irxd2Go9A0	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
4	1NpWfIMxU5ebEJHBAv2G	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095

5 rows × 501 columns

In [137]:

```
from tqdm import tqdm_notebook as tqdm

from sklearn.preprocessing import normalize
pxl_column = []
for i in range(500):
    pxl_column.append('pxl' + str(i))
pxl_df = pd.DataFrame(normalize(pixel_intensity , axis = 0), columns = pxl_column)
Img_id=pd.DataFrame(img_file)
pxl_df.insert(0,column='ID',value=Img_id)
import joblib
filename = "pixel_df500.pkl"
joblib.dump(pxl_df,filename)
```

Out[137]:

```
['pixel_df500.pkl']
```

In [88]:

```
from sklearn.preprocessing import normalize
pxl_column = []
for i in range(800):
    pxl_column.append('pxl' + str(i))
pxl_df = pd.DataFrame(normalize(pixel_intensity , axis = 0), columns = pxl_column)
```

In [90]:

```
Img_id=pd.DataFrame(img_file)
pxl_df.insert(0,column='ID',value=Img_id)
```

In [91]:

```
pxl_df.head()
```

Out[91]:

	ID	pxl0	pxl1	pxl2	pxl3	pxl4	pxl5	px
0	jxuKaQ3YtEZANs06iWUf	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
1	GBQtWjTxb4v21NCDAaoI	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
2	ilogAd4QsU38IFBcuwDp	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
3	clkRCKEyz7Irxd2Go9A0	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
4	1NpWfIMxU5ebEJHBAv2G	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095

5 rows × 801 columns

In [167]:

```
result_df=pd.read_csv("result_asm.csv")
```

In [168]:

```
result_df
```

Out[168]:

	Unnamed: 0	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:
0	0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0
1	1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0
2	2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0
3	3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0
4	4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0
...	...	...	...	...	...	...	...	...
10863	10863	kqvJp5E0wbWgu9mnzSQB	0	5062	0	303	3772	0
10864	10864	KQwj9O6dlPxNyf8zW0gp	0	3520	0	458	463638	0
10865	10865	KRNHAm094TC7OJfEPp8h	0	82940	0	397	36052	0
10866	10866	ksKnqcBVTC0a3zSGoveR	0	13682	0	454	842620	0
10867	10867	kSNnYl3ZLvB2WI7V4iEt	0	6003	0	466	1643010	0

10868 rows × 54 columns

In [92]:

```
import joblib
filename = "pixel_df2.pkl"
joblib.dump(pxl_df,filename)
```

Out[92]:

```
['pixel_df2.pkl']
```

In [118]:

```
asmbigram_df=pd.read_csv('asmbigram_process_vect_df.csv')
#top_asm_uptobig = pd.merge(left=asmbigram_process_vect_df, right=asm_tot.drop(columns=['Un
```

In [127]:

```
asmbigram_df=pd.read_csv('asmbigram_process_vect_df.csv')
#top_asm_uptobig = pd.merge(left=asmbigram_process_vect_df, right=asm_tot.drop(columns=['Un
result_df=pd.read_csv("result_asm.csv")
```

In [97]:

```
asmbigram_df['ID']=asmbigram_df['ID'].map(lambda x : x.rstrip(".asm"))
```

In [115]:

```
asmbigram_df.iloc[:, 1:]=normalize(asmbigram_df.iloc[:, 1:], axis = 0)
```

In [133]:

```
asmbigram_df=pd.read_csv('asmbigram_process_vect_df.csv')
#top_asm_uptobig = pd.merge(left=asmbigram_process_vect_df, right=asm_tot.drop(columns=['Un
result_df=pd.read_csv("result_asm.csv")
fin_feat= pd.merge(left=asmbigram_df.drop(columns=['Unnamed: 0']), right=result_df[['ID','C
```

In [135]:

```
fin_feat.head()
```

Out[135]:

	ID	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn
0	8VU3wPKq2lBY5bvnHQMS	0.000000	0.000852	0.000000	0.000350	0.000000	0.000419	0.0
1	ErYMOiwT8h3yqvfkm9Pu	0.002241	0.000162	0.001467	0.000700	0.001123	0.000000	0.0
2	dKNk2oDiqevnRTsgCmG0	0.003326	0.002049	0.000000	0.000700	0.000000	0.000000	0.0
3	dFUABXPZzn3mL06OEjek	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
4	EuH31db0IYoVintDaL9v	0.000868	0.008073	0.000000	0.003923	0.000749	0.001397	0.0

