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

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."

3. Exploratory Data Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
import tqdm as tqdm
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.feature_selection import chi2
from sklearn.feature_extraction.text import CountVectorizer
import scipy
```

In [2]:

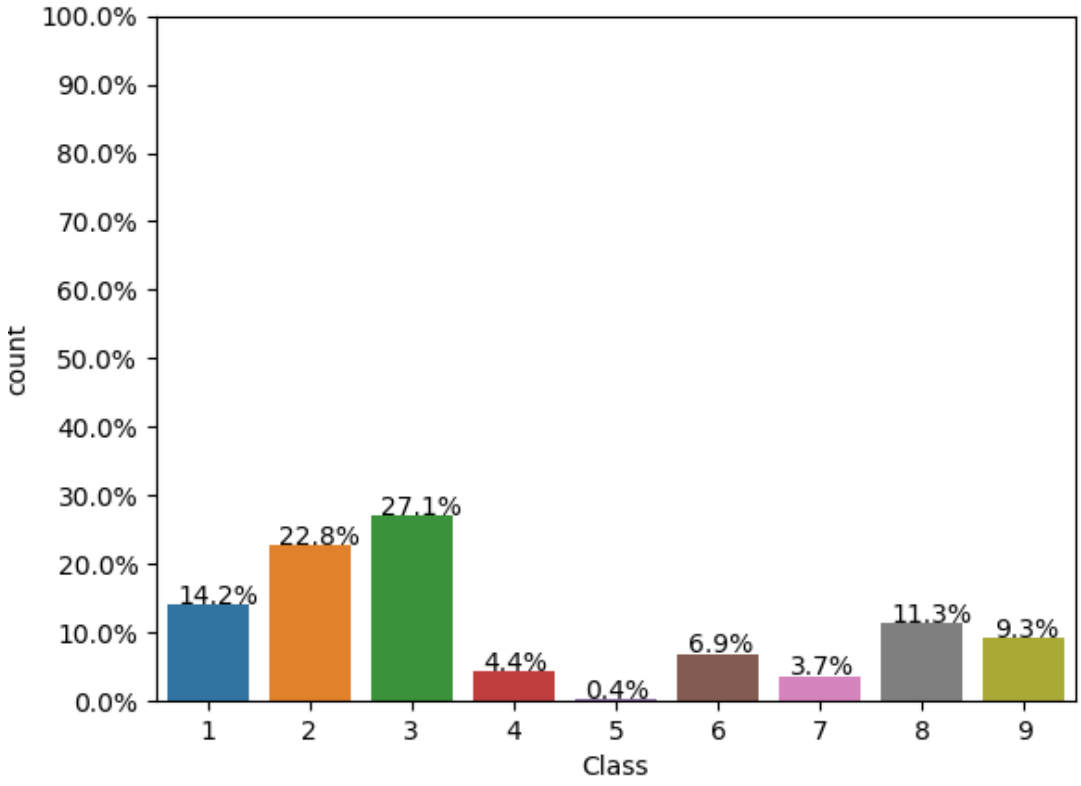
```
#separating byte files and asm files
source = 'train'
destination = 'byteFiles'

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

# 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):
    os.rename(source,'asmFiles')
    source='asmFiles'
    data_files = os.listdir(source)
    for file in data_files:
        print(file)
        if (file.endswith("bytes")):
            shutil.move(source+'\\'+file,destination)
```

```
ATDpmb5qodJ1VRrc9KNg.asm
ATDpmb5qodJ1VRrc9KNg.bytes
AtDzQGnjHw4lNLqJsukC.asm
AtDzQGnjHw4lNLqJsukC.bytes
AtDZWHR1rcVobv0yp8Yu.asm
AtDZWHR1rcVobv0yp8Yu.bytes
atexryL75EAFchJOBMbg.asm
atexryL75EAFchJOBMbg.bytes
aTfDJBX2PNZRIjUG1SKz.asm
aTfDJBX2PNZRIjUG1SKz.bytes
ATfrBeFJV9U2NDdCmEcZ.asm
ATfrBeFJV9U2NDdCmEcZ.bytes
atGqPui5xlRbN6FDWhsZ.asm
atGqPui5xlRbN6FDWhsZ.bytes
atHAQyVSuXhR93zYp0r7.asm
atHAQyVSuXhR93zYp0r7.bytes
ATI0r3nmuPxolGfaqVFZ.asm
ATI0r3nmuPxolGfaqVFZ.bytes
AtjeimUnQDzEqb6TN8cC.asm
AtjeimUnQDzEqb6TN8cC.bytes
```

3.1. Distribution of malware classes in whole data set

In [6]:

```
Y=pd.read_csv("data/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()
```



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [37]:

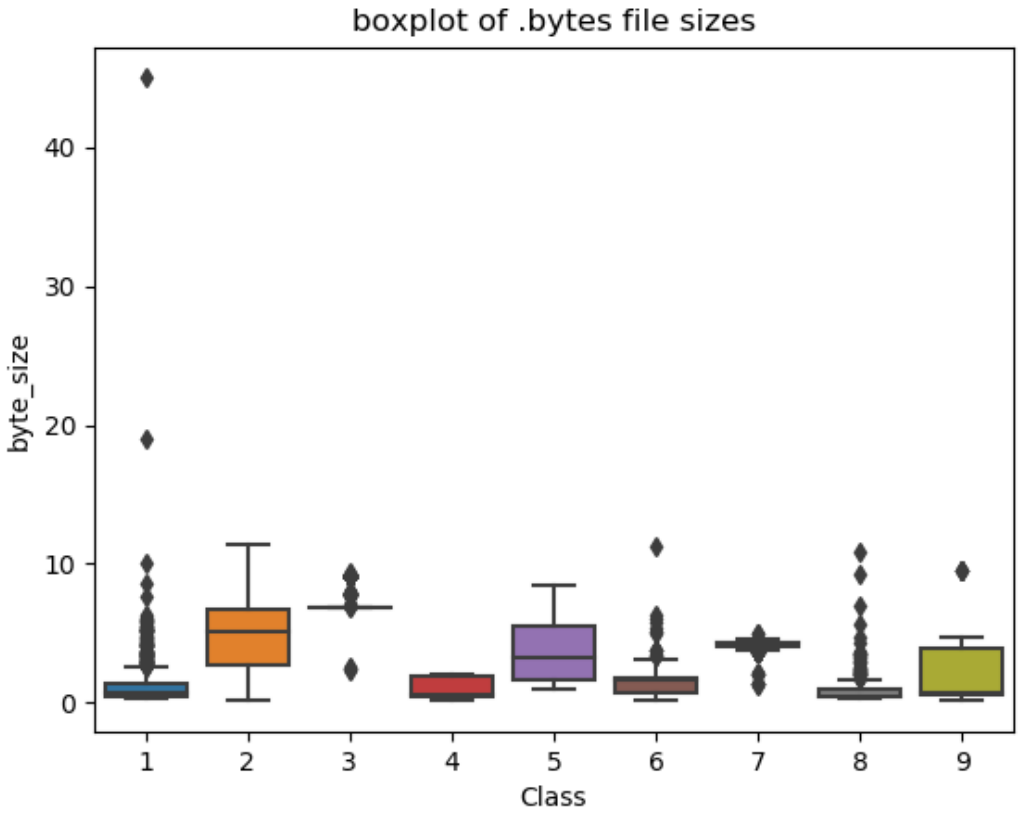
```
#file sizes of byte files

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,'byte_size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

```
                     ID  byte_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 [38]:

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



3.2.3 feature extraction from byte files

In [5]:

```
#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('data/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 [26]:

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

Out[26]:

	ID	0	1	2	3	4	5	6	7	8	...	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965.0	...	2804
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291.0	...	451

2 rows × 258 columns

In [27]:

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

Out[27]:

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

In [28]:

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

Out[28]:

	ID	0	1	2	3	4	5	6	7	8	...	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965.0	...	3101
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291.0	...	439

2 rows × 260 columns

In [2]:

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

In [30]:

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

Out[30]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	1.0000	0.062368	0.042596	0.064213	0.012338	0.197884	0.248006
1	01IsoiSMh5gxyDYTl4CB	0.0561	0.141548	0.117708	0.125729	0.033844	0.442154	0.590801

2 rows × 260 columns

In [41]:

```
data_y = result['Class']
data_y[0:2]
```

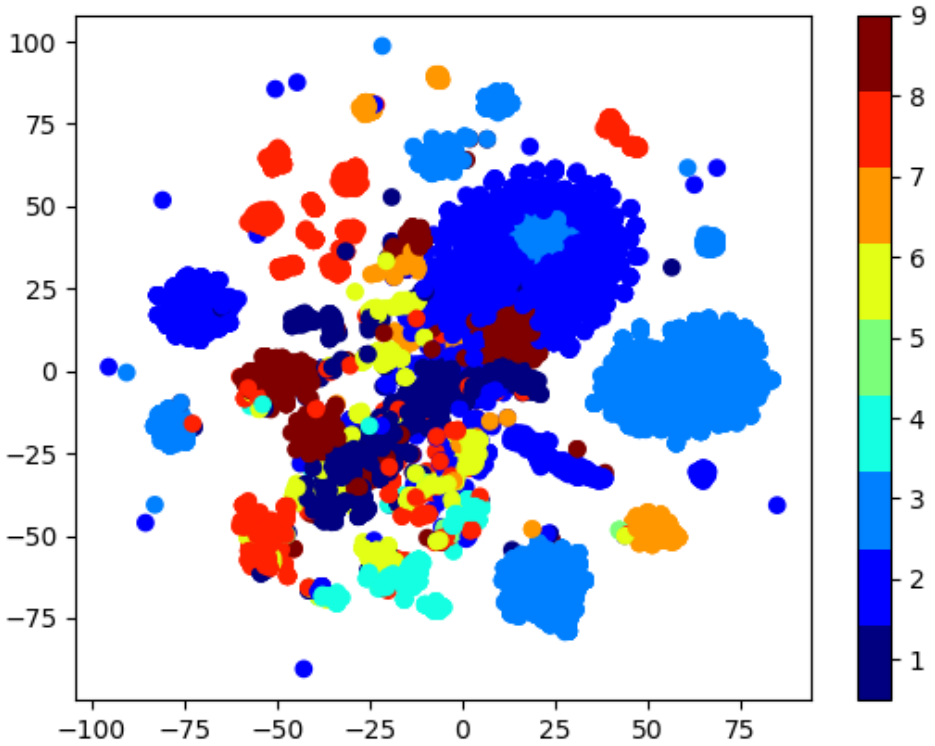
Out[41]:

```
0    9
1    2
Name: Class, dtype: int64
```

3.2.4 Multivariate Analysis

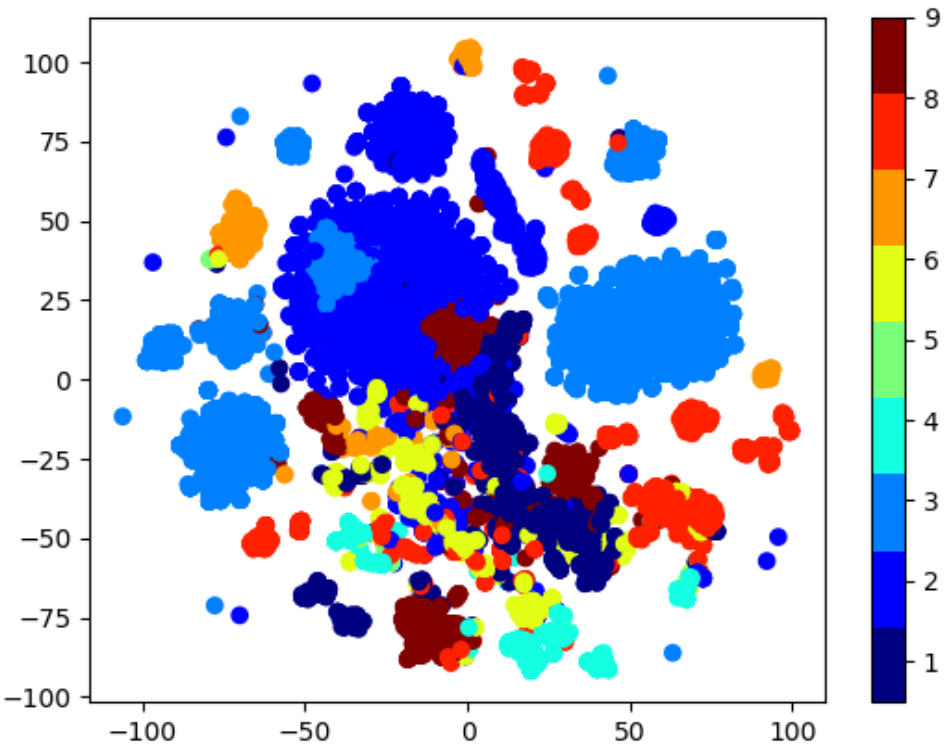
In [88]:

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



In [13]:

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



Train Test split

In [9]:

```
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'], axis=1), da
# 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
```