5 rows × 678 columns

In [136]:

```
pxl_df.head()
```

Out[136]:

	ID	pxl0	pxl1	pxl2	pxl3	pxl4	pxl5	px
0	jxuKaQ3YtEZANs06iWUf	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
1	GBQtWjTxb4v21NCDAaoI	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
2	ilogAd4QsU38IFBcuwDp	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
3	clkRCKEyz7Irxd2Go9A0	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
4	1NpWfIMxU5ebEJHBAv2G	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095

5 rows × 801 columns

In [137]:

```
fin_feat= pd.merge(left=fin_feat, right=pxl_df,how='left', left_on='ID', right_on='ID')
```

In [139]:

```
fin_feat.to_csv("finalz_feat.csv")
```

In [ ]:

```
final_y=fin_feat['Class']
```

In [150]:

```
fin_feat.drop(columns='ID',axis=0,inplace=True)
```

In [151]:

```
fin_feat.fillna(fin_feat.mean(), inplace=True)
final_y=final_y.replace(np.nan, 3.0)
X_train, X_test, y_train, y_test = train_test_split(fin_feat, final_y,stratify=final_y,test
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [152]:

```
X_train, X_test, y_train, y_test = train_test_split(fin_feat, final_y,stratify=final_y,test
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=500,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

In [156]:

```
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

```
The train log loss is: 0.01771805180415808
The cross validation log loss is: 0.05545844279355746
The test log loss is: 0.06888394702744774
```

In [230]:

```
def top_features(data,class_y, top):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, class_y)
    top_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    return top_feature_indx[:top]
```

In [23]:

```
asmbigram_df=pd.read_csv('asmbigram_process_vect_df.csv')
result_df=pd.read_csv("result_asm.csv")
asmbigram_df= pd.merge(left=asmbigram_df.drop(columns=['Unnamed: 0']), right=result_df[['ID
```

In [24]:

```
asmbigram_df.drop(columns=['ID','Class']).fillna(asmbigram_df.drop(columns=['ID','Class']),
final_y=asmbigram_df["Class"].replace(np.nan, 3.0)
```

In [25]:

```
idx=top_features(asmbigram_df.drop(columns=['ID','Class']),final_y,210)
```

In [28]:

```
idx=np.insert(idx,0,0)
```

In [29]:

```
fin_asmbig=asmbigram_df.iloc[:,idx]
```

In [30]:

```
fin_asmbig.head()
```

Out[30]:

	ID	mov jmp	retn retf	cmp shr	add push	cmp rol	call shr	pop push	sub jmp
0	8VU3wPKq2lBY5bvnHQMS	0.001074	0.0	0.0	0.001027	0.0	0.000000	0.000372	0.000000
1	ErYMOiwT8h3yqvfkm9Pu	0.000137	0.0	0.0	0.000616	0.0	0.000000	0.001487	0.000000
2	dKNk2oDiqevnRTsgCmG0	0.001645	0.0	0.0	0.001848	0.0	0.000000	0.000000	0.000305
3	dFUABXPZzn3mL06OEjek	0.000000	0.0	0.0	0.000616	0.0	0.000000	0.000000	0.000000
4	EuH31db0lYoVintDaL9v	0.010694	0.0	0.0	0.000068	0.0	0.004411	0.000186	0.003960

5 rows × 211 columns

In [31]:

```
fin_asmbig.to_csv("fin_asmbig.csv")
```

In [5]:

```
asm_tri=pd.read_csv("asm_trigram_file.csv")
asm_tri['ID']=asm_tri['ID'].map(lambda x : x.rstrip('.asm') )
```

In [ ]:

```
asm_tri.iloc[:, 1:17577]=normalize(asm_tri.iloc[:, 1:17577], axis = 0)
```

In [ ]:

```
asm_tri.head()
```

In [10]:

```
#asm_tri=pd.read_csv("asm_trigram_file.csv")
result_df=pd.read_csv("result_asm.csv")
asm_tri= pd.merge(left=asm_tri, right=result_df[['ID','Class']],how='left', left_on='ID', r
```

In [15]:

```
asm_tri.drop(columns=['ID','Class']).fillna(asm_tri.drop(columns=['ID','Class']), inplace=T
final_y=asm_tri["Class"].replace(np.nan, 3.0)
```

In [19]:

```
indx=np.insert(indx,0,0)
```

In [21]:

```
fin_asmtri=asm_tri.iloc[:,indx]
```

In [22]:

```
fin_asmtri.to_csv("fin_asmtri.csv")
```

In [43]:

```
tot_feat.head()
```

Out[43]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 (
0	D2FbBoJAweiY8X0pxTOn	0.005062	0.017984	0.002374	0.003711	0.004159	0.001102	0.0007
1	ATVo94avrEyjnexXU8W7	0.000974	0.000443	0.000343	0.001211	0.000205	0.000962	0.0011
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
3	0GUIi7xAlODwZ4YBenNM	0.002237	0.001810	0.001739	0.001693	0.001989	0.000210	0.0000
4	CdeQsaIqAiMVRjT2Pun6	0.002025	0.006216	0.002031	0.002702	0.007431	0.010027	0.0088

5 rows × 66052 columns

byte_bigram_top_300

In [61]:

```
```

Out[61]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	
0	D2FbBoJAweiY8X0pxTOn	0.005062	0.017984	0.002374	0.003711	0.004159	0.001102	0.
1	ATVo94avrEyjnexXU8W7	0.000974	0.000443	0.000343	0.001211	0.000205	0.000962	0
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
3	0GUIi7xAlODwZ4YBenNM	0.002237	0.001810	0.001739	0.001693	0.001989	0.000210	0.
4	CdeQsaIqAiMVRjT2Pun6	0.002025	0.006216	0.002031	0.002702	0.007431	0.010027	0.
...	...	...	...	...	...	...	...	
4995	9LM4YBhu0K8oxaUGDTln	0.003108	0.015098	0.012275	0.012516	0.006764	0.009170	0.
4996	14DUiwIb5xrfkF30cThJ	0.003470	0.000966	0.000279	0.001848	0.000180	0.000210	0.
4997	jzCaxOEMKRNd8V7qBmGi	0.003076	0.008821	0.003910	0.004488	0.005930	0.004987	0.
4998	DGmbwEzu381CyXgAMB2q	0.001069	0.000583	0.000394	0.001398	0.000295	0.000980	0.
4999	dOeUI4W0VjhNE7uF8lz1	0.003645	0.009083	0.002717	0.002749	0.003774	0.002187	0.

5000 rows × 66052 columns

In [77]:

```
#tot_feat.drop(columns=['ID','Class'])[:2500].fillna(tot_feat.drop(columns=['ID','Class'])[
final_y=tot_feat["Class"][:2500].replace(np.nan, 3.0)
```

In [ ]:

```
import joblib
filename = "bytebigram_top300_impfeat.pkl"
bytebigram_top300=joblib.load(filename)
bytebigram_top300=bytebigram_top300.insert(0,'ID')
tot_feat[bytebigram_top300]
```

In [120]:

```
fin_bytebigram=pd.read_csv("fin_bytebigram.csv")
```

In [14]:

```
import joblib
file_name="bigram_topindez.pkl"
bigram_topindez=joblib.load(file_name)
```

In [15]:

```
bigram_topindez=np.insert(bigram_topindez,0,0)
```

In [16]:

```
fin_bytebigram=tot_feat.iloc[:,bigram_topindez]
```

In [18]:

```
fin_bytebigram.head()
```

Out[18]:

	ID	ed 51	d7 ed	42 e0	92 d8	80 d2	52 35	66 8
0	D2FbBoJAweiY8X0pxTOn	0.000013	0.000000	0.002054	0.000000	0.001442	0.000010	0.0008
1	ATVo94avrEyjnexXU8W7	0.000189	0.011799	0.009241	0.015083	0.007931	0.000128	0.0006
2	cf4nzsoCmudt1kwleOTI	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
3	0GUIi7xAlODwZ4YBenNM	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0001
4	CdeQsaIqAiMVRjT2Pun6	0.000067	0.004291	0.005134	0.002011	0.004326	0.000039	0.0023

5 rows × 301 columns

In [83]:

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

In [74]:

```
result_asm=pd.read_csv("result_asm.csv")
```

In [76]:

```
asm_size_byte.rename(columns={"size": "asm_size"},inplace=True)
```

In [77]:

```
asm_bow= pd.merge(left=result_asm, right=asm_size_byte[['ID','asm_size']],how='inner', left
```

In [86]:

```
asm_bow.head()
```

Out[86]:

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

5 rows × 54 columns

In [84]:

```
#asm_bow=pd.read_csv('asm_bow_with_size.csv')
asm_bow=normalize(asm_bow)
```

In [85]:

```
byte_bow['ID']=byte_bow['ID'].map(lambda x:x.rstrip(".txt"))
```

In [24]:

```
fin_asmbigram=pd.read_csv("fin_asmbig.csv")
```

In [25]:

```
fin_asmtrigram=pd.read_csv("fin_asmtri.csv")
```

In [29]:

```
import joblib
pxl_asm=joblib.load('pixel_df2.pkl')
```

In [28]:

```
asm_bow['asm_size'].isna().sum
```

Out[28]:

```
<bound method Series.sum of 0        False
1        False
2        False
3        False
4        False
         ...
10863    False
10864    False
10865    False
10866    False
10867    False
Name: asm_size, Length: 10868, dtype: bool>
```

In [30]:

```
fin_asmbigram.head()
```

Out[30]:

	Unnamed: 0	ID	mov jmp	retn retf	cmp shr	add push	cmp rol	call shr	po pus
0	0	8VU3wPKq2lBY5bvnHQMS	0.001074	0.0	0.0	0.001027	0.0	0.000000	0.00037
1	1	ErYMOiwT8h3yqvfkm9Pu	0.000137	0.0	0.0	0.000616	0.0	0.000000	0.00148
2	2	dKNk2oDiqevnRTsgCmG0	0.001645	0.0	0.0	0.001848	0.0	0.000000	0.00000
3	3	dFUABXPZzn3mL06OEjek	0.000000	0.0	0.0	0.000616	0.0	0.000000	0.00000
4	4	EuH31db0IYoVintDaL9v	0.010694	0.0	0.0	0.000068	0.0	0.004411	0.00018

5 rows × 212 columns

In [51]:

```
asm_bow.shape
```

Out[51]:

```
314
```

In [39]:

```
byte_bow.shape
```

Out[39]:

```
(10868, 261)
```

In [40]:

```
fin_bytebigram.shape
```

Out[40]:

```
(10868, 301)
```

In [34]:

```
fin_asmtrigram.head()
```

Out[34]:

	Unnamed: 0	ID	shl lea push	mov mov jmp	retn push jmp	call add push	mov mov shr	mov jmp jmp	
0	0	8VU3wPKq2lBY5bvnHQMS	0.0	0.001155	0.0	0.000657	0.002306	0.000000	0
1	1	ErYMOiwT8h3yqvfkm9Pu	0.0	0.000072	0.0	0.000164	0.000000	0.003101	0
2	2	dKNk2oDiqevnRTsgCmG0	0.0	0.001949	0.0	0.000985	0.000419	0.003101	0
3	3	dFUABXPZzn3mL06OEjek	0.0	0.000000	0.0	0.000000	0.000000	0.000000	0
4	4	EuH31db0IYoVintDaL9v	0.0	0.016675	0.0	0.000000	0.008596	0.003876	0

5 rows × 302 columns

In [35]:

```
pxl_asm.head()
```

Out[35]:

	ID	pxl0	pxl1	pxl2	pxl3	pxl4	pxl5	px
0	jxuKaQ3YtEZANs06iWUf	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
1	GBQtWjTxb4v21NCDAaoI	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
2	ilogAd4QsU38IFBcuwDp	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
3	clkRCKEyz7Irxd2Go9A0	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095
4	1NpWfIMxU5ebEJHBAv2G	0.009592	0.009592	0.009592	0.009592	0.009592	0.009592	0.0095

5 rows × 801 columns

In [ ]:

```
asm_bow #52
byte_bow #258

fin_asmbigram #211
fin_asmtrigram #300

fin_bytebigram #300
pxl_asm #800
```

In [103]:

```
byte_bow.head()
```

Out[103]:

	Unnamed: 0	ID	0	1	2	3	4	5
0	0.000000	01azqd4inc7m9jpocgv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835
1	0.000092	01isoismh5gxydytl4cb	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873
2	0.000184	01jsnpxsalgw6apedxru	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280
3	0.000276	01kcpwa9k2boxqes5rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354
4	0.000368	01suzwmjeixsk7a8dqbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232

5 rows × 261 columns

In [200]:

```
fin_bytebigram.head()
```

Out[200]:

	ID	ed 51	d7 ed	42 e0	92 d8	80 d2	52 35	66 8b
0	d2fbbojaweiy8x0pxton	0.000013	0.000000	0.002054	0.000000	0.001442	0.000010	0.000890
1	atvo94avreyjnexxu8w7	0.000189	0.011799	0.009241	0.015083	0.007931	0.000128	0.000653
2	cf4nzsocmudt1kwleoti	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0guii7xalodwz4ybennm	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000178
4	cdeqsaiqaimvrjt2pun6	0.000067	0.004291	0.005134	0.002011	0.004326	0.000039	0.002373

5 rows × 301 columns

In [157]:

```
final_feat_right['ID']=final_feat_right['ID'].str.lower()
final_feat_left['ID']=final_feat_left['ID'].str.lower()
```

In [140]:

```
final_feat_left= pd.merge(left=fin_bytebigram,right=byte_bow.drop(columns=['Unnamed: 0']),h
```

In [142]:

```
final_feat_right= pd.merge(left=fin_asmbigram, right=fin_asmtrigram.drop(columns=['Unnamed:
```

In [147]:

```
final_feat_updown=pd.merge(left=pxl_df,right=asm_bow,how='inner', left_on='ID', right_on='I
```

In [158]:

```
final_feat_a= pd.merge(left=final_feat_right, right=final_feat_left,how='inner', left_on='I
```

In [160]:

```
final_feat= pd.merge(left=final_feat_a, right=final_feat_updown,how='inner', left_on='ID',
```

In [162]:

```
final_feat.drop(columns=['ID.1_x','ID.1_y'],inplace=True)
final_feat.drop(columns=['Class_x','Unnamed: 0'],inplace=True)
```

In [163]:

```
final_feat.shape
```

Out[163]:

```
(9982, 1620)
```

In [2]:

```
final_feat=pd.read_csv('final_feat_asm_byte_1420.csv')
```

In [84]:

```
final_feat=pd.read_csv("final_feat_asm_byte.csv")
```

In [127]:

```
final_y=final_feat['Class_y']
```

In [166]:

```
final_feat.drop(columns=['Class_y','ID']).fillna(final_feat.drop(columns=['Class_y','ID']).
final_y=final_y.replace(np.nan, 3.0)
```

In [167]:

```
X_train, X_test, y_train, y_test = train_test_split(final_feat.drop(columns=['Class_y','ID'
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size
```

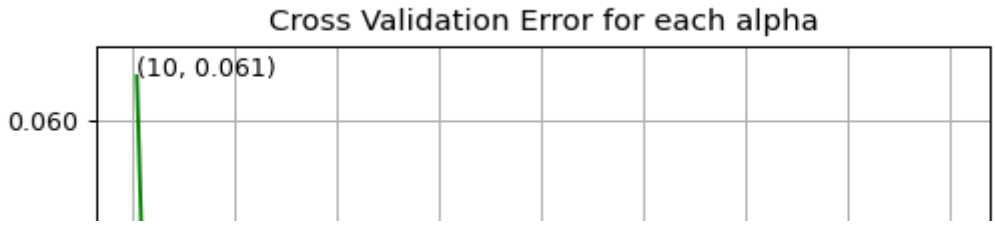
In [ ]:

```
final_feat['asm_size']
```

In [204]:

```
from tqdm import tqdm_notebook as tqdm
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=3)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
HBox(children=(FloatProgress(value=0.0, max=6.0), HTML(value='')))


log_loss for c =  10 is 0.06072273874909181
log_loss for c =  50 is 0.051800390060227144
log_loss for c =  100 is 0.051294007952642745
log_loss for c =  500 is 0.05102610464083191
log_loss for c =  1000 is 0.051025430497676535
log_loss for c =  2000 is 0.05102526311545177