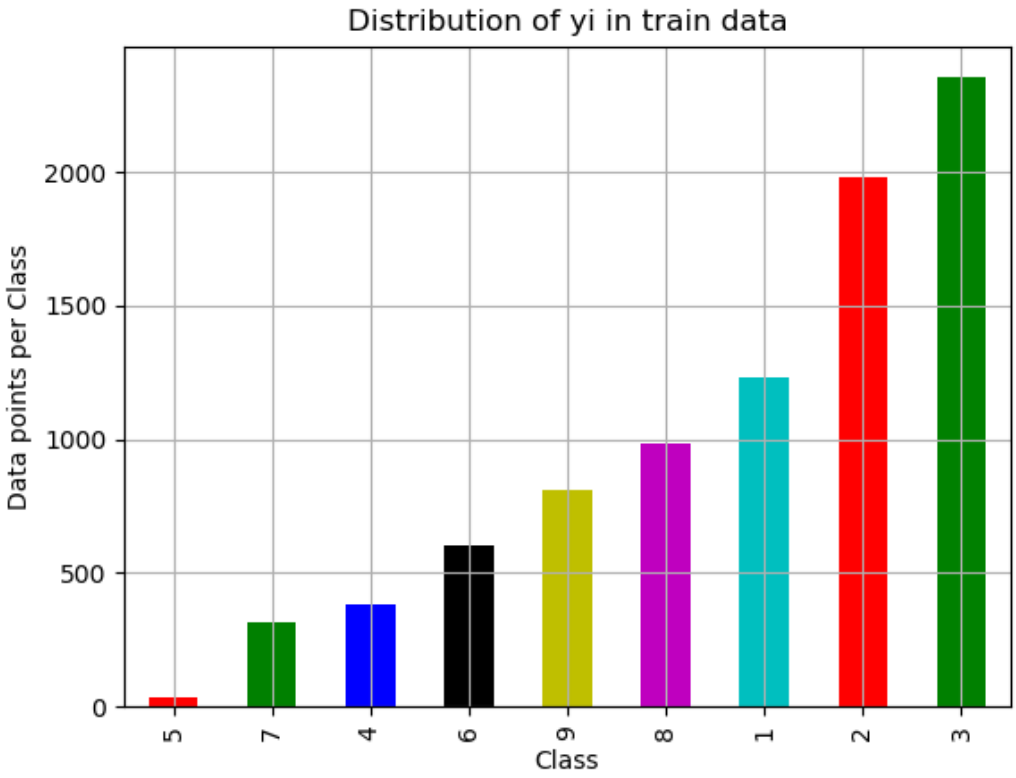
In [10]:

```
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 [89]:

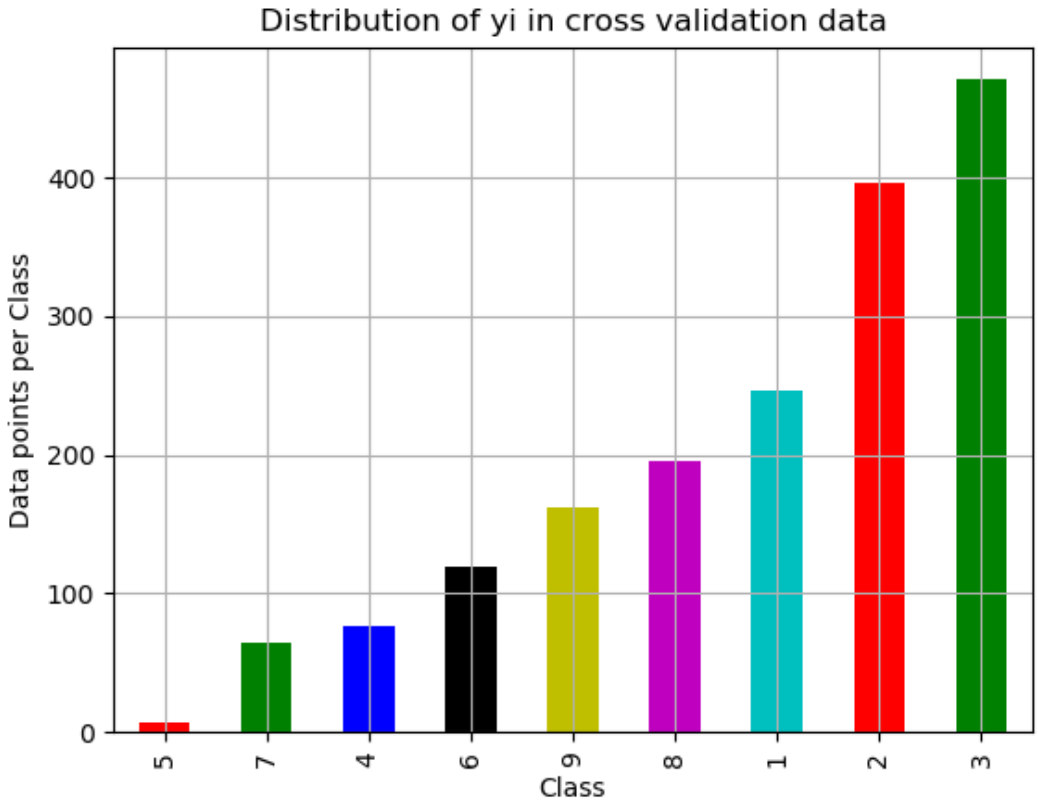
```
# it returns a dict, keys as class labels and values as the number of data points in that c
train_class_distribution = y_train.value_counts().sort_values()
test_class_distribution = y_test.value_counts().sort_values()
cv_class_distribution = y_cv.value_counts().sort_values()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.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 = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.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 = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.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], '(',
```




```
Number of data points in class 9 : 2354 ( 27.076 %)
Number of data points in class 8 : 1982 ( 22.797 %)
Number of data points in class 7 : 1233 ( 14.182 %)
Number of data points in class 6 : 982 ( 11.295 %)
Number of data points in class 5 : 810 ( 9.317 %)
Number of data points in class 4 : 601 ( 6.913 %)
Number of data points in class 3 : 380 ( 4.371 %)
Number of data points in class 2 : 318 ( 3.658 %)
Number of data points in class 1 : 34 ( 0.391 %)
--------------------------------------------------------------------------------
----
```

Distribution of yi in test data

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




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

In [3]:

```
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

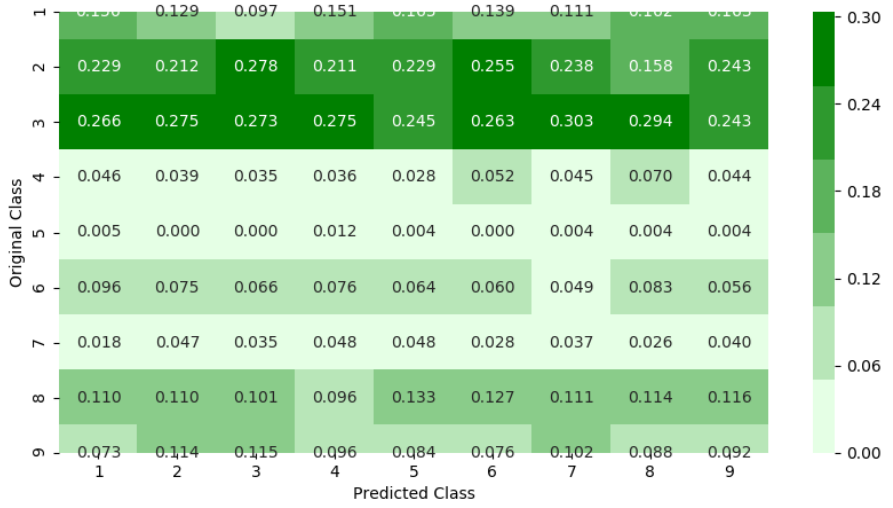
4.1.1. Random Model

In [19]:

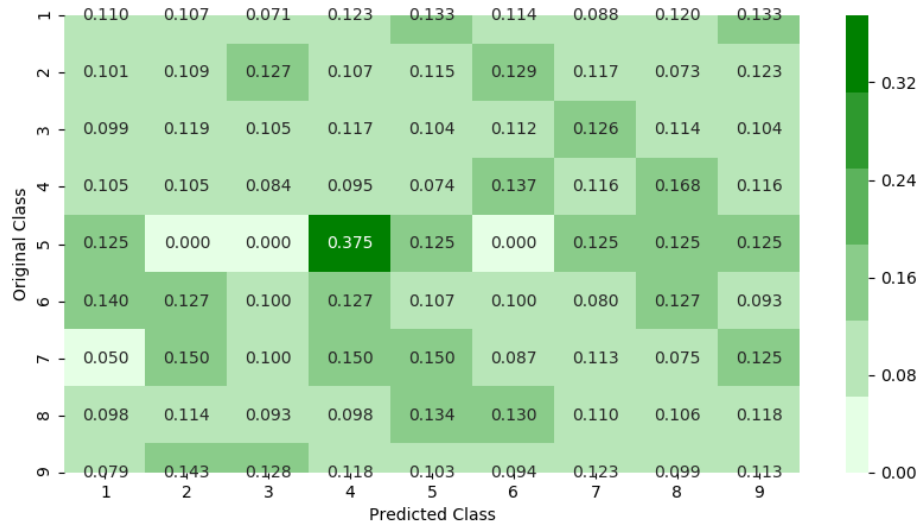
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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 2.493368164762796
Log loss on Test Data using Random Model 2.4675438569855914
Number of misclassified points  89.28242870285189
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```


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




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



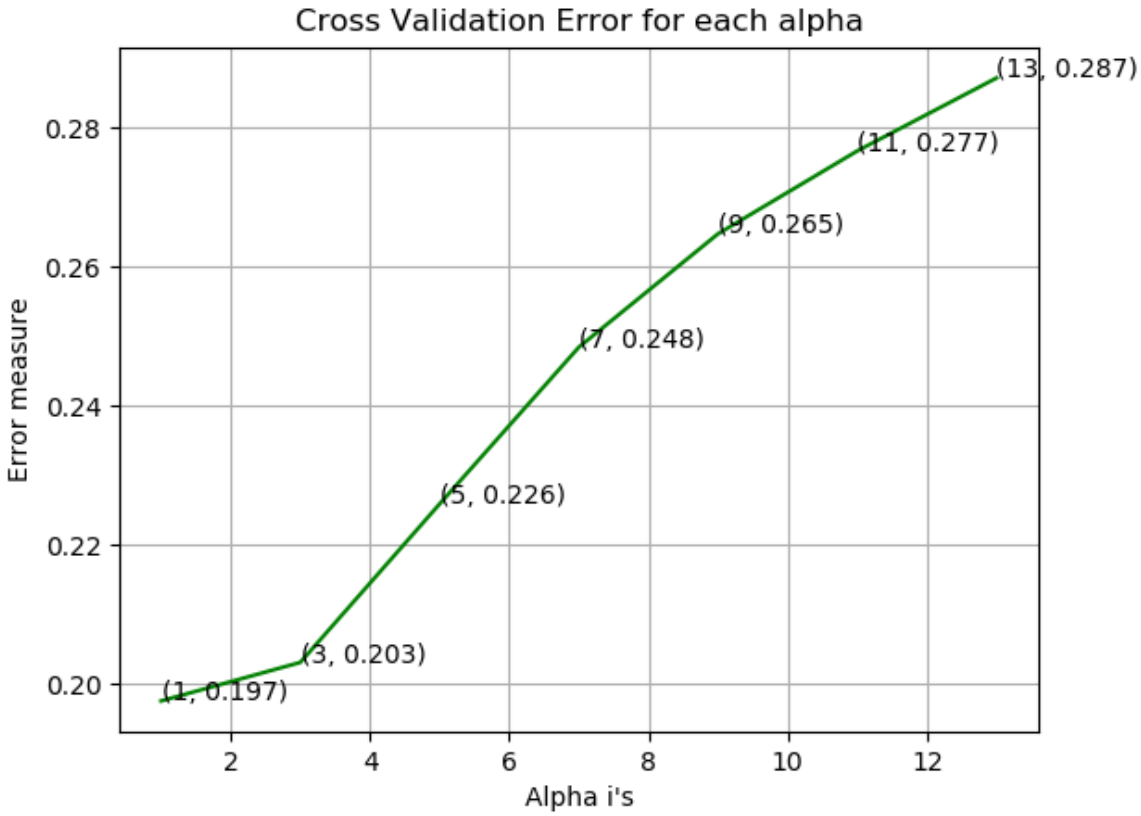

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.2. K Nearest Neighbour Classification

In [20]:

```
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))
```

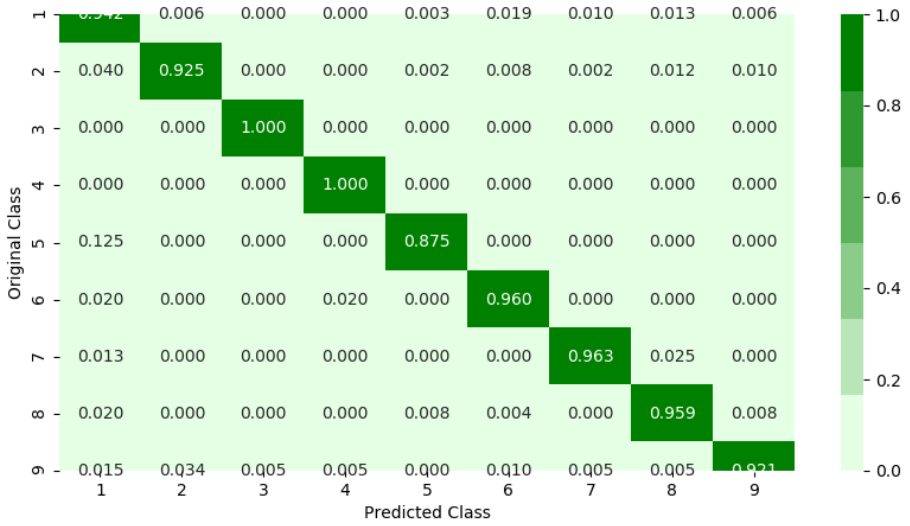
```
log_loss for k =  1 is 0.19749510055519265
log_loss for k =  3 is 0.20304154518289938
log_loss for k =  5 is 0.2258467787299477
log_loss for k =  7 is 0.24842301289926702
log_loss for k =  9 is 0.26475728593349296
log_loss for k =  11 is 0.2766905460011016
log_loss for k =  13 is 0.28719334587665374
```




```
For values of best alpha =  1 The train log loss is: 0.06962796621537387
For values of best alpha =  1 The cross validation log loss is: 0.1974951005
5519265
For values of best alpha =  1 The test log loss is: 0.2112775327628347
Number of misclassified points  4.185832566697332
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```

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


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



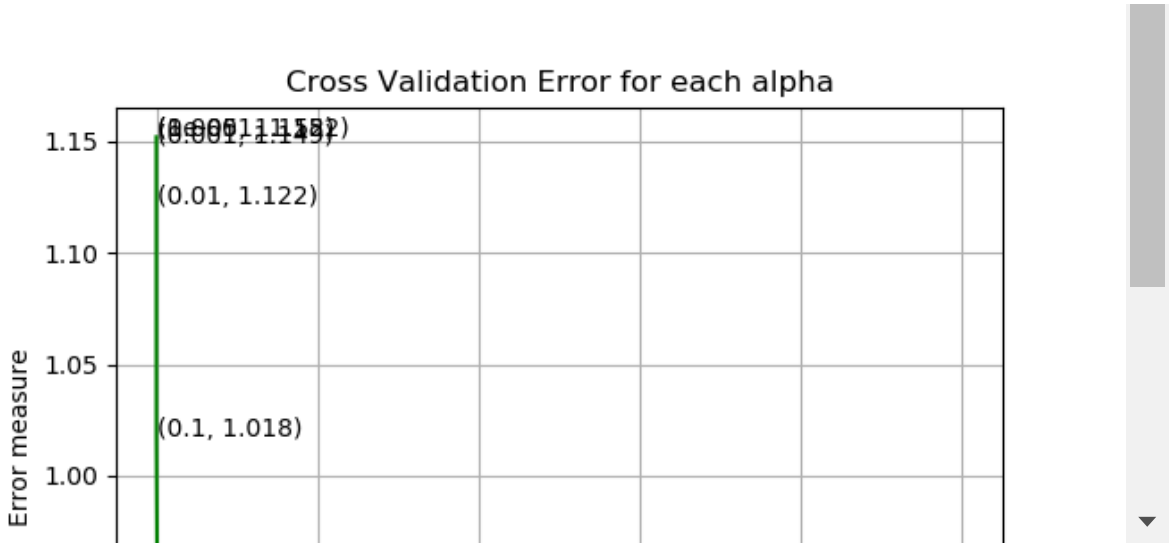

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.3. Logistic Regression

In [21]:

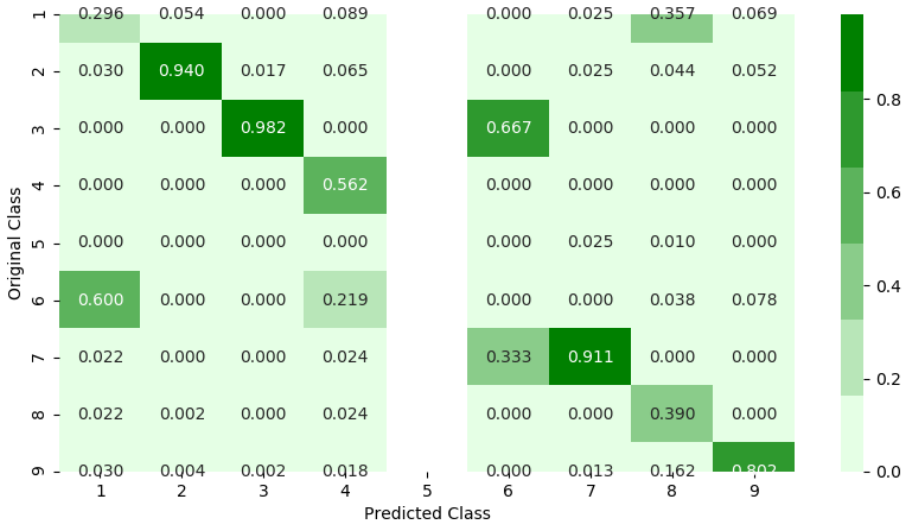
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = 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 1.1519982718652113
log_loss for c =  0.0001 is 1.1519749021488803
log_loss for c =  0.001 is 1.1487204942570848
log_loss for c =  0.01 is 1.1216331448857226
log_loss for c =  0.1 is 1.0178480181893563
log_loss for c =  1 is 0.9570996808894141
log_loss for c =  10 is 0.9028990980723989
log_loss for c =  100 is 0.8826204997478924
log_loss for c =  1000 is 0.9264962758219621
```




```
log loss for train data 0.8658518669539474
log loss for cv data 0.8826204997478924
log loss for test data 0.8750084162412813
Number of misclassified points  28.242870285188594
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```


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



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

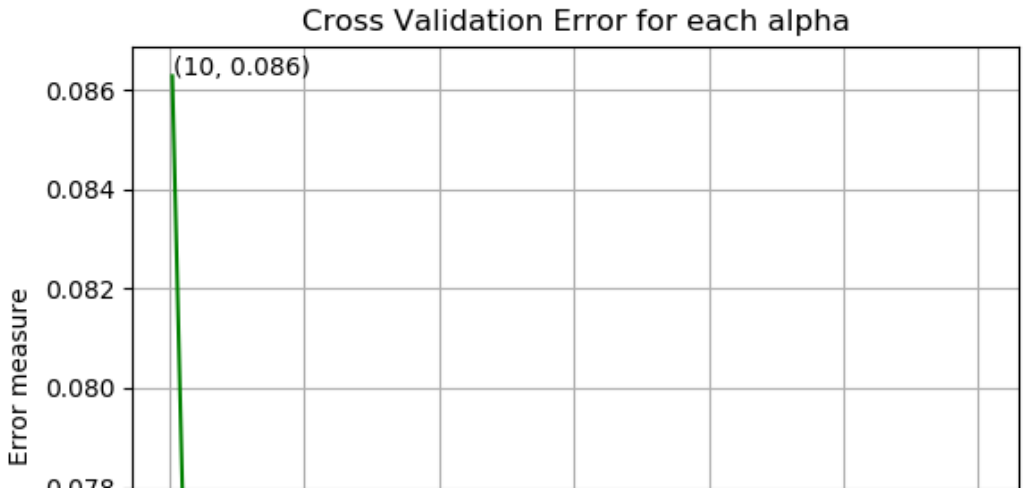

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.4. Random Forest Classifier

In [22]:

```
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 0.08628998519291152
log_loss for c =  50 is 0.07736514150715934
log_loss for c =  100 is 0.0761081403915417
log_loss for c =  500 is 0.07472203870196603
log_loss for c =  1000 is 0.07447000167682892
log_loss for c =  2000 is 0.07438670973365663
log_loss for c =  3000 is 0.07418545380358307
```




```
For values of best alpha =  3000 The train log loss is: 0.025703086924089385
For values of best alpha =  3000 The cross validation log loss is: 0.0741854
5380358307
For values of best alpha =  3000 The test log loss is: 0.07605172932410158
Number of misclassified points  1.9319227230910765
---------------------------------------------------- Confusion matrix --------
----------------------------------------
```


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


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

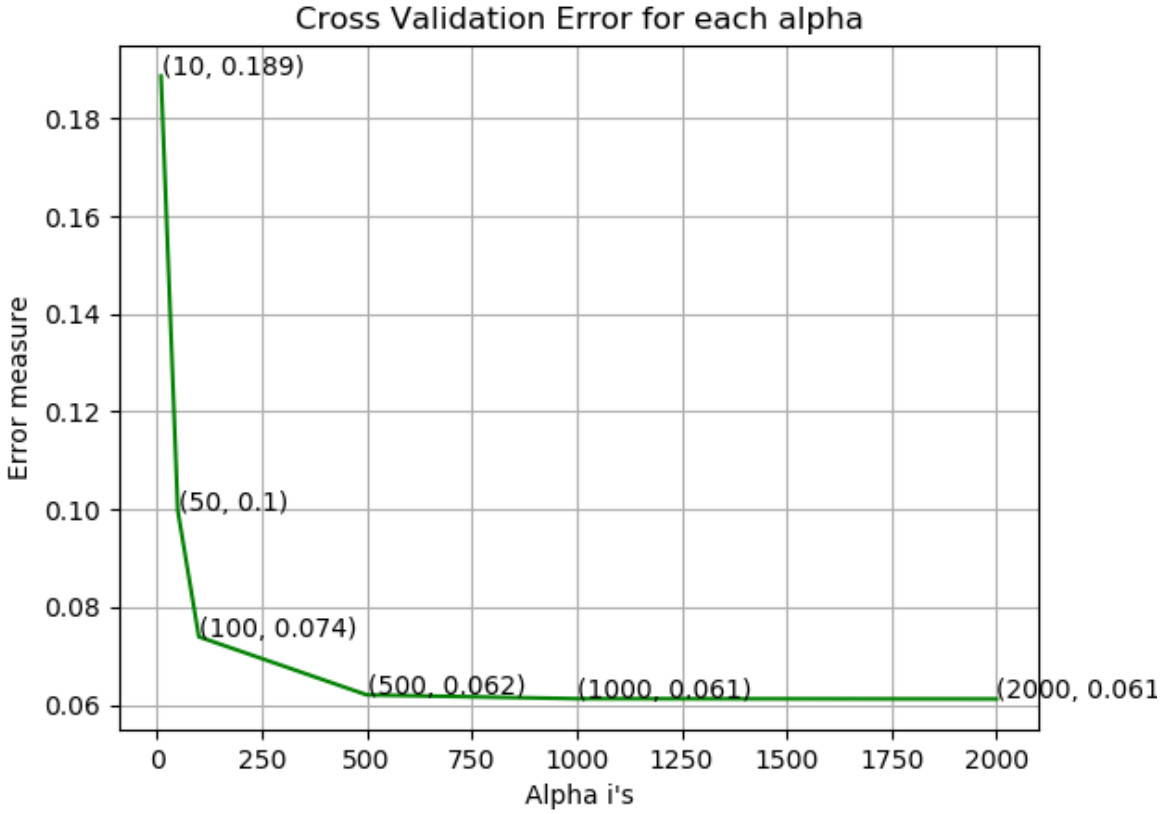

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.5. XgBoost Classification

In [23]:

```
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 0.18867884432315696
log_loss for c =  50 is 0.0997541839162173
log_loss for c =  100 is 0.0739429000115492
log_loss for c =  500 is 0.06205462745114886
log_loss for c =  1000 is 0.06127921396830704
log_loss for c =  2000 is 0.061225480976740486
```




```
For values of best alpha =  2000 The train log loss is: 0.023048242931716045
For values of best alpha =  2000 The cross validation log loss is: 0.0612254
80976740486
For values of best alpha =  2000 The test log loss is: 0.06948602664935873
Number of misclassified points  1.4259429622815087
---------------------------------------------------- Confusion matrix --------
----------------------------------------
```

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


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


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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [24]:

```
# 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 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 13.2min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 39.6min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 49.0min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 56.3min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 62.4min remaining: 13.
7min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 67.0min remaining:  4.
3min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 67.4min finished
```

Out[24]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step
=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1...
                                           seed=None, silent=None, subsample
=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5,
1],
                                       'learning_rate': [0.01, 0.03, 0.05,
0.1,
                                                         0.15, 0.2],
                                       'max_depth': [3, 5, 10],
                                       'n_estimators': [100, 200, 500, 100
0,
                                                        2000],
                                       'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [25]:

```
print (random_cfl1.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 5, 'learning_rate': 0.
03, 'colsample_bytree': 0.5}
```

In [26]:

```
# 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=500, learning_rate=0.15, 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 0.02282986512900174
cv loss 0.061403714439339445
test loss 0.06969943174202615
```

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 [27]:

```
'''#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')'''
```

Out[27]:

```
"#intially create five folders\n#first \n#second\n#thrid\n#fourth\n#fifth\n#
this code tells us about random split of files into five folders\nfolder_1
='first'\nfolder_2 ='second'\nfolder_3 ='third'\nfolder_4 ='fourth'\nfolder_
5 ='fifth'\nfolder_6 = 'output'\nfor i in [folder_1,folder_2,folder_3,folder
_4,folder_5,folder_6]:\n    if not os.path.isdir(i):\n        os.makedirs(i)
\n\nsource='train/'\nfiles = os.listdir('train')\nID=df['Id'].tolist()\ndata
=range(0,10868)\nr.shuffle(data)\ncount=0\nfor i in range(0,10868):\n    if
i % 5==0:\n        shutil.move(source+files[data[i]],'first')\n    elif i%5=
=1:\n        shutil.move(source+files[data[i]],'second')\n    elif i%5 ==
2:\n        shutil.move(source+files[data[i]],'thrid')\n    elif i%5 ==3:\n
shutil.move(source+files[data[i]],'fourth')\n    elif i%5==4:\n        shuti
l.move(source+files[data[i]],'fifth')"
```

In [17]:

```
#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 [7]:

```
dfasm=pd.read_csv("data/asmoutputfile.csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='left')
result_asm.head()
```

Out[7]:

	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

4.2.1.1 Files sizes of each .asm file

In [8]:

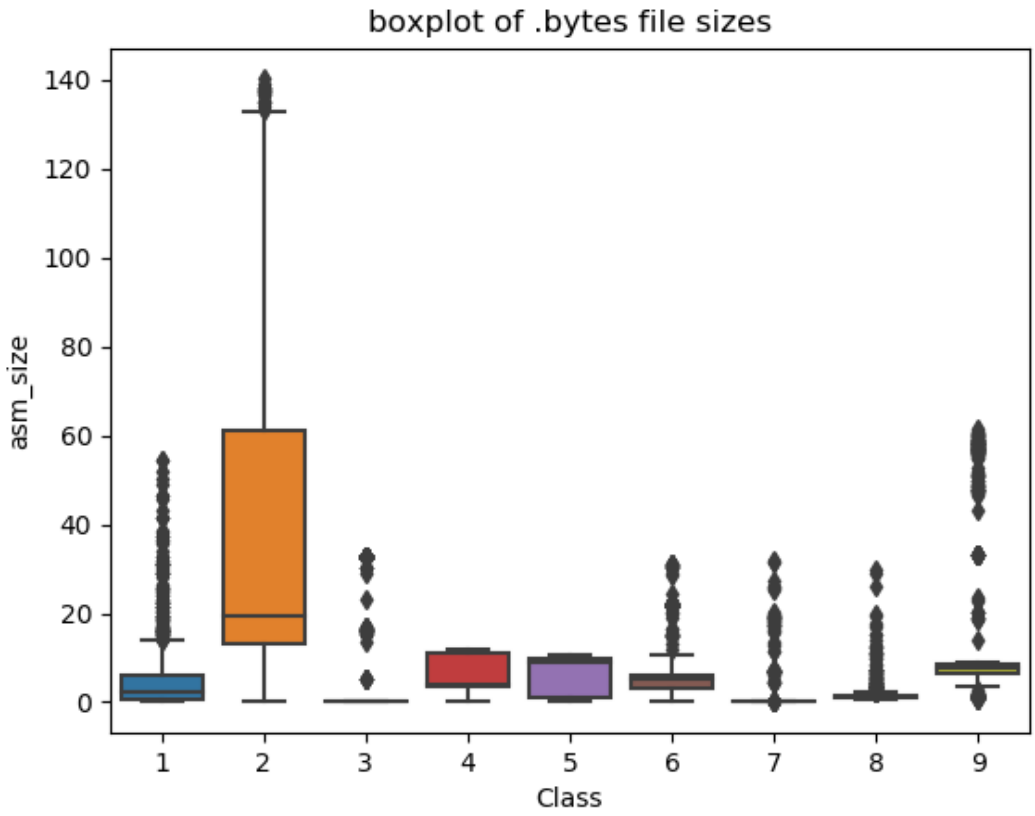
```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1,
    # st_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,'asm_size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

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

4.2.1.2 Distribution of .asm file sizes

In [46]:

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



In [9]:

```
# 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[9]:

	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 [10]:

```
result_asm.to_csv('data/asm_feat_size.csv')
```

In [18]:

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

Out[18]:

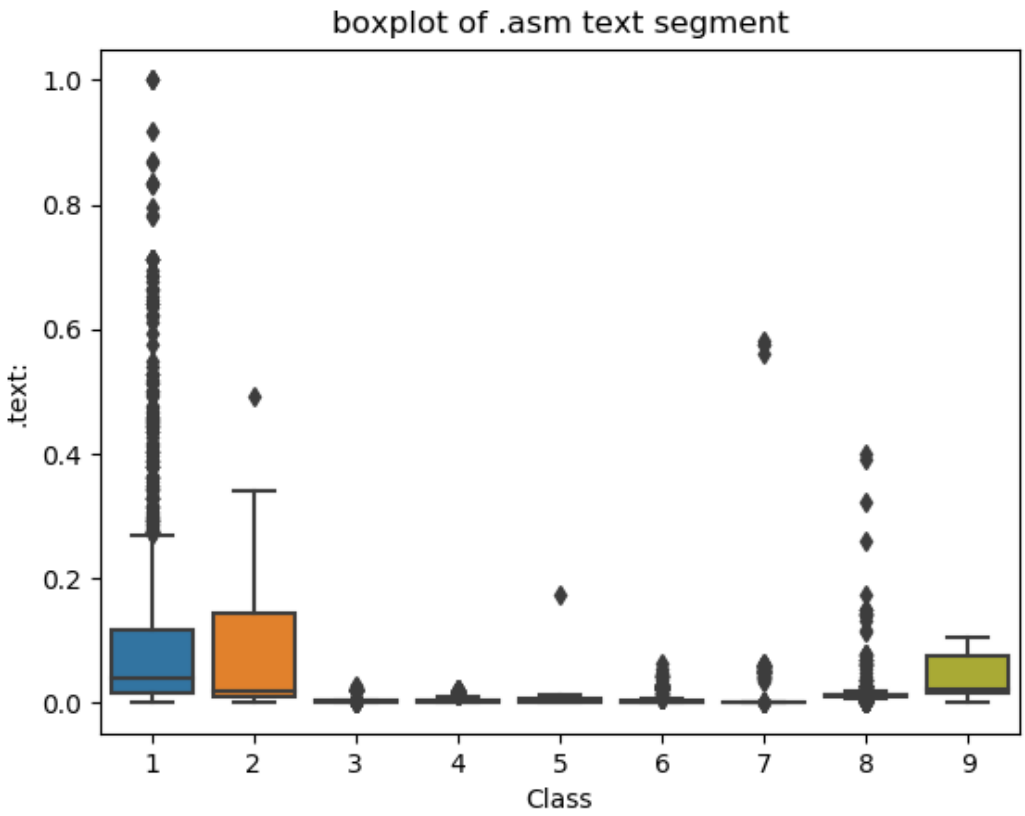
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.e
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	
3	3X2nY7iQaPBIWDrAZqJe	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 [48]:

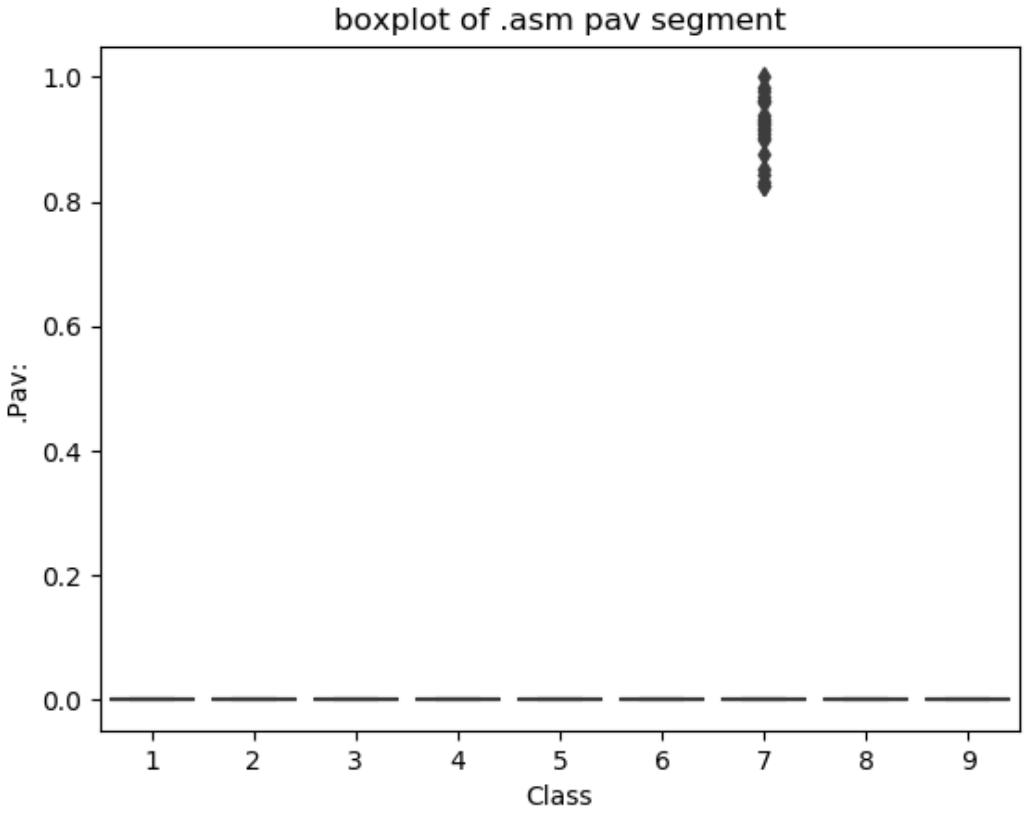
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```



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

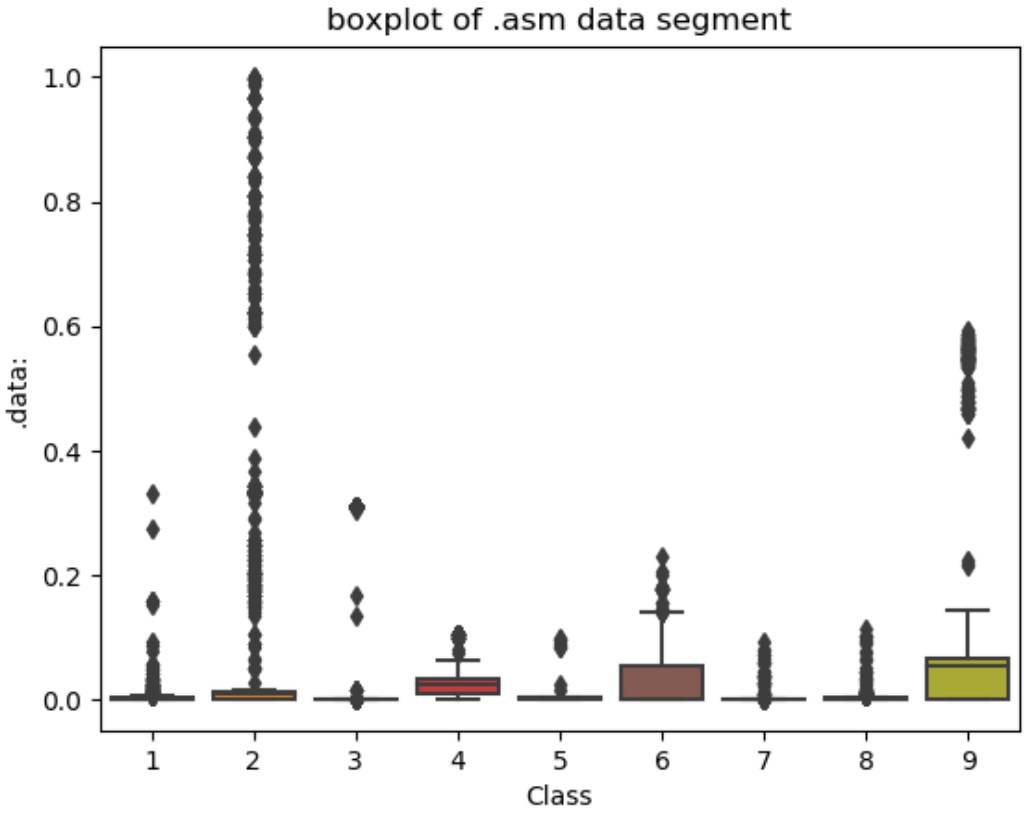
In [189]:

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



In [49]:

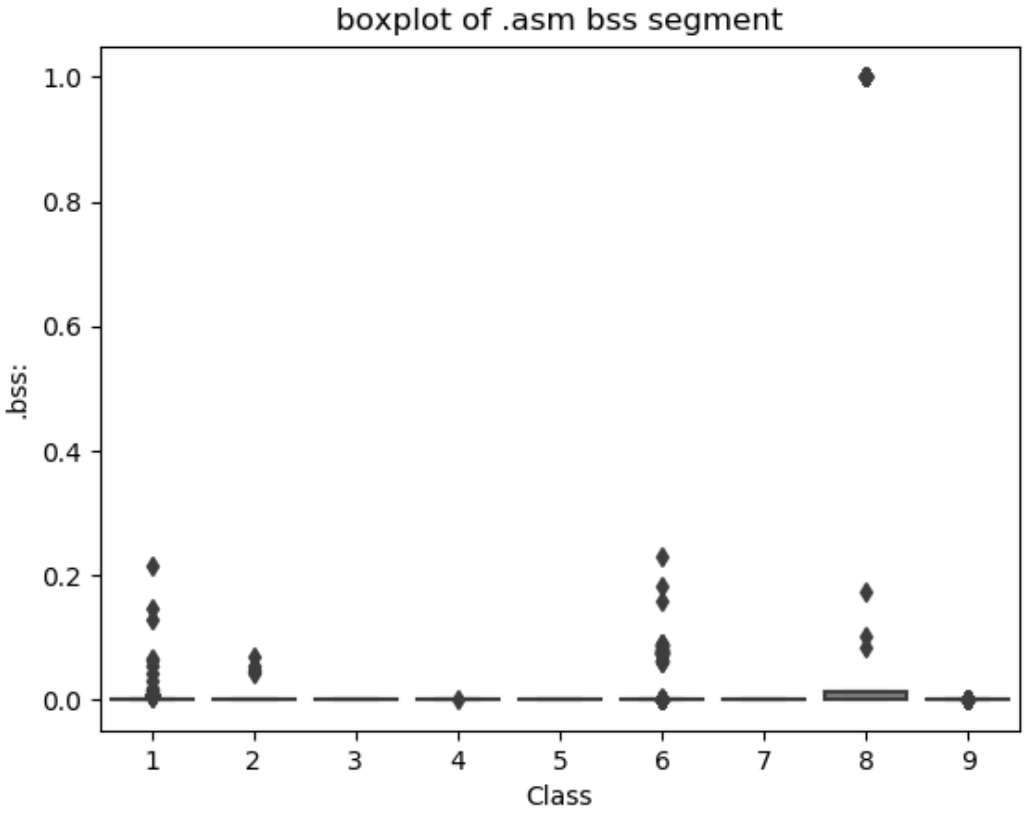
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



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

In [50]:

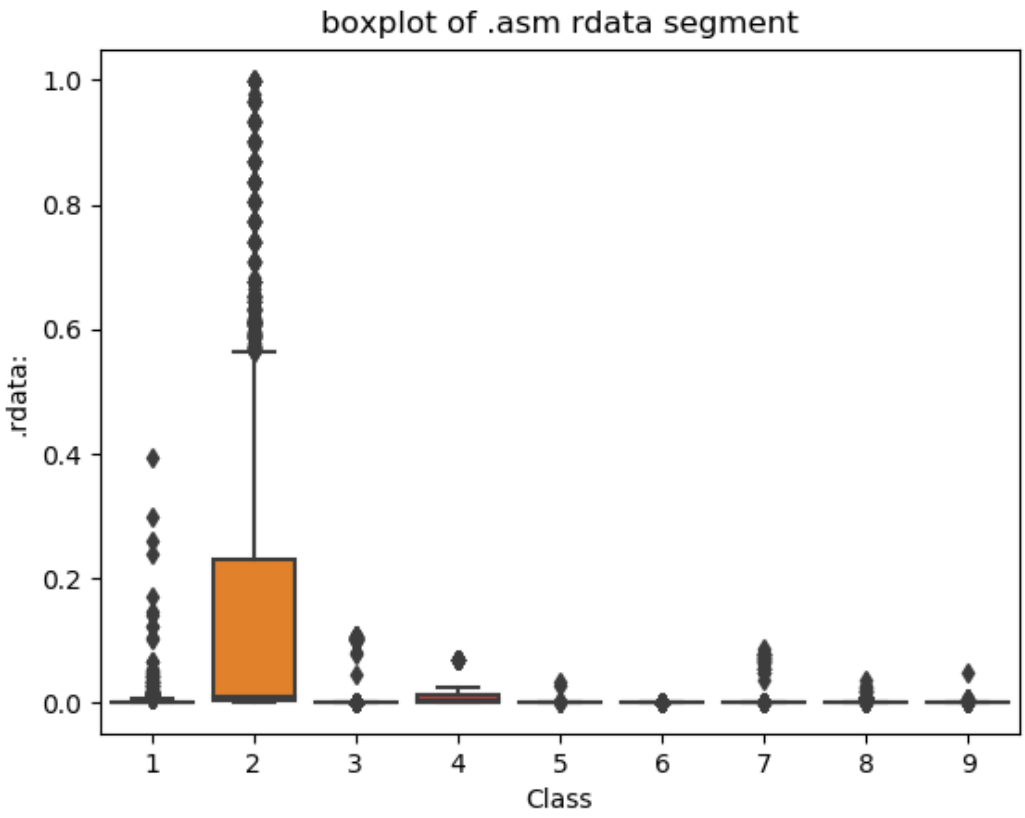
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [51]:

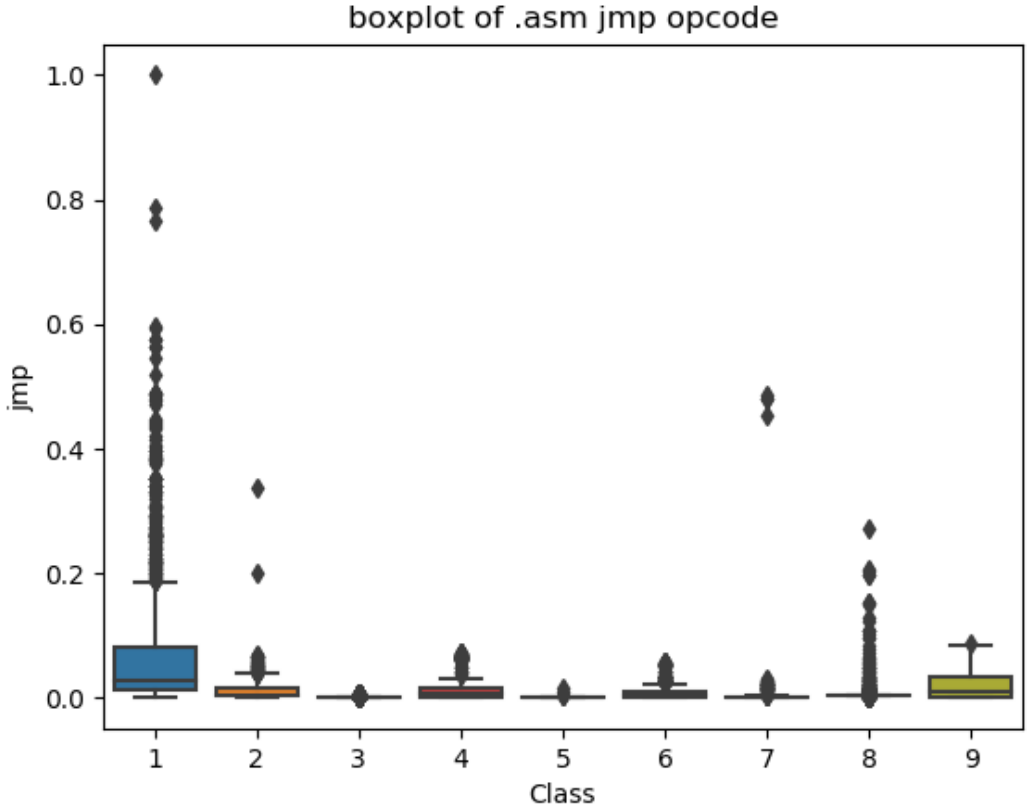
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

In [52]:

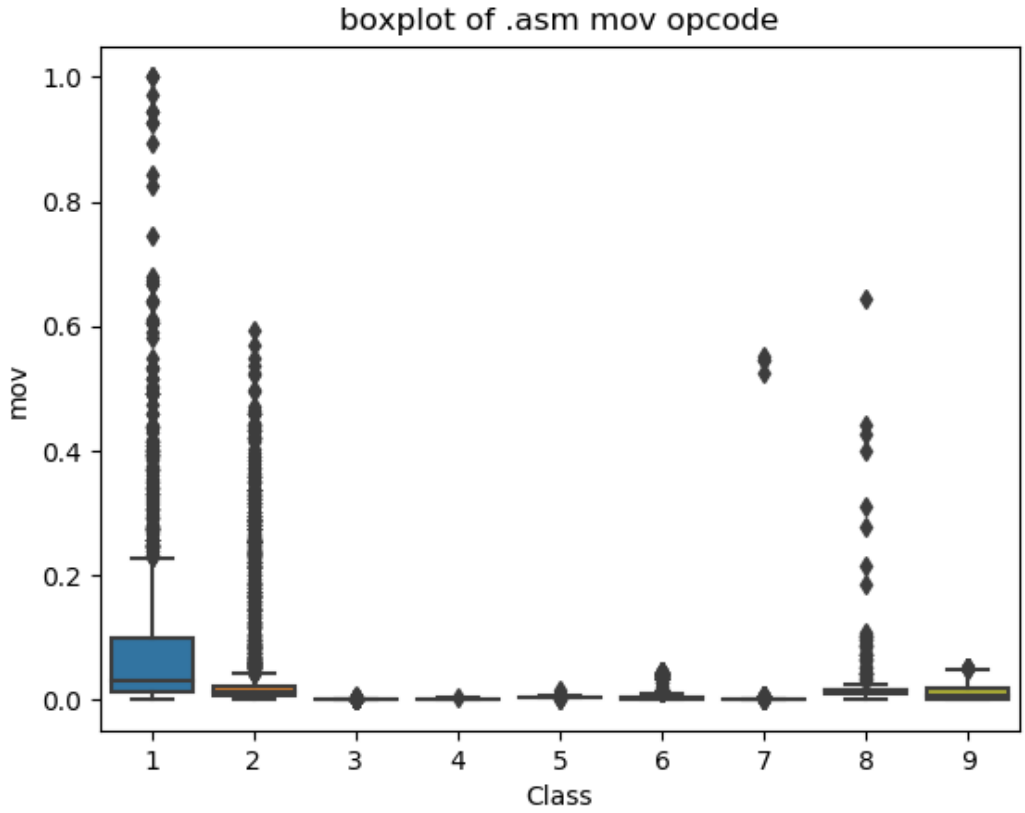
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



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

In [53]:

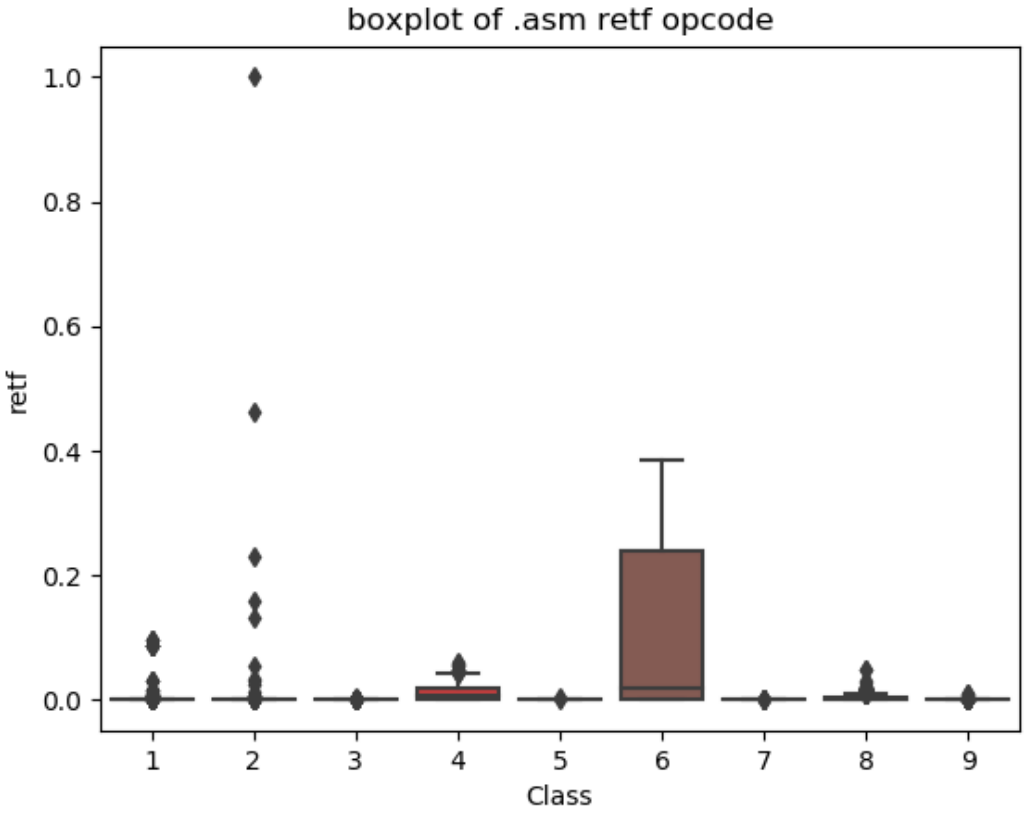
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



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

In [41]:

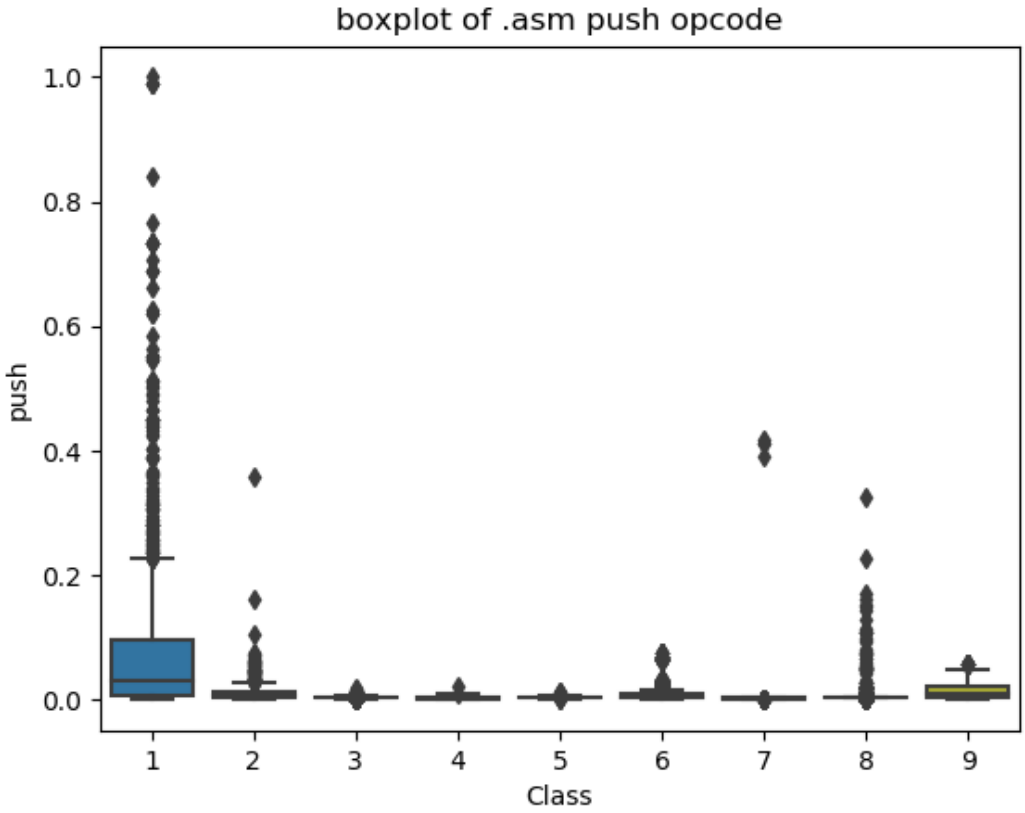
```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```



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

In [42]:

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

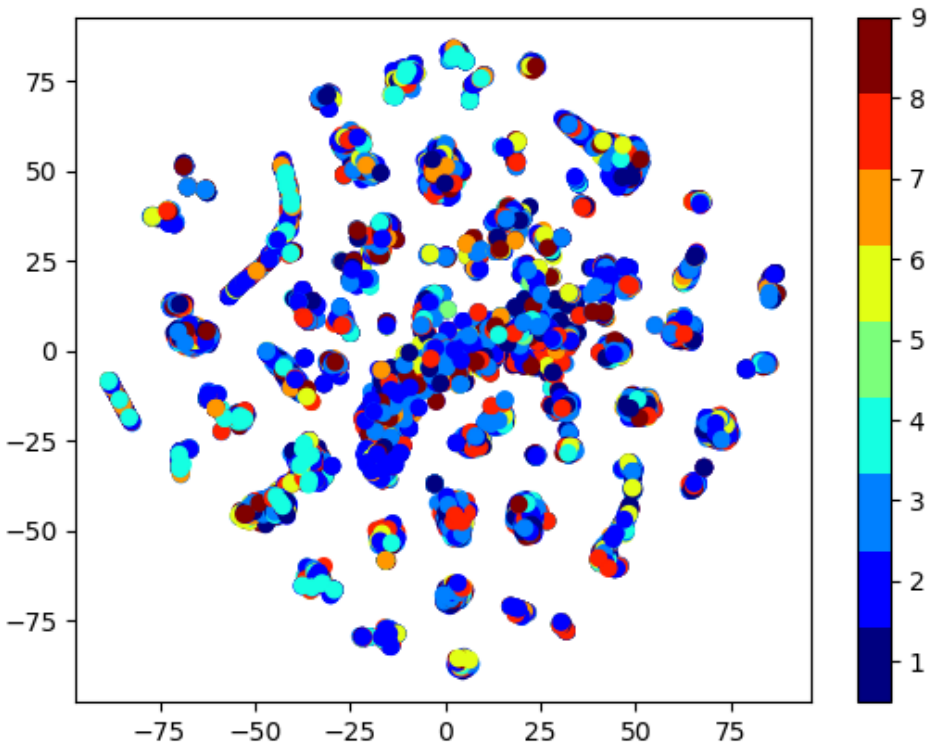


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

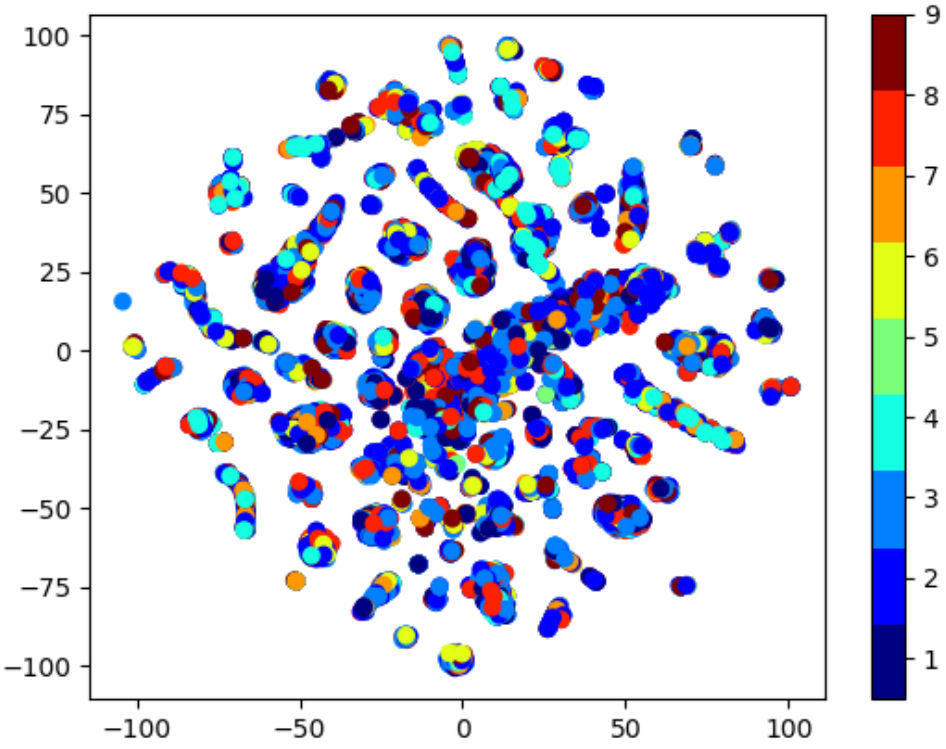
In [43]:

```
# 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()
```



In [44]:

```
# 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()
```



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 [17]:

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

In [18]:

```
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 [19]:

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

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

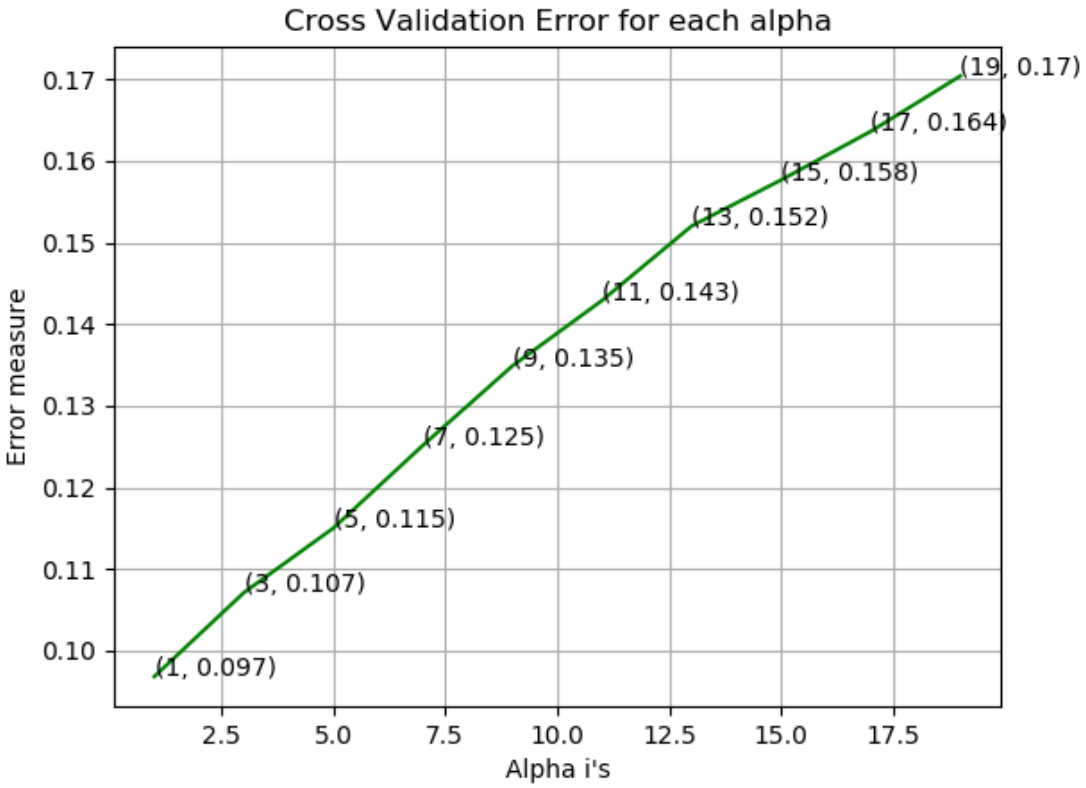
4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

In [48]:

```
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 0.09682793472561187
log_loss for k =  3 is 0.10711122311810306
log_loss for k =  5 is 0.11502846605417708
log_loss for k =  7 is 0.12516846403798176
log_loss for k =  9 is 0.13490688438509477
log_loss for k =  11 is 0.14291697005700715
log_loss for k =  13 is 0.15204082777855393
log_loss for k =  15 is 0.1576867582314274
log_loss for k =  17 is 0.16363407748019593
log_loss for k =  19 is 0.17042429588622443
```




```
log loss for train data 0.026132663468517604
log loss for cv data 0.09682793472561187
log loss for test data 0.09924949281638122
Number of misclassified points  1.6559337626494939
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```


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


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

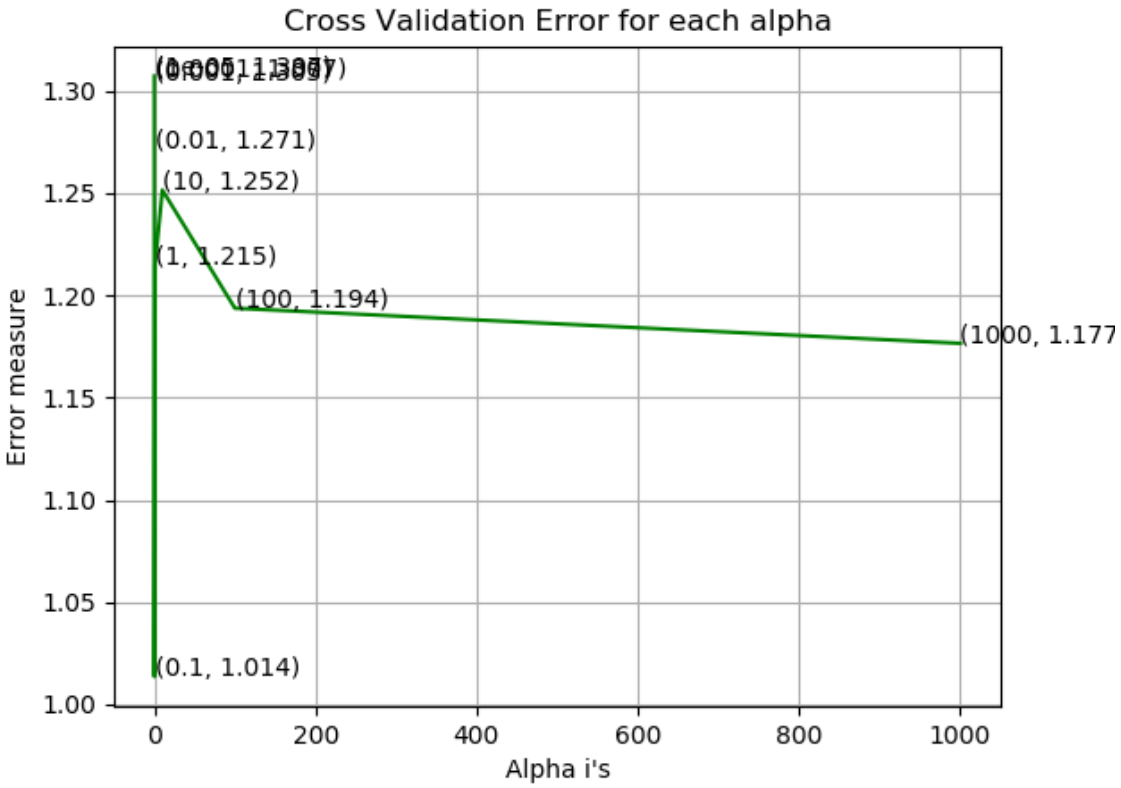

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.2 Logistic Regression

In [49]:

```
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 1.307253942752918
log_loss for c =  0.0001 is 1.3070432964917558
log_loss for c =  0.001 is 1.3046545580767734
log_loss for c =  0.01 is 1.2714589382416652
log_loss for c =  0.1 is 1.0136190009409562
log_loss for c =  1 is 1.21508361120584
log_loss for c =  10 is 1.2515035880485939
log_loss for c =  100 is 1.1937260816729645
log_loss for c =  1000 is 1.1765507462807978
```




```
log loss for train data 1.000575944129987
log loss for cv data 1.0136190009409562
log loss for test data 0.9761904970319099
Number of misclassified points  27.00091996320147
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```


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


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

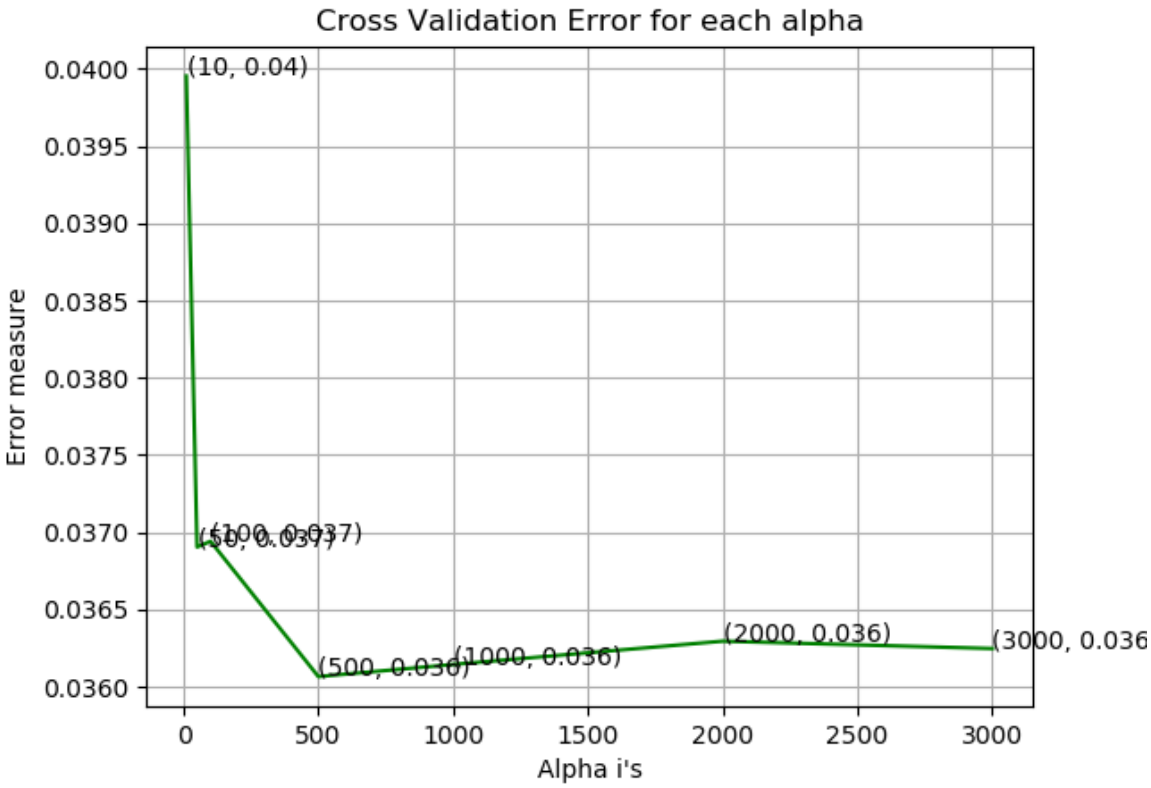

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.3 Random Forest Classifier

In [50]:

```
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 0.03995385126105277
log_loss for c =  50 is 0.03690480348075474
log_loss for c =  100 is 0.036941912047109365
log_loss for c =  500 is 0.03606615756589014
log_loss for c =  1000 is 0.036144702599728074
log_loss for c =  2000 is 0.03629543255832047
log_loss for c =  3000 is 0.036247629135674364
```




```
log loss for train data 0.015066085140636946
log loss for cv data 0.03606615756589014
log loss for test data 0.03282513910965592
Number of misclassified points  0.6439742410303588
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```


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


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

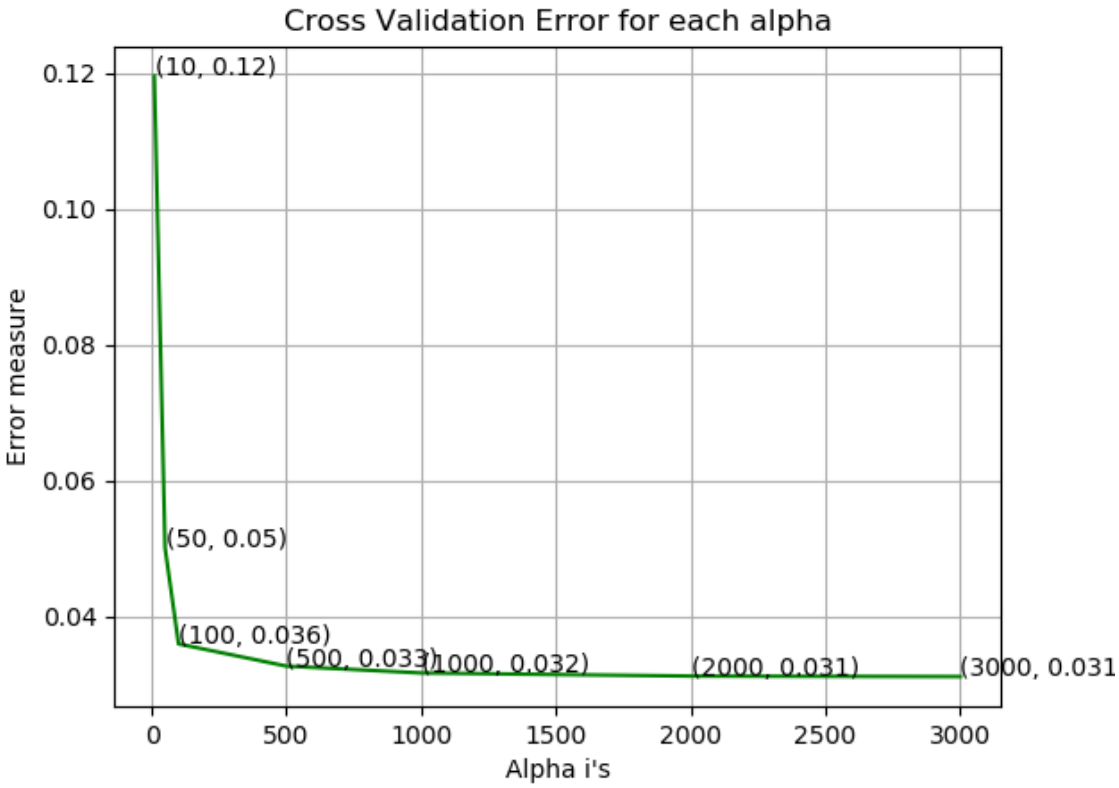

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.4 XgBoost Classifier

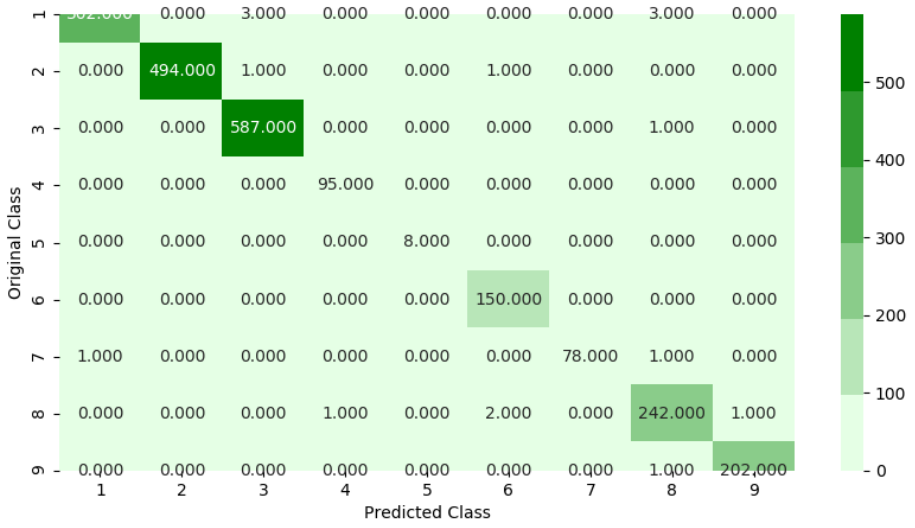
In [51]:

```
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 0.11959651251293399
log_loss for c =  50 is 0.050065886358793464
log_loss for c =  100 is 0.035941398024741396
log_loss for c =  500 is 0.03267380516972519
log_loss for c =  1000 is 0.03166026993240983
log_loss for c =  2000 is 0.031202178751351846
log_loss for c =  3000 is 0.031126262282496372
```




```
For values of best alpha =  3000 The train log loss is: 0.012003756665255189
For values of best alpha =  3000 The cross validation log loss is: 0.0311262
62282496372
For values of best alpha =  3000 The test log loss is: 0.03390721259959973
Number of misclassified points  0.7359705611775529
---------------------------------------------------- Confusion matrix --------
----------------------------------------
```




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


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


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

4.4.5 Xgboost Classifier with best hyperparameters

In [52]:

```
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 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   11.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   42.1s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed:  3.5min remaining:   4
5.7s
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed:  3.6min remaining:   1
3.8s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  3.7min finished
```

Out[52]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step
=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1...
                                           seed=None, silent=None, subsample
=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5,
1],
                                       'learning_rate': [0.01, 0.03, 0.05,
0.1,
                                                         0.15, 0.2],
                                       'max_depth': [3, 5, 10],
                                       'n_estimators': [100, 200, 500, 100
0,
                                                        2000],
                                       'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [53]:

```
print (random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.
1, 'colsample_bytree': 0.5}
```

In [27]:

```
x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.05,colsample_bytree=0.3,
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))
plot_confusion_matrix(y_test_asm, c_cfl.predict(X_test_asm))
```

```
train loss 0.01615163262186882
cv loss 0.035634383241981586
test loss 0.046460211505267066
Number of misclassified points  1.011959521619135
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```


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

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


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

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

4.5.1. Merging both asm and byte file features

In [20]:

```
result.head()
```

Out[20]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.0020
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0047
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0050
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0003
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0001

5 rows × 260 columns

In [21]:

```
result_asm.head()
```

Out[21]:

	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 [22]:

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

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

In [23]:

```
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[23]:

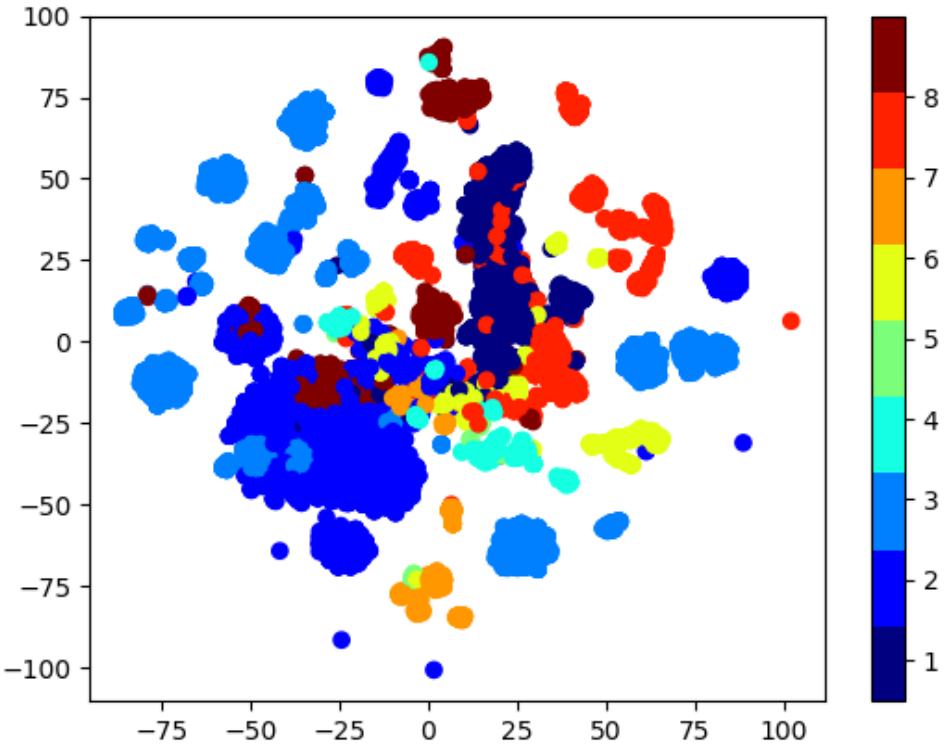
	0	1	2	3	4	5	6	7	8	
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0
3	0.009209	0.001708	0.000404	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 [59]:

```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```



4.5.3. Train and Test split

In [24]:

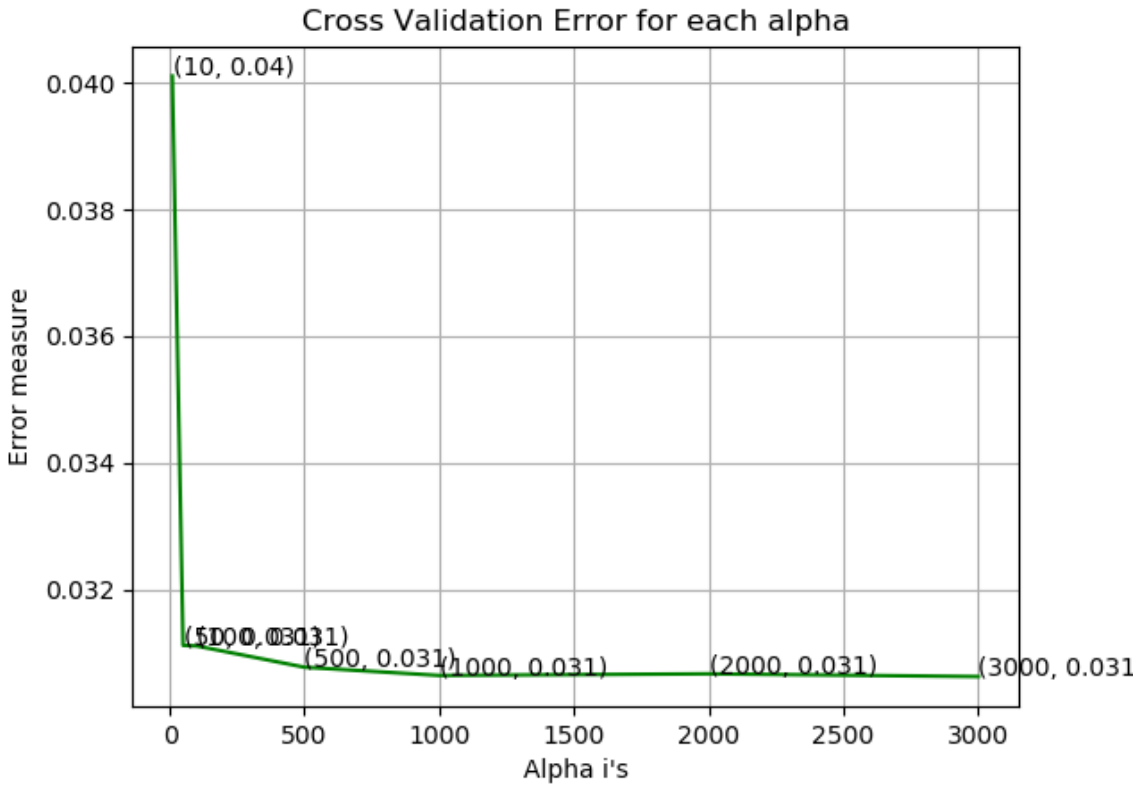
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,st
```

4.5.4. Random Forest Classifier on final features

In [61]:

```
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 0.04010334524774437
log_loss for c =  50 is 0.031125592348978843
log_loss for c =  100 is 0.03111712937132961
log_loss for c =  500 is 0.0307817219656332
log_loss for c =  1000 is 0.030651717983497927
log_loss for c =  2000 is 0.03067763267671245
log_loss for c =  3000 is 0.030633869038434843
```




```
For values of best alpha =  3000 The train log loss is: 0.01686240657446133
For values of best alpha =  3000 The cross validation log loss is: 0.0306338
69038434843
For values of best alpha =  3000 The test log loss is: 0.04577255153200972
```

4.5.5. XgBoost Classifier on final features

In [62]:

```
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 0.07877851917097604
log_loss for c =  50 is 0.034955361801798354
log_loss for c =  100 is 0.02918490777987974
log_loss for c =  500 is 0.027533094151848996
log_loss for c =  1000 is 0.02761627141137908
log_loss for c =  2000 is 0.02973224696252711
log_loss for c =  3000 is 0.029737618415905435


For values of best alpha =  500 The train log loss is: 0.012367745990822613
For values of best alpha =  500 The cross validation log loss is: 0.02973761
8415905435
For values of best alpha =  500 The test log loss is: 0.035017557790415314
```

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

In [63]:

```
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)
random_cfl.fit(X_train_merge, y_train_merge)
```

```
                                          objective='binary:logistic',
                                          random_state=0, reg_alpha=0,
                                          reg_lambda=1...
                                          seed=None, silent=None, subsamp
le=1,
                                          verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.
5, 1],
                                        'learning_rate': [0.01, 0.03, 0.0
5, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 10
00,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [64]:

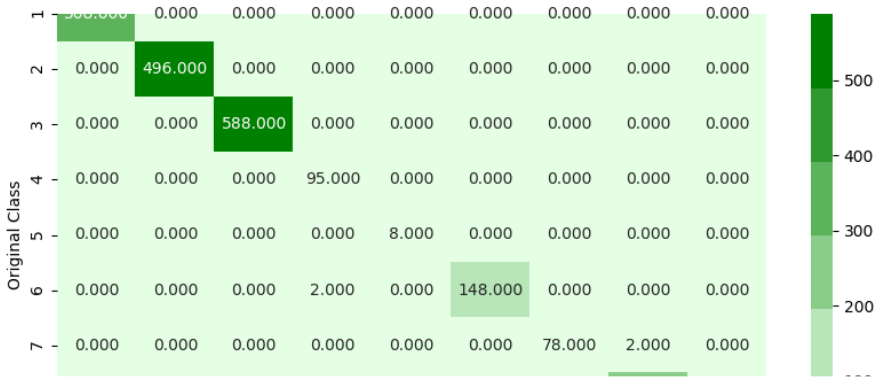
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.0
5, 'colsample_bytree': 0.5}
```

In [90]:

```
x_cfl=XGBClassifier(n_estimators=2000,max_depth=3,learning_rate=0.05,colsample_bytree=1,sub
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 ('train loss',log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_merge, sig_clf.predict(X_test_merge))
```

```
train loss 0.01258649505086597
cv loss 0.03156271911278616
test loss 0.03450578167268943
Number of misclassified points  0.5059797608095675
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```



**********Assignment**********

reference:https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.ipynb (https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.ipynb)

Bi-gram on byte file

In [8]:

```
byte_vocab = "0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,

#creating bi-gram features
def byte_bigram_features(vocab):
    byte_bigram_vocab=[]
    for vo in (vocab.split(",")):
        for j in range(len(vocab.split(","))):
            byte_bigram_vocab.append(vo+" "+vocab.split(',')[j])
    return byte_bigram_vocab

bi_vocab = byte_bigram_features(byte_vocab)
print("Bi_vocab size", len(bi_vocab))
```

```
Bi_vocab size 66049
```

In [9]:

```
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
import scipy

def extract_byte_bigram(thread_file):

    files = thread_file
    fname = []

    #defining sparse matrix to strore sparse vectors
    features = scipy.sparse.csr_matrix((0, 66049))

    vectorizer = CountVectorizer(lowercase=True, ngram_range=(2, 2), vocabulary=bi_vocab)

    for file in tqdm(files):

        if (file.endswith(".bytes")):
            with open("byteFiles/"+file) as byte_file:

                file_id = file.split('.')[0]

                fname.append(file_id)

                file_data = []

                file_data.append(byte_file.read().replace("\n"," ").lower())

                vectors = vectorizer.fit_transform(file_data)

                features = scipy.sparse.vstack([features,vectors])

            byte_file.close()

    scipy.sparse.save_npz("data/byte_bigram.npz",features)

    return fname
```

In [10]:

```
threadFiles = os.listdir('byteFiles')
byte_file_names = extract_byte_bigram(threadFiles)
```

```
100%|████████████████████████████████████████████████████████████████
██| 10868/10868 [25:46:51<00:00,  8.54s/it]
```

Bi gram on opcode ASM file

In [2]:

```
opcodes = 'add,al,bt,call,cdq,cld,cli,cmc,cmp,const,cwd,daa,db,dd,dec,dw,endp,ends,faddp,fc

#creating bi-gram features
def asm_bigram_features(vocab):
    asm_bigram_vocab = []
    for vo in (vocab.split(",")):
        for j in range(len(vocab.split(","))):
            asm_bigram_vocab.append(vo+" "+vocab.split(',')[j])
    return asm_bigram_vocab
```

In [3]:

```
bi_vocab_asm = asm_bigram_features(opcodes)
print("Bi_vocab asm sample", len(bi_vocab_asm))
```

```
Bi_vocab asm sample 8649
```

In [6]:

```
from tqdm import tqdm
def extract_asm_bigram(thread_file):
    files = thread_file
    fname = []
    features = scipy.sparse.csr_matrix((0,8649))

    vectorizer = CountVectorizer(ngram_range=(2, 2), vocabulary=bi_vocab_asm)

    for file in tqdm(files):
        if (file.endswith(".asm")):
            file_id = file.split('.')[0]
            fname.append(file_id)
            with open("asmFiles/"+file,encoding='latin-1') as asm_file:
                file_data = []
                file_data.append(asm_file.read().replace("\n"," ").lower())

                vector = vectorizer.fit_transform(file_data)

                features = scipy.sparse.vstack([features,vector])

            asm_file.close()

    scipy.sparse.save_npz("data/asm_bigram.npz",features)
    return fname
```

In [7]:

```
threadFiles = os.listdir('asmfiles')
asm_file_names = extract_asm_bigram(threadFiles)
```

```
100%|██████████████████████████████████████████████████████████████████████
███| 10868/10868 [6:04:11<00:00,  2.01s/it]
```

tri-gram on asm opcode

In [15]:

```
opcodes = 'add,al,bt,call,cdq,cld,cli,cmc,cmp,const,cwd,daa,db,dd,dec,dw,endp,ends,faddp,fc

#creating bi-gram features
def asm_trigram_features(vocab):
    asm_trigram_vocab = []
    for vo in (vocab.split(",")):
        for j in range(len(vocab.split(","))):
            for k in range(len(vocab.split(","))):
                asm_trigram_vocab.append(vo+" "+vocab.split(',')[j]+" "+vocab.split(',')[k]
    return asm_trigram_vocab
```

In [16]:

```
tri_vocab_asm = asm_trigram_features(opcodes)
print("tri_vocab size", len(tri_vocab_asm))
```

```
tri_vocab size 804357
```

In [17]:

```
from tqdm import tqdm
def extract_asm_trigram(thread_file):
    files = thread_file
    fname = []
    features = scipy.sparse.csr_matrix((0,804))

    vectorizer = CountVectorizer(ngram_range=(3, 3), vocabulary=tri_vocab_asm)

    for file in tqdm(files):
        if (file.endswith(".asm")):
            file_id = file.split('.')[0]
            fname.append(file_id)
            with open("asmFiles/"+file,encoding='latin-1') as asm_file:
                file_data = []
                file_data.append(asm_file.read().replace("\n"," ").lower())

                vector = vectorizer.fit_transform(file_data)

                features = scipy.sparse.vstack([features,vector])

            asm_file.close()

    scipy.sparse.save_npz("data/asm_trigram.npz",features)
    return fname
```

In [18]:

```
threadFiles = os.listdir('asmfiles')
extract_asm_trigram(threadFiles)
```

```
100%|████████████████████████████████████████████████████████████████
███| 10868/10868 [7:55:41<00:00,  2.63s/it]
```

Byte Image feature

In [5]:

```
import array
def read_image(filename):
    f = open(filename,'rb')
    ln = os.path.getsize(filename) # length of file in bytes
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
    a.fromfile(f,ln-rem)
    f.close()
    g = np.reshape(a,(len(a)//width,width))
    g = np.uint8(g)
    g = np.resize(g,(1000,))
    return list(g)
```

In [7]:

```
from csv import writer
# Do byte image extraction
def extract_byte_image_features(tfiles):
    fname = [i.split('.')[0] for i in tfiles if '.bytes' in i]
    byte_files = [i for i in tfiles if '.bytes' in i]

    ftot = len(byte_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/image-features-byte.csv'
    print('feature file:', feature_file)

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names =['ID']+ [("BYTE_{:s}".format(str(x))) for x in range(1000)]
        fw.writerow(column_names)
        for idx, fname in enumerate(byte_files):
            file_id = fname.split('.')[0]
            image_data = read_image(ext_drive + fname)
            outrows.append([file_id]+image_data)

            # Print progress
            if (idx+1) % 10 == 0:
                    print(pid, idx + 1, 'of', ftot, 'files processed.')
                    fw.writerows(outrows)
                    outrows = []

        # Write remaining files
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []

    return fname
```

In [8]:

```
ext_drive = 'byteFiles/'
tfiles = os.listdir(ext_drive)
byte_imgf_name  = extract_byte_image_features(tfiles)
```

```
Process id: 31472
feature file: data/image-features-byte.csv
31472 10 of 10868 files processed.
31472 20 of 10868 files processed.
31472 30 of 10868 files processed.
31472 40 of 10868 files processed.
31472 50 of 10868 files processed.
31472 60 of 10868 files processed.
31472 70 of 10868 files processed.
31472 80 of 10868 files processed.
31472 90 of 10868 files processed.
31472 100 of 10868 files processed.
31472 110 of 10868 files processed.
31472 120 of 10868 files processed.
31472 130 of 10868 files processed.
31472 140 of 10868 files processed.
31472 150 of 10868 files processed.
31472 160 of 10868 files processed.
31472 170 of 10868 files processed.
```

ASM Image feature

In [9]:

```
def extract_asm_image_features(tfiles):
    fname = [i.split('.')[0] for i in tfiles if '.asm' in i]
    asm_files = [i for i in tfiles if '.asm' in i]

    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/image-features-asm.csv'
    print('feature file:', feature_file)

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names =['ID']+ [("ASM_{:s}".format(str(x))) for x in range(1000)]
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image(ext_drive + fname)
            outrows.append([file_id]+image_data)

            # Print progress
            if (idx+1) % 10 == 0:
                    print(pid, idx + 1, 'of', ftot, 'files processed.')
                    fw.writerows(outrows)
                    outrows = []

        # Write remaining files
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []

    return fname
```

In [10]:

```
ext_drive = 'asmFiles/'
tfiles = os.listdir(ext_drive)
asm_imgf_name =  extract_asm_image_features(tfiles)
```

```
Process id: 31472
feature file: data/image-features-asm.csv
31472 10 of 10868 files processed.
31472 20 of 10868 files processed.
31472 30 of 10868 files processed.
31472 40 of 10868 files processed.
31472 50 of 10868 files processed.
31472 60 of 10868 files processed.
31472 70 of 10868 files processed.
31472 80 of 10868 files processed.
31472 90 of 10868 files processed.
31472 100 of 10868 files processed.
31472 110 of 10868 files processed.
31472 120 of 10868 files processed.
31472 130 of 10868 files processed.
31472 140 of 10868 files processed.
31472 150 of 10868 files processed.
31472 160 of 10868 files processed.
31472 170 of 10868 files processed.
```

In [4]:

```
result_byte_size = pd.read_csv("data/result_with_size.csv",index_col=0)
print("result_byte_size",result_byte_size.shape)

asm_feat_size = pd.read_csv("data/asm_feat_size.csv",index_col=0)
print("asm_feat_size",asm_feat_size.shape)

byte_bigram = scipy.sparse.load_npz("data/byte_bigram.npz").astype('float64')
print("byte_bigram",byte_bigram.data.shape)

asm_bigram = scipy.sparse.load_npz("data/asm_bigram.npz").astype('float64')
print("asm_bigram",asm_bigram.data.shape)

asm_trigram = scipy.sparse.load_npz("data/asm_trigram.npz").astype('float64')
print("asm_trigram",asm_trigram.data.shape)

byte_img = pd.read_csv("data/image-features-byte.csv",index_col=0).astype('float64')
print("byte_img",byte_img.shape)

asm_img = pd.read_csv("data/image-features-asm.csv",index_col=0).astype('float64')
print("asm_img",asm_img.shape)
```

```
result_byte_size (10868, 260)
asm_feat_size (10868, 54)
byte_bigram (454204072,)
asm_bigram (172367,)
asm_trigram (20196,)
byte_img (10868, 1000)
asm_img (10868, 1000)
```

In [5]:

```
merge_result = (pd.merge(result_byte_size,asm_feat_size,on='ID', how='left'))
merge_result
```

Out[5]:

	ID	0	1	2	3	4	5	6	7	8	...
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...
...	...	...	...	...	...	...	...	...	...	...	...
10863	loIP1tiwELF9YNZQjSUO	5268	1177	1072	1222	1238	1159	1143	1126	1149	...
10864	LOP6HaJKXpkic5dyuVnT	3032	298	248	293	274	213	203	222	257	...
10865	LOqA6FX02GWguYrI1Zbe	5671	221	270	323	313	155	248	147	261	...
10866	LoWgaidpb2IUM5ACcSGO	3637	437	453	506	511	390	431	407	405	...
10867	lS0IVqXeJrN6Dzi9Pap1	3534	373	385	432	495	399	393	373	399	...

10868 rows × 313 columns

In [6]:

```
y = merge_result['Class_y']
new = merge_result.drop(['ID','rtn','.BSS:','.CODE','Class_y'], axis=1)
```

Logistic Rregression on (merged asm-byte, byte bigram, asm bigram, and asm trigram)

In [43]:

```
X = scipy.sparse.hstack((new, byte_bigram, asm_bigram, asm_trigram)).tocsr()
```

In [44]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_si
```

In [34]:

```
from sklearn.feature_selection import SelectKBest, chi2
sk = SelectKBest(chi2, k=40000)
byte_bigram = sk.fit_transform(byte_bigram, y)
print("After reduction:",byte_bigram.shape)
byte_bigram = scipy.sparse.csr_matrix(byte_bigram)
```

In [36]:

```
X_train = scipy.sparse.save_npz("data/X_train.npz",X_train)
X_test = scipy.sparse.save_npz("data/X_test.npz",X_test)
X_cv = scipy.sparse.save_npz("data/X_cv.npz",X_cv)
```

In [46]:

```
X_train = scipy.sparse.load_npz("data/X_train.npz")
X_test = scipy.sparse.load_npz("data/X_test.npz")
X_cv = scipy.sparse.load_npz("data/X_cv.npz")
```

In [47]:

```
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)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',(log_loss(y_train, predict_y, labels=logisticR.classes_, e
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-
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 1.9005470863486864
log_loss for c =  0.0001 is 1.9005305305150424
log_loss for c =  0.001 is 1.9005703762752528
log_loss for c =  0.01 is 1.9005459326431722
log_loss for c =  0.1 is 1.9005172337401448
log_loss for c =  1 is 1.9005125470665105
log_loss for c =  10 is 1.9005120857526603
log_loss for c =  100 is 1.9005120359496557
log_loss for c =  1000 is 1.9005120317329027
```

RandomForest model with hyperparameter tuning (RandomSearchCV)

In [7]:

```
X = scipy.sparse.hstack((new, byte_img, asm_bigram, asm_img)).tocsr()
```

In [8]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_si
```

In [9]:

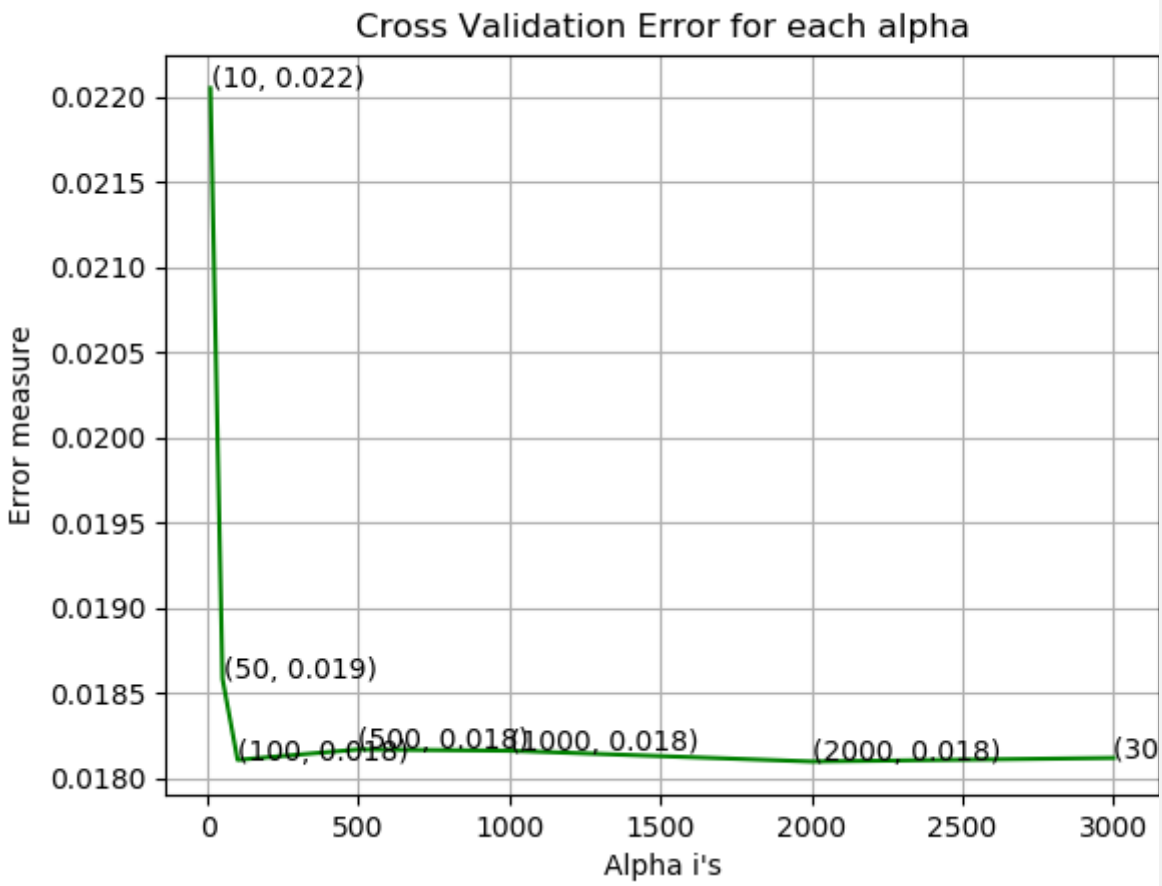
```
y_train = pd.DataFrame(y_train)
```

In [41]:

```
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,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_
```

```
log_loss for c =  10 is 0.02204840119331876
log_loss for c =  50 is 0.018584886713897405
log_loss for c =  100 is 0.018109319561570775
log_loss for c =  500 is 0.018170315693775654
log_loss for c =  1000 is 0.018163033037145795
log_loss for c =  2000 is 0.018099710012387877
log_loss for c =  3000 is 0.018120646091256513
```

Figure 2




```
For values of best alpha =  2000 The train log loss is: 0.011152302916362924
For values of best alpha =  2000 The cross validation log loss is: 0.0180997
10012387877
For values of best alpha =  2000 The test log loss is: 0.022388535590989717
```

Training Xgboost model with hyperparameter tuning (RandomSearchCV)

In [42]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15],
     'n_estimators':[100,200,250,300,500,700,1000,2000],
     'max_depth':[3,5,7,10],
    'colsample_bytree':[0.1,0.3,0.5]
}

random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10)
random_cfl.fit(X_train, y_train)
print (random_cfl.best_params_)
```

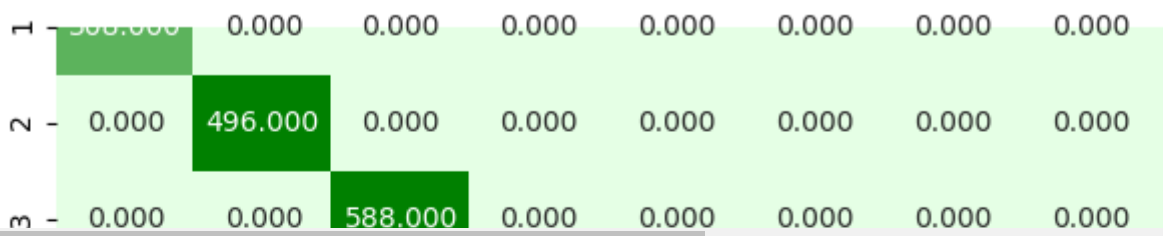
```
0.3, score=0.999, total= 8.3min
[CV] n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree=
0.3
[CV]  n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree
=0.3, score=0.998, total=11.0min
[CV] n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree=
0.3
[CV]  n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree
=0.3, score=0.999, total=10.8min
[CV] n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree=
0.3
[CV]  n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree
=0.3, score=0.998, total=10.9min
[CV] n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree=
0.3
[CV]  n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree
=0.3, score=0.997, total=10.9min
[CV] n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree=
0.3
[CV]  n_estimators=250, max_depth=10, learning_rate=0.01, colsample_bytree
```

In [45]:

```
x_cfl=XGBClassifier(n_estimators=200,max_depth=3,learning_rate=0.05,colsample_bytree=0.3)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
train loss 0.0076463355272154065
train loss 0.007433272261757593
test loss 0.008846957483662843
Number of misclassified points  0.045998160073597055
---------------------------------------------------- Confusion matrix ------
-----------------------------------------
```

Figure 3



Model Performance summary

1. ML models on byte features

In [40]:

```
from prettytable import PrettyTable

x = PrettyTable(["Model", "Train loss", "Cv loss", "Test loss","missclassifies points"])

x.add_row(["Rnadom model","-", 2.49,2.46,89.28])
x.add_row(["KNN", 0.06,0.19,0.21,4.18])
x.add_row(["Logistic regrssion", 0.86,0.88,0.87,28.2])
x.add_row(["Random forest", 0.02,0.07,0.07,1.93])
x.add_row(["XGBoost", 0.02,0.06,0.06,1.42])

print(x.get_string(title="Model results"))
```

```
+--------------------+------------+---------+-----------+-------------------
----+
|       Model        | Train loss | Cv loss | Test loss | missclassifies poi
nts |
+--------------------+------------+---------+-----------+-------------------
----+
|    Rnadom model    |     -      |   2.49  |    2.46   |         89.28
|
|        KNN         |    0.06    |   0.19  |    0.21   |          4.18
|
| Logistic regrssion |    0.86    |   0.88  |    0.87   |          28.2
|
|   Random forest    |    0.02    |   0.07  |    0.07   |          1.93
|
|      XGBoost       |    0.02    |   0.06  |    0.06   |          1.42
|
+--------------------+------------+---------+-----------+-------------------
----+
```

2. ML model on asm features

In [42]:

```
x = PrettyTable(["Model", "Train loss", "Cv loss", "Test loss","missclassifies points"])

x.add_row(["KNN", 0.02,0.09,0.099,1.655])
x.add_row(["Random forest", 0.01,0.03,0.03,0.64])
x.add_row(["XGBoost", 0.01,0.03,0.04,1.011])

print(x.get_string(title="Model results"))
```

```
+---------------+------------+---------+-----------+-----------------------+
|     Model     | Train loss | Cv loss | Test loss | missclassifies points |
+---------------+------------+---------+-----------+-----------------------+
|      KNN      |    0.02    |   0.09  |   0.099   |         1.655         |
| Random forest |    0.01    |   0.03  |    0.03   |          0.64         |
|    XGBoost    |    0.01    |   0.03  |    0.04   |         1.011         |
+---------------+------------+---------+-----------+-----------------------+
```

3. ASM + BYTE features

In [41]:

```
x = PrettyTable(["Model", "Train loss", "Cv loss", "Test loss","missclassifies points"])

x.add_row(["Random Forest",0.01,0.03, 0.04,0.51])
x.add_row(["XGBoost", 0.01,0.03,0.03,0.50])

print(x.get_string(title="Model results"))
```

```
+---------------+------------+---------+-----------+-----------------------+
|     Model     | Train loss | Cv loss | Test loss | missclassifies points |
+---------------+------------+---------+-----------+-----------------------+
| Random Forest |    0.01    |   0.03  |    0.04   |          0.51         |
|    XGBoost    |    0.01    |   0.03  |    0.03   |          0.5          |
+---------------+------------+---------+-----------+-----------------------+
```

4. ASM + BYTE (unigram, file_size, bigrams byte, bygram asm, asm_trigram, byte image, asm image)

In [12]:

```
from prettytable import PrettyTable
x = PrettyTable(["Model", "Train loss", "Cv loss", "Test loss","missclassifies points"])

x.add_row(["Logistic regression",0.01,0.03, 0.04,0.51])
x.add_row(["Random Forest",0.01,0.03, 0.02,0.49])
x.add_row(["XGBoost", 0.007,0.007,0.008,0.09])

print(x.get_string(title="Model results"))
```

```
+---------------------+------------+---------+-----------+------------------
-----+
|        Model        | Train loss | Cv loss | Test loss | missclassifies po
ints |
+---------------------+------------+---------+-----------+------------------
-----+
| Logistic regression |    0.01    |   0.03  |    0.04   |          0.51
|
|    Random Forest    |    0.01    |   0.03  |    0.02   |          0.49
|
|       XGBoost       |   0.007    |  0.007  |   0.008   |          0.09
|
+---------------------+------------+---------+-----------+------------------
-----+
```

The Loss got reduced to 0.008

--- Done