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



In [103]:

```
X_test.shape
```

Out[103]:

```
(2174, 1917)
```

In [104]:

```
307+496+588+92+147+79+242+200
```

Out[104]:

```
2151
```

In [105]:

```
2151/2174
```

Out[105]:

```
0.9894204231830727
```

In [112]:

```
2174-2151
```

Out[112]:

```
23
```

In [108]:

```
23/2151
```

Out[108]:

```
0.010692701069270108
```

In [199]:

```
X_train.shape
```

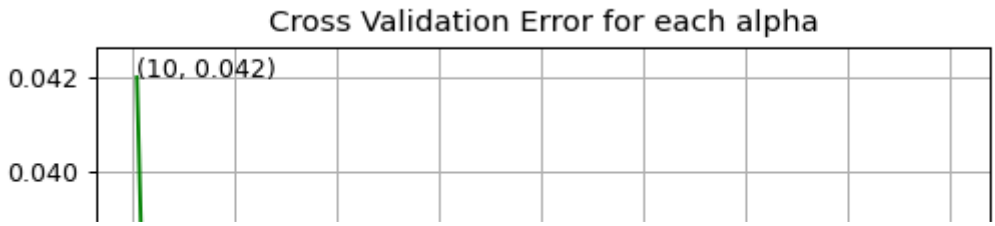
Out[199]:

```
(6955, 1228)
```

In [387]:

```
from tqdm import tqdm_notebook as tqdm
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[50,100,500,2000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=3)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
HBox(children=(FloatProgress(value=0.0, max=6.0), HTML(value='')))


log_loss for c =  10 is 0.042016942724568324
log_loss for c =  50 is 0.029016844307897728
log_loss for c =  100 is 0.029333940567562455
log_loss for c =  500 is 0.029320128895201362
log_loss for c =  1000 is 0.029320363258021002
log_loss for c =  2000 is 0.02932024635349758

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



In [258]:

```
X_train.shape
```

Out[258]:

```
(6955, 1528)
```

In [198]:

```
r_cfl=XGBClassifier(n_estimators=500,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

```
The train log loss is: 0.012882318862895835
The cross validation log loss is: 0.03208785565858091
The test log loss is: 0.029954291545974926
```

In [ ]:

```
from tqdm import tqdm_notebook as tqdm
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pytho
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

alpha=[50,100,500,2000]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,n_jobs=3)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],n_jobs=2)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

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

```
HBox(children=(FloatProgress(value=0.0, max=4.0), HTML(value='')))
```

In [ ]:

```
r_cfl=XGBClassifier(n_estimators=500,max_depth= 3,learning_rate= 0.1,
 colsample_bytree= 1,random_state=42,n_jobs=-1)
r_cfl.fit(X_train ,y_train )
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train )
predict_y = sig_clf.predict_proba(X_train)
print ( "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print( "The test log loss is:",log_loss(y_test, predict_y))
```

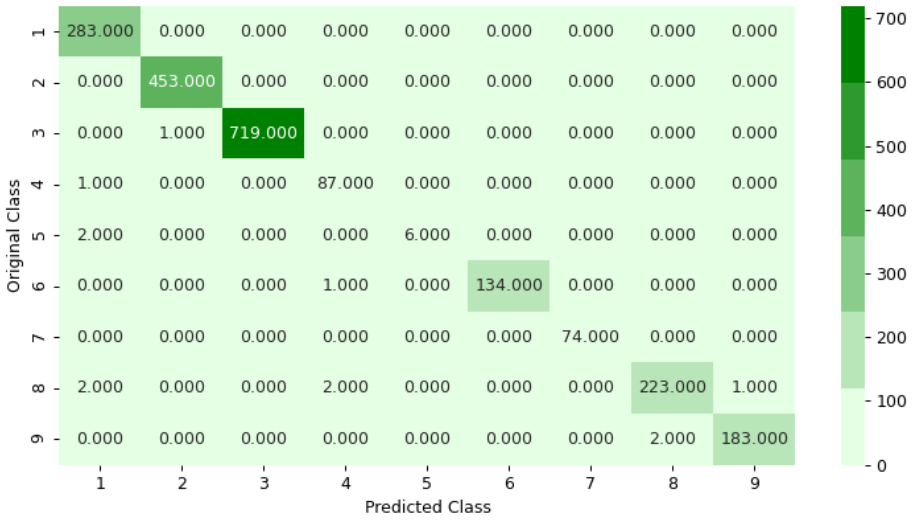
```
The train log loss is: 0.011677288128305258
The cross validation log loss is: 0.029473259079896015
The test log loss is: 0.02063813700116953
```

In [262]:

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

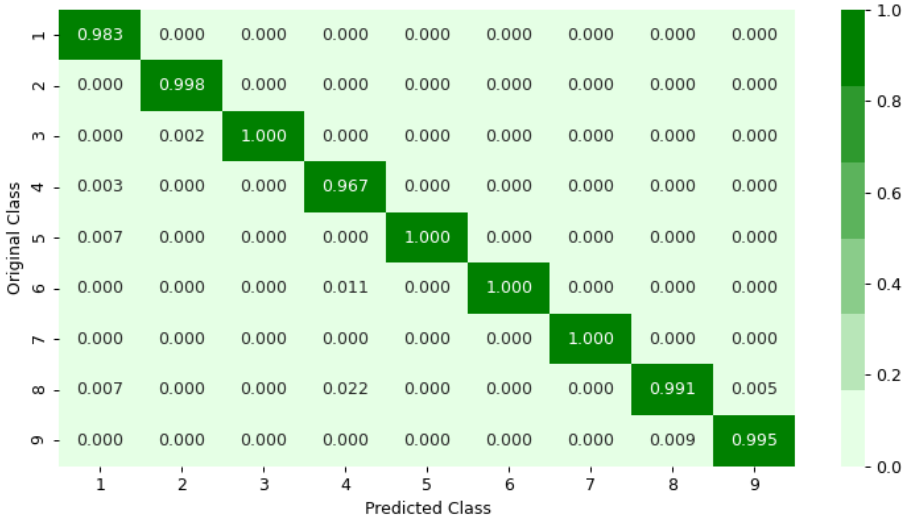
```
Number of misclassified points  0.5519779208831647
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>


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

In [268]:

```
x=PrettyTable()
x.field_names = ["model", "featuriazation", "test_log_loss", "Miss-classfied_points"]

x.add_row(["Xg_boost","10k_byte_bigarm+bow_byte+size of bye",  0.0500170, 1.0119595])
x.add_row(["xg_boost", "asm_bow+800_pxl_byte", 0.0356, 0.64397])
x.add_row(["random_forest", "byte_top300+pxl_400+trigramasm_400df+asmbig250", 0.1035,3.2658
x.add_row(["Xg_boost", "byte_top300+pxl_400+trigramasm_400df+asmbig250",0.11663, 3.0358])
x.add_row(["Xgboost","byte_bigram + 800_bytepxl features" ,  0.0309563, 0.64397])
x.add_row(["Xg_boost","asm_bigram+500_asmpxl_features+bytte_bigram and trigram", 0.02063813
print(x)
```

```
+---------------+------------------------------------------------------------+-
--------------+-----------------------+
|     model     |                      featuriazation                        |
test_log_loss | Miss-classfied_points |
+---------------+------------------------------------------------------------+-
--------------+-----------------------+
|    Xg_boost   |           10k_byte_bigarm+bow_byte+size of bye             |
0.050017   |       1.0119595       |
|    xg_boost   |                    asm_bow+800_pxl_byte                    |
0.0356     |        0.64397        |
| random_forest |      byte_top300+pxl_400+trigramasm_400df+asmbig250        |
0.1035     |        3.26586        |
|    Xg_boost   |      byte_top300+pxl_400+trigramasm_400df+asmbig250        |
0.11663    |         3.0358        |
|    Xgboost    |            byte_bigram + 800_bytepxl features              |
0.0309563   |        0.64397        |
|    Xg_boost   | asm_bigram+500_asmpxl_features+bytte_bigram and trigram    |
0.02063813  |        0.55197        |
+---------------+------------------------------------------------------------+-
--------------+-----------------------+
```

In [ ]: