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

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

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

2.1.2. Example Data Point

.asm file

```
.text:00401000                                  assume es:nothing, ss:nothing, ds:_data,    fs:nothing, gs:
nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24    08                           lea     eax, [esp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                               mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                        call    ??0exception@std@@QAE@ABQBD@Z ; std::exce
ption::exception(char const * const &)
.text:0040100D C7 06 08    BB 42 00                    mov     dword ptr [esi],    offset off_42BB08
.text:00401013 8B C6                               mov     eax, esi
.text:00401015 5E                               pop     esi
.text:00401016 C2 04 00                               retn    4
.text:00401016                          ; ------------------------------------------------------------------
----------
.text:00401019 CC CC CC    CC CC CC CC                    align 10h
.text:00401020 C7 01 08    BB 42 00                    mov     dword ptr [ecx],    offset off_42BB08
.text:00401026 E9 26 1C    00 00                        jmp     sub_402C51
.text:00401026                          ; ------------------------------------------------------------------
----------
.text:0040102B CC CC CC    CC CC                        align 10h
.text:00401030 56                               push    esi
.text:00401031 8B F1                               mov     esi, ecx
.text:00401033 C7 06 08    BB 42 00                    mov     dword ptr [esi],    offset off_42BB08
.text:00401039 E8 13 1C    00 00                        call    sub_402C51
.text:0040103E F6 44 24    08 01                        test    byte ptr    [esp+8], 1
.text:00401043 74 09                               jz      short loc_40104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E    00 00                        call    ??3@YAXPAX@Z    ; operator delete(void *)
.text:0040104B 83 C4 04                               add     esp, 4
.text:0040104E
.text:0040104E                          loc_40104E:                  ; CODE XREF: .text:00401043j
.text:0040104E 8B C6                               mov     eax, esi
.text:00401050 5E                               pop     esi
.text:00401051 C2 04 00                               retn    4
```

```
.text:00401051                          ; ---------------------------------------------------------------------------
```

.bytes file

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

3. Exploratory Data Analysis

In [1]:

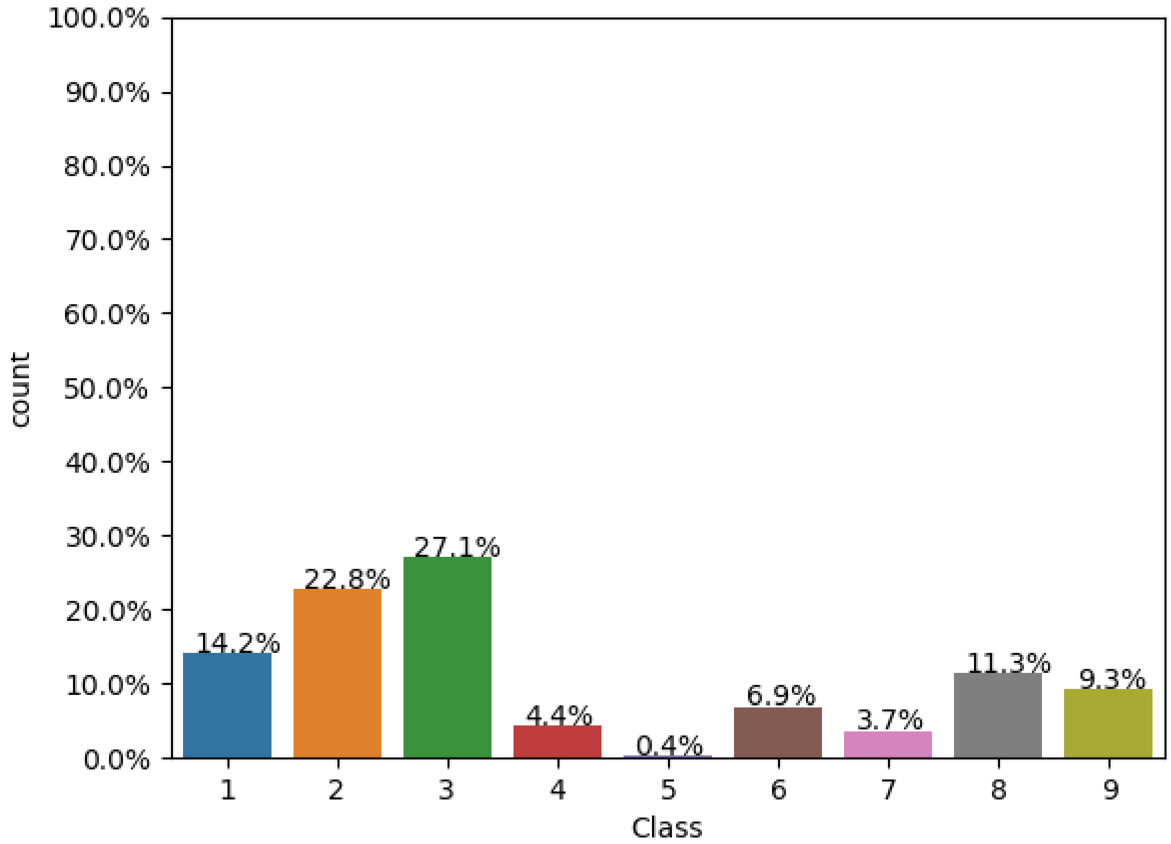
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import 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
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
#separating byte files and asm files

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

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

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

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

3.1. Distribution of malware classes in whole data set

In [3]:
```
Y=pd.read_csv("trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

#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()
```

Figure 1



In [4]:
```
Y=pd.read_csv("trainLabels.csv")
```

3.2. Feature extraction

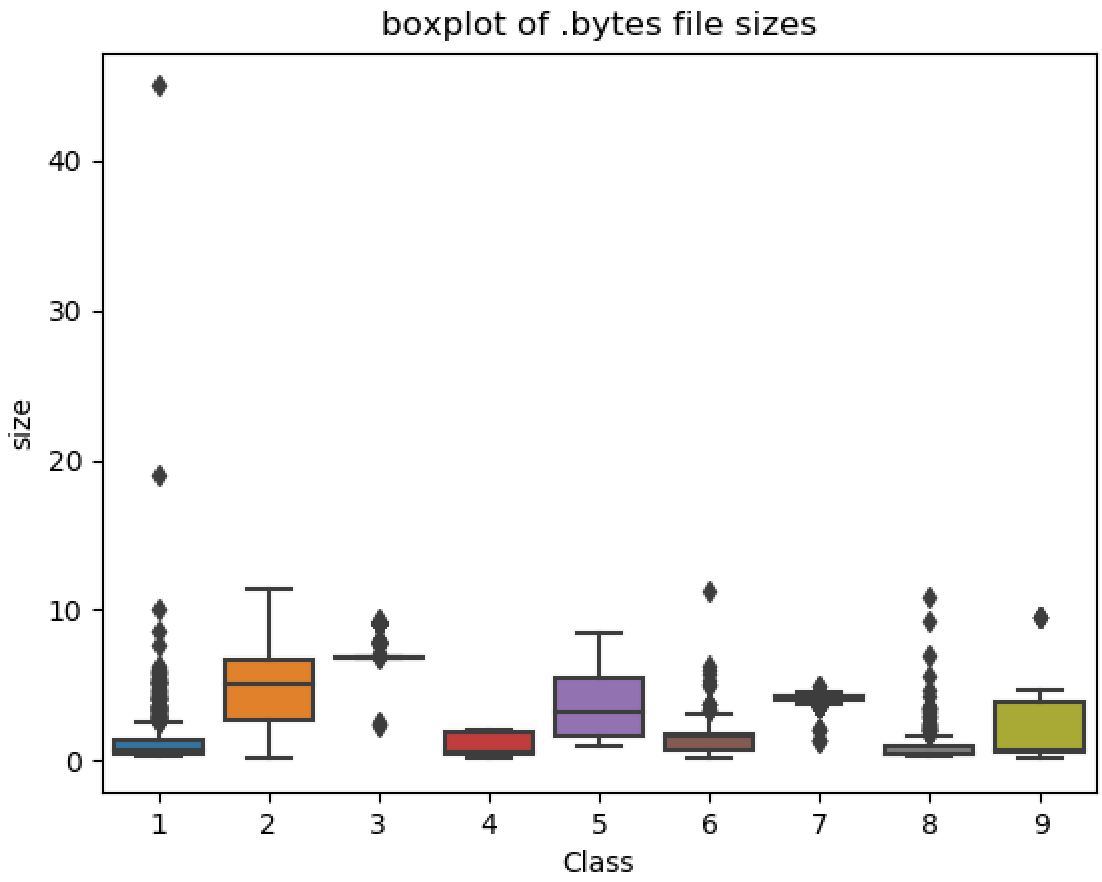
3.2.1 File size of byte files as a feature

In [5]:
```
#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_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

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

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

In [5]:
```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



3.2.3 feature extraction from byte files

DO not re-run this cell

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

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

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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,2
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 [6]:

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

Out[6]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8	f9	fa	fb	fc	fd	fe	ff
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687	3101	3211	3097	2758	3099	2759	5753
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536	439	281	302	7639	518	17001	54902

2 rows × 258 columns

In [7]:

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

Out[7]:

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

In [8]:
```
byte_features_with_size = byte_features.merge(data_size_byte, on='ID')
byte_features_with_size.to_csv("result_with_size.csv")
byte_features_with_size.head(2)
```

Out[8]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	Class
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759	5753	1824	9
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	17001	54902	8588	2

2 rows × 260 columns

In [9]:
```
# 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 [10]:
```
result.head(2)
```

Out[10]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	f
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.01356	0.013107	0.01363
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.00192	0.001147	0.00132

2 rows × 260 columns

In [11]:
```
data_y = result['Class']
result.head()
```

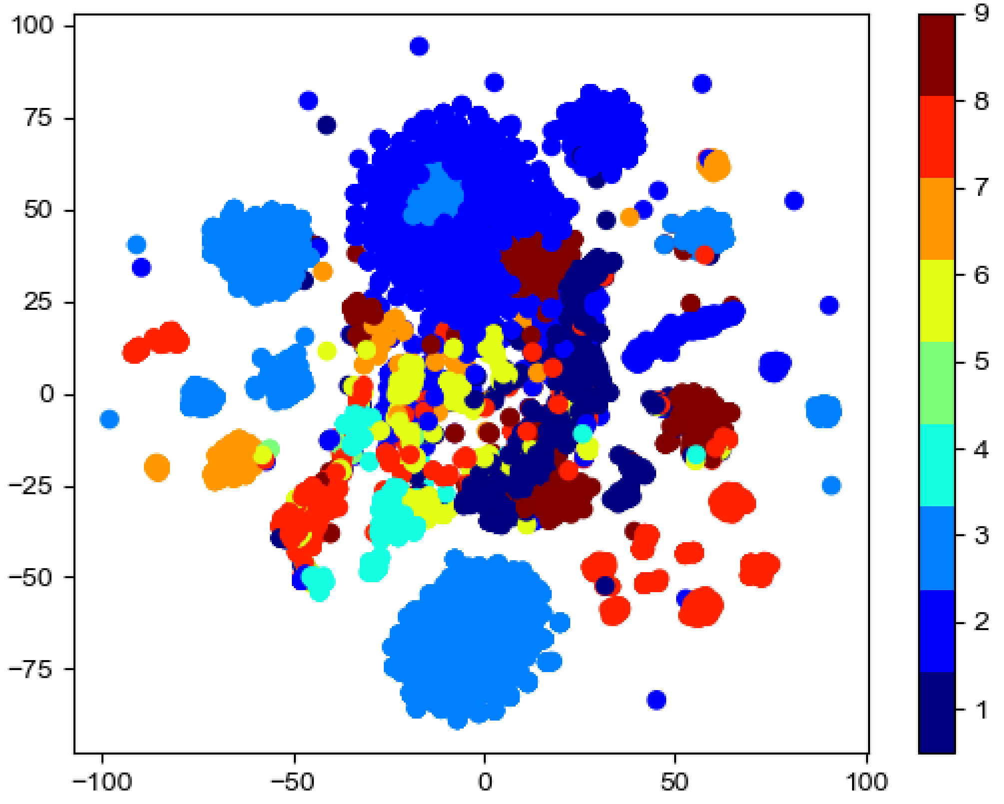
Out[11]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.013560	0.013107	0.013
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.001920	0.001147	0.001
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.009804	0.011777	0.012
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.002121	0.001886	0.002
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.001530	0.000853	0.001

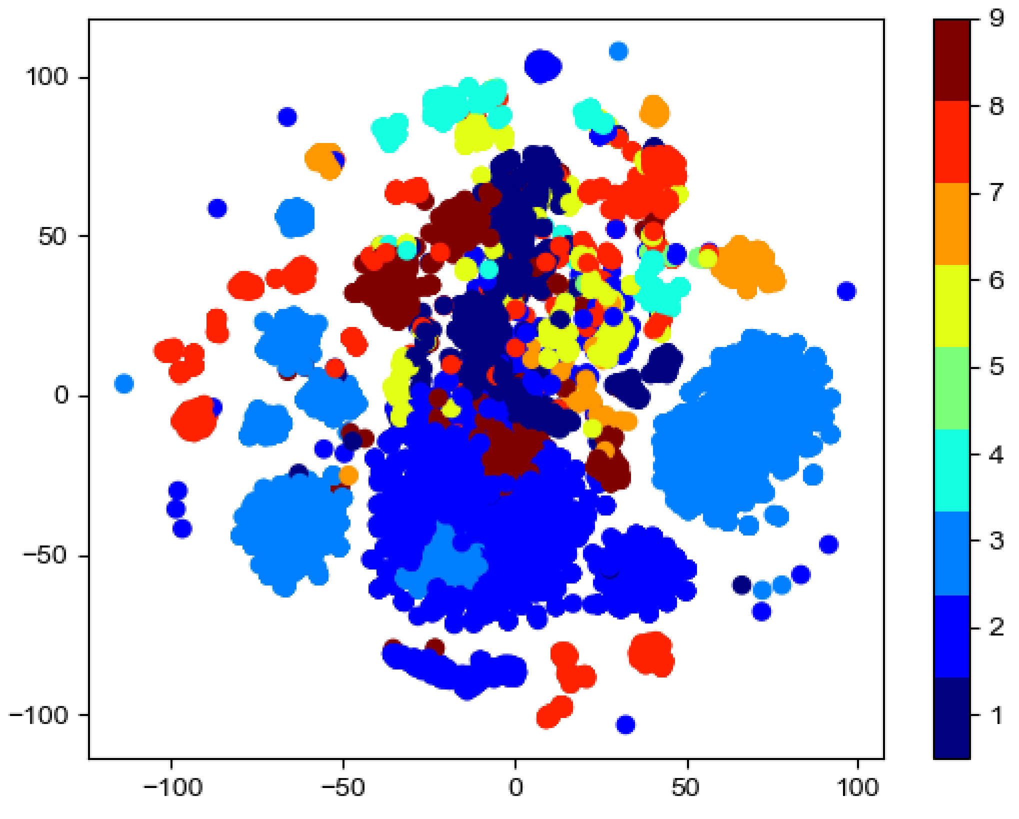
5 rows × 260 columns

3.2.4 Multivariate Analysis

In [18]:
```
#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 [19]:
```
#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()
```



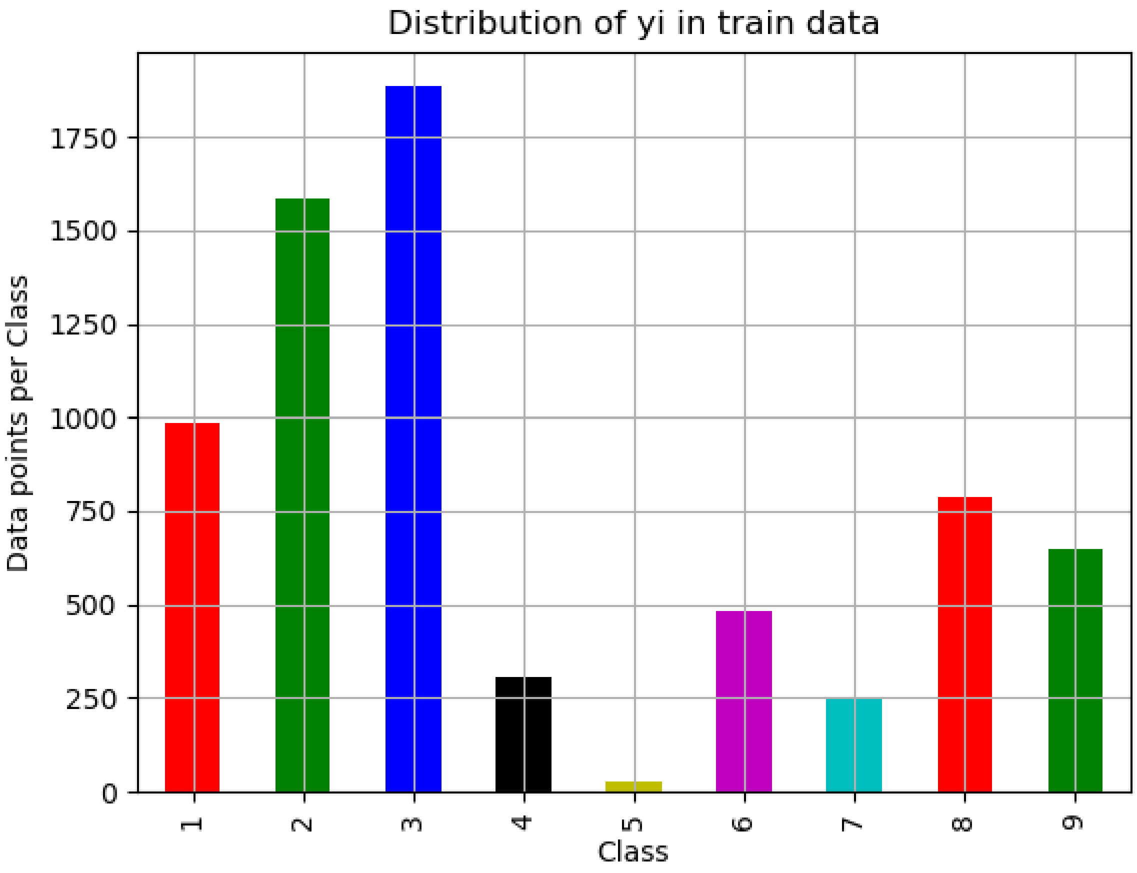
Train Test split

In [12]:
```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), data_y,stratify=data_y,test_size
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [str
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

In [13]:
```
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 [14]:
```
plt.close()
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = y_train.value_counts().sort_index()#sortlevel()
test_class_distribution = y_test.value_counts().sort_index()#.sortlevel()
cv_class_distribution = y_cv.value_counts().sort_index()#.sortlevel()

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

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




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

In [12]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # 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 diamensional array
    # 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))
```

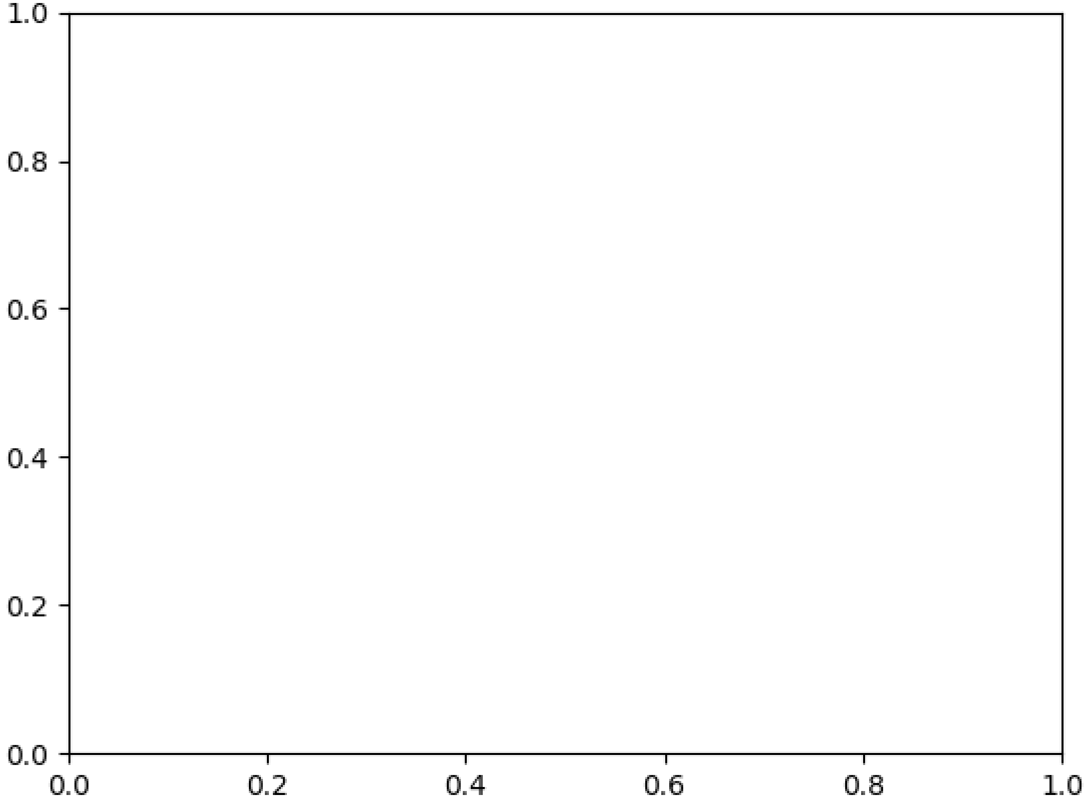
4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

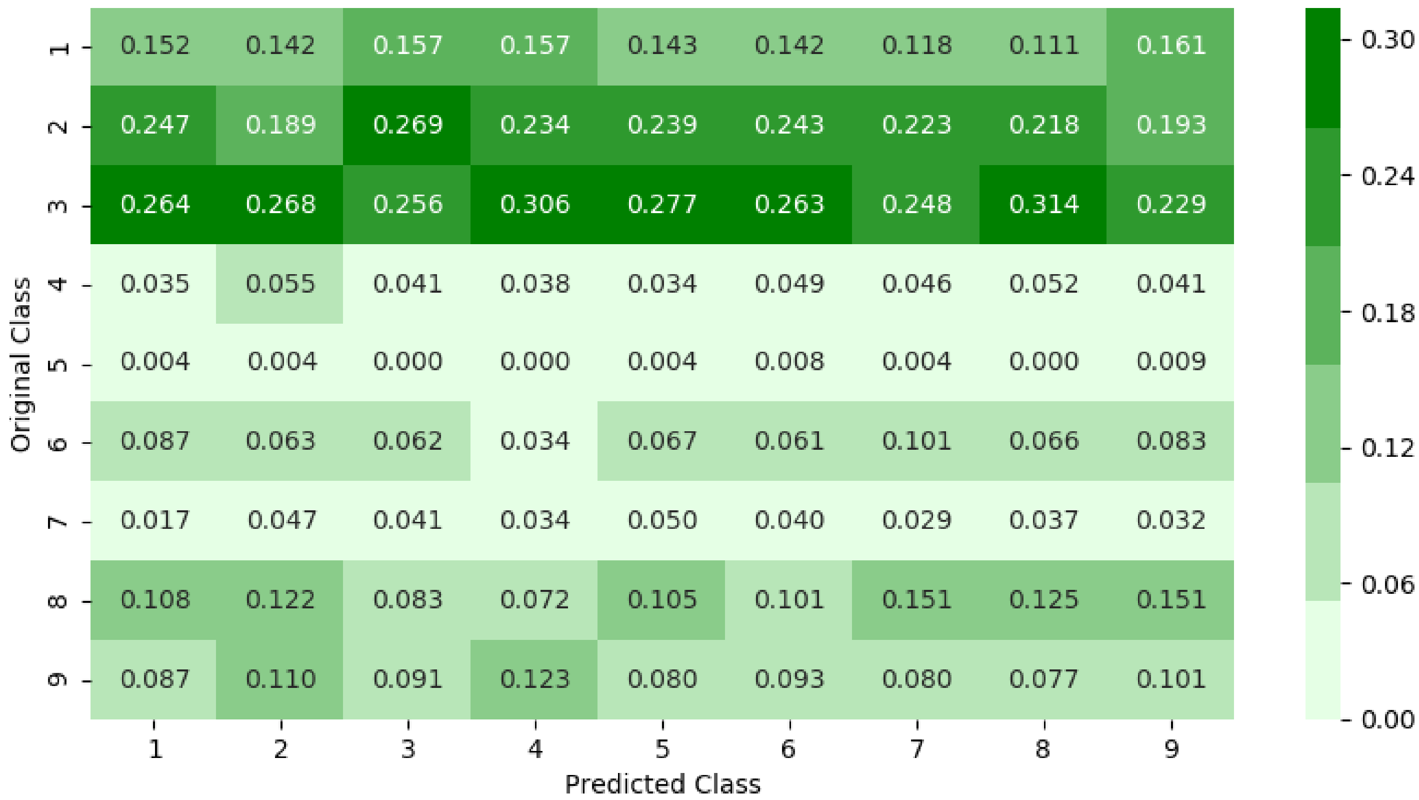
4.1.1. Random Model

In [20]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

```
Log loss on Cross Validation Data using Random Model 2.493476250570028
Log loss on Test Data using Random Model 2.4604734259698806
Number of misclassified points  89.28242870285189
-------------------------------------------------- Confusion matrix --------------------------------------------------
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


```
-------------------------------------------------- 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 [ ]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighbo
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

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

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

```
best_alpha = np.argmin(cv_log_error_array)

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

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

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

4.1.3. Logistic Regression

In [ ]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------

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-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()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
```

```
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

4.1.4. Random Forest Classifier

In [22]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-constructio
# --------------------------------

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

```
log_loss for c =  10 is 0.11157164384551466
log_loss for c =  50 is 0.09696429708050865
log_loss for c =  100 is 0.0950865275864462
log_loss for c =  500 is 0.09594983163463172
log_loss for c =  1000 is 0.09568041109203713
log_loss for c =  2000 is 0.09617303082764878
log_loss for c =  3000 is 0.09617435618600804
```

```
For values of best alpha =  100 The train log loss is: 0.028625251450565735
```

```
For values of best alpha =  100 The cross validation log loss is: 0.0950865275864462
For values of best alpha =  100 The test log loss is: 0.08665332884582864
Number of misclassified points  2.2539098436062557
-------------------------------------------------- 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]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGB
# -------------------------
# 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_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# 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, verbose=True, xgb_model=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

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

```
log_loss for c =  10 is 0.21141965982131736
log_loss for c =  50 is 0.11973015361207139
log_loss for c =  100 is 0.09604572803993582
log_loss for c =  500 is 0.09309776229406479
log_loss for c =  1000 is 0.0926392195921875
log_loss for c =  2000 is 0.09337221416872009
```

```
For values of best alpha =  1000 The train log loss is: 0.022925073792437377
For values of best alpha =  1000 The cross validation log loss is: 0.0926392195921875
For values of best alpha =  1000 The test log loss is: 0.06532751432198568
Number of misclassified points  1.3339466421343145
-------------------------------------------------- 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-codes-python/
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  8.1min remaining:   54.1s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  9.0min finished
```

Out[24]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [25]:
```
print (random_cfl1.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 1000, 'max_depth': 10, 'learning_rate': 0.2, 'colsample_bytree': 0.3}
```

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/python/python_api.html?#xgboost.XGB
# -------------------------
# 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_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# 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, verbose=True, xgb_model=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, max_depth=3)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

```
train loss 0.023071593469527234
cv loss 0.0922959108365791
test loss 0.0655375289593565
```

4.2 Modeling with .asm files

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


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

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

4.2.1 Feature extraction from asm files

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

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

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

In [ ]:

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

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

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BS
    #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', 'dec', 'add','imul', 'xchg', 'or'
    #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:','.rsrc:','.tls:','.reloc:','.BS
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or'
    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:','.rsrc:','.tls:','.reloc:','.BS
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or'
    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:','.rsrc:','.tls:','.reloc:','.BS
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or'
    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:','.rsrc:','.tls:','.reloc:','.BS
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or'
    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 [13]:

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

Out[13]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip	C
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83	0	17	48	29	
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12	0	14	0	20	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14	0	11	0	9	
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	0	8	0	6	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	0	11	0	11	

5 rows × 53 columns

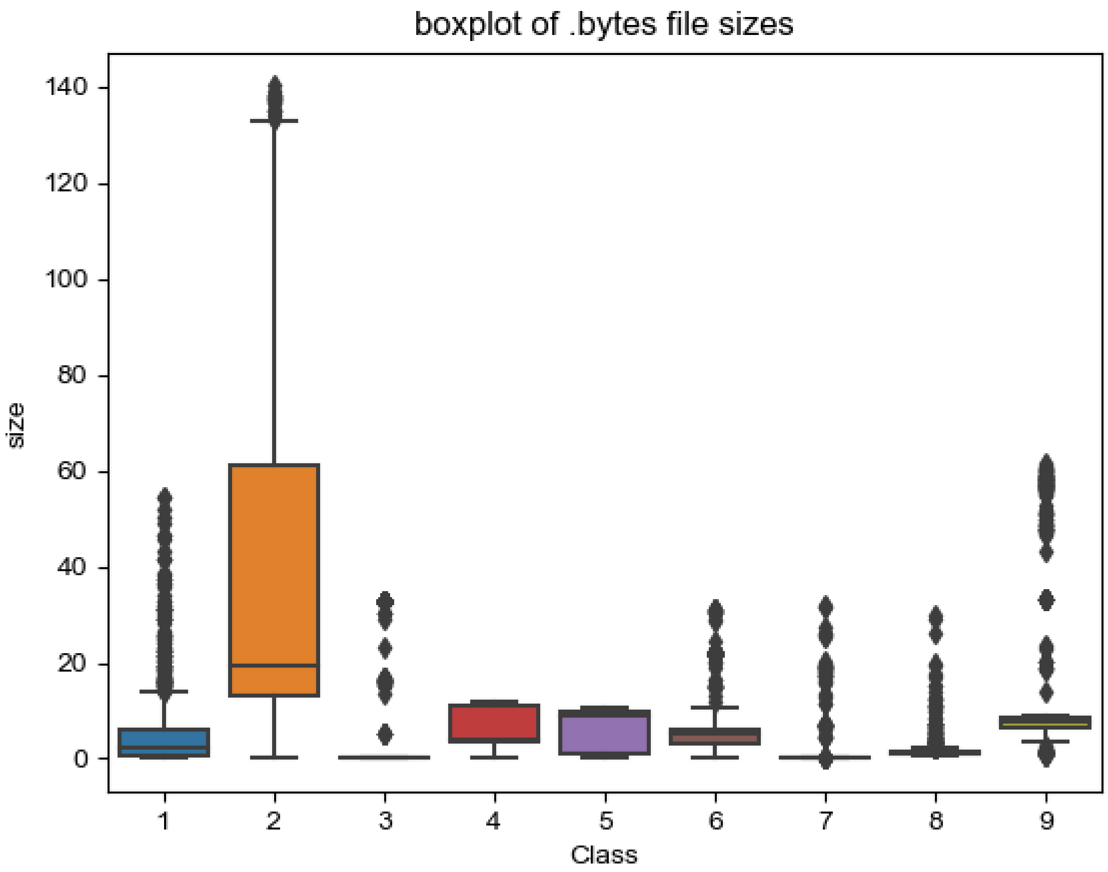
4.2.1.1 Files sizes of each .asm file

In [14]:
```
#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_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

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

4.2.1.2 Distribution of .asm file sizes

In [25]:
```
#boxplot of asm files
ax = sns.boxplot(x="Class", y="size", data=asm_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



In [17]:
```python
# 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[17]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	48	29	1
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	0	20	1
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	0	9	1
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	0	6	1
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	0	11	1

5 rows × 54 columns

In [18]:
```python
# we normalize the data each column
result_asm = normalize(result_asm)
result_asm.head()
```

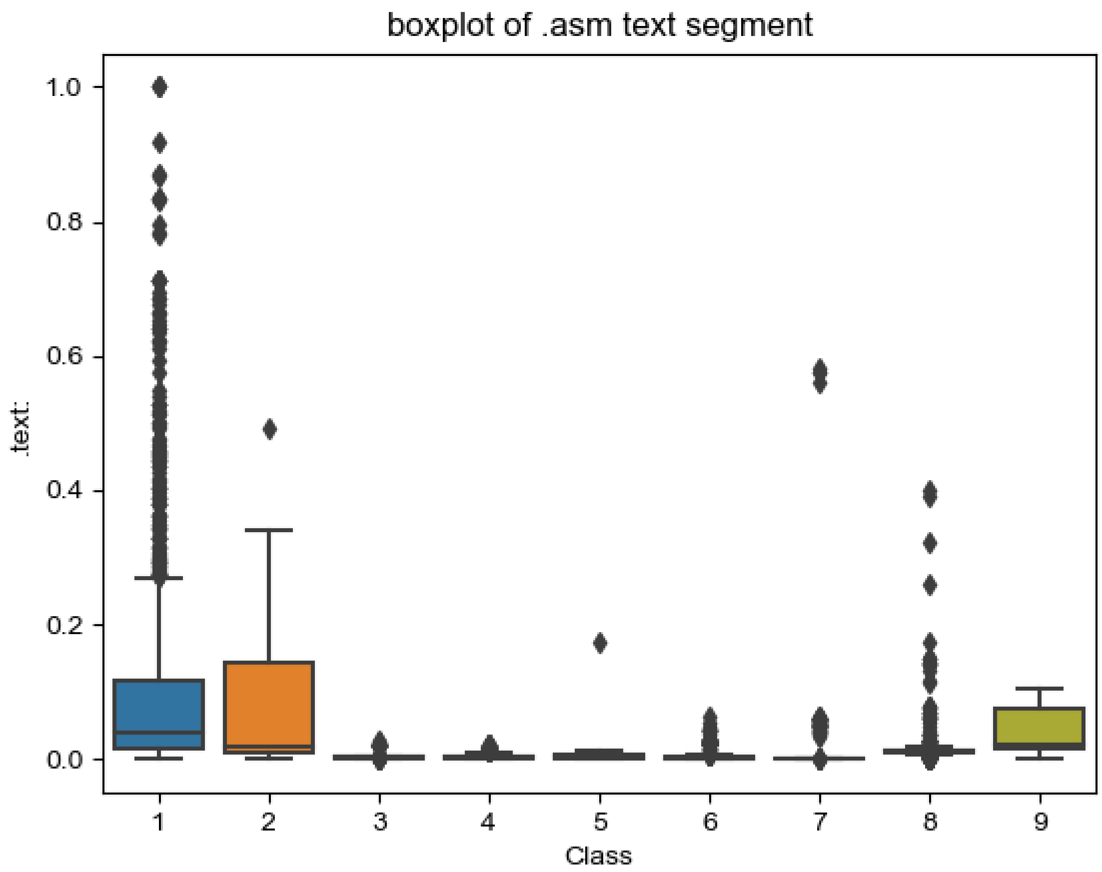
Out[18]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.C
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.C
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.C
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.C
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.C

5 rows × 54 columns

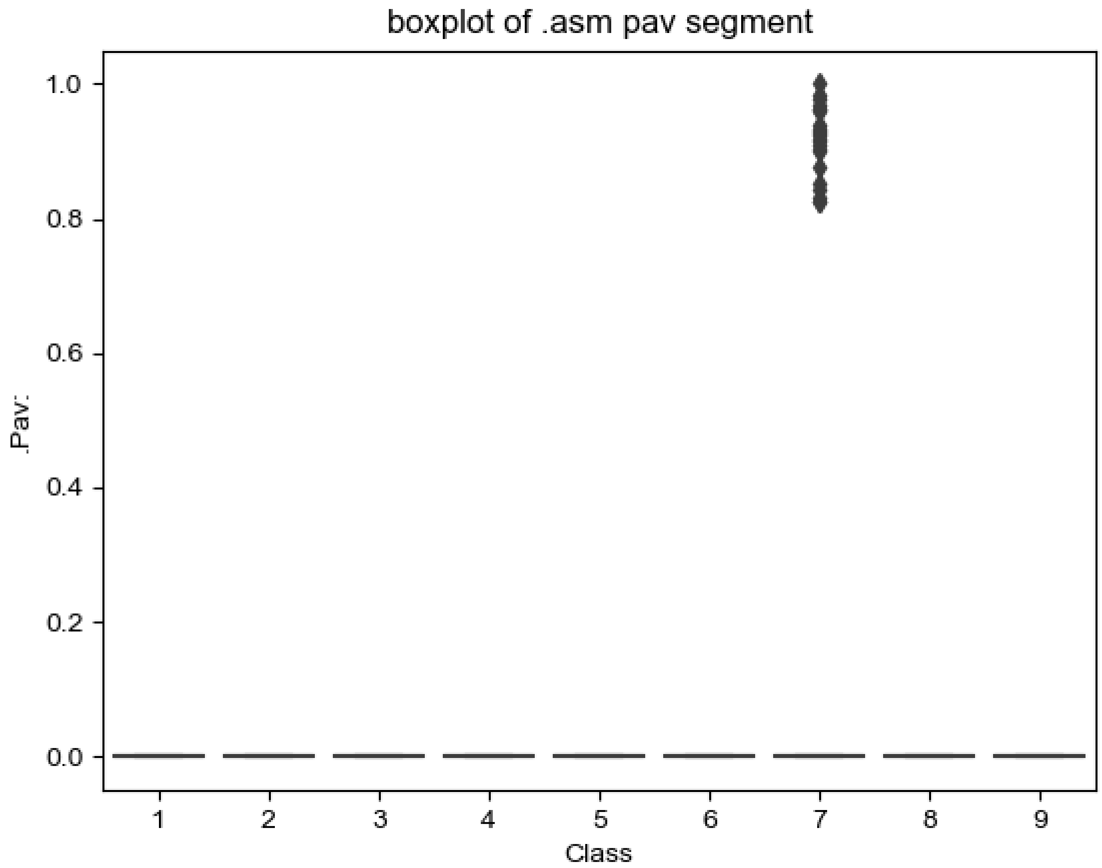
4.2.2 Univariate analysis on asm file features

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

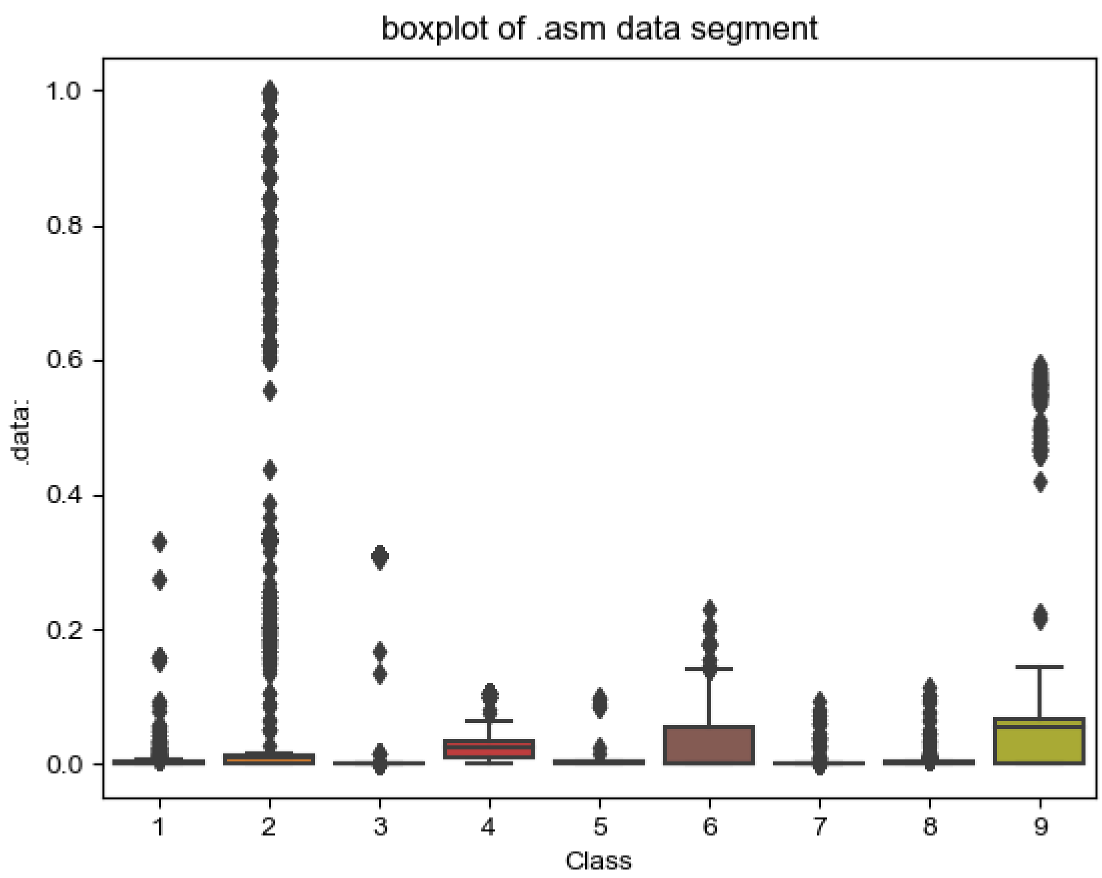



```
The plot is between Text and class
Class 1,2 and 9 can be easly separated
```

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

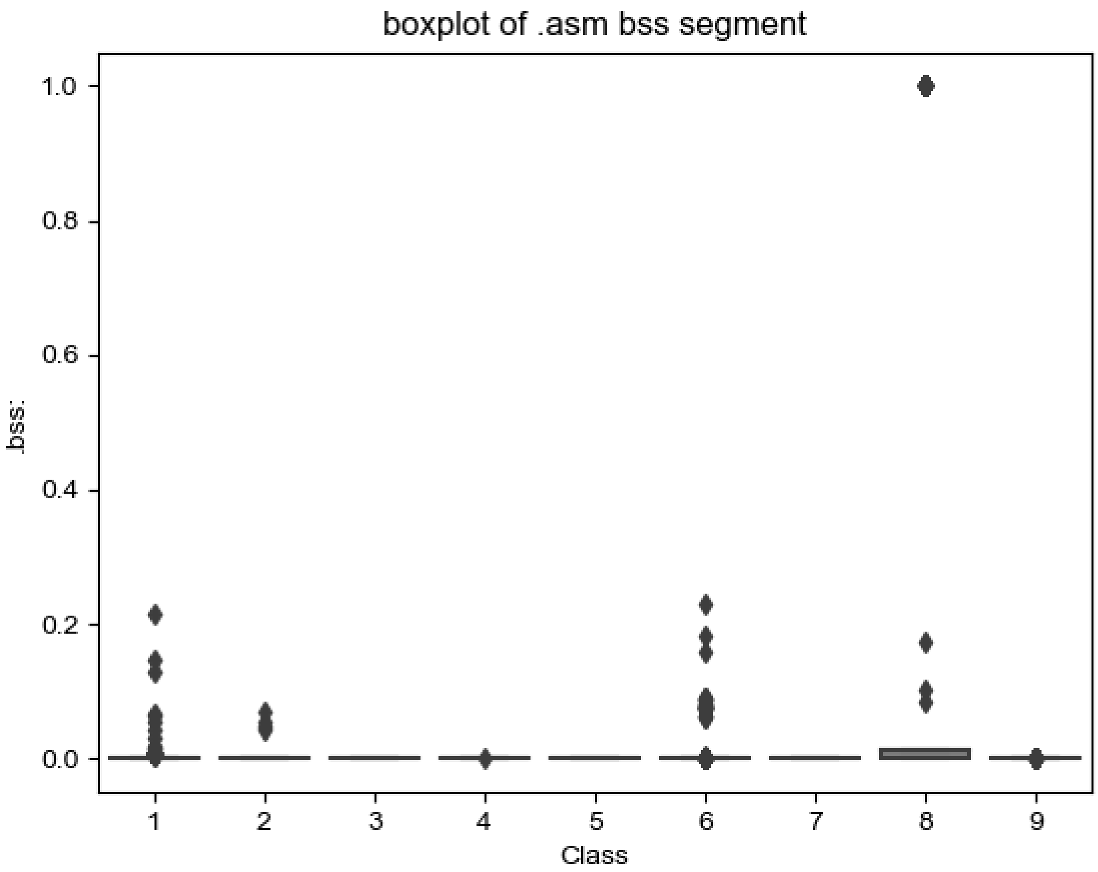


In [30]:
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



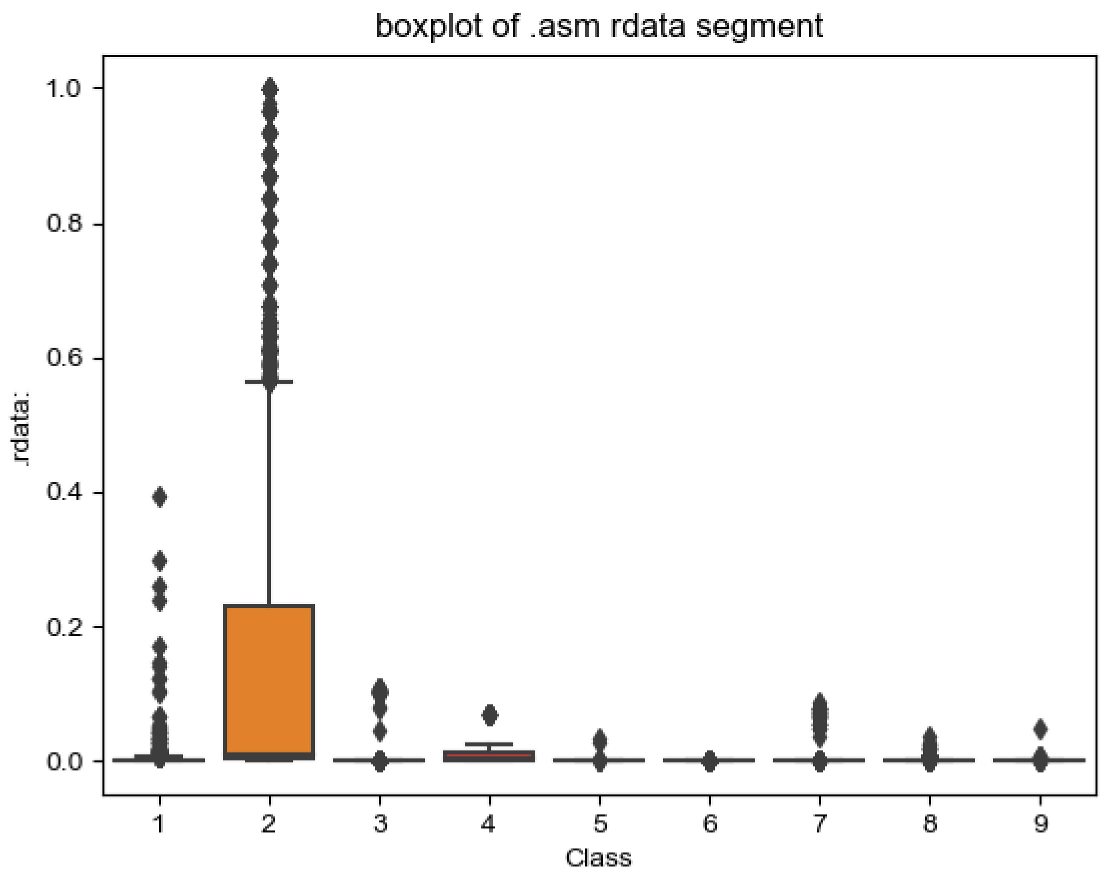
The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

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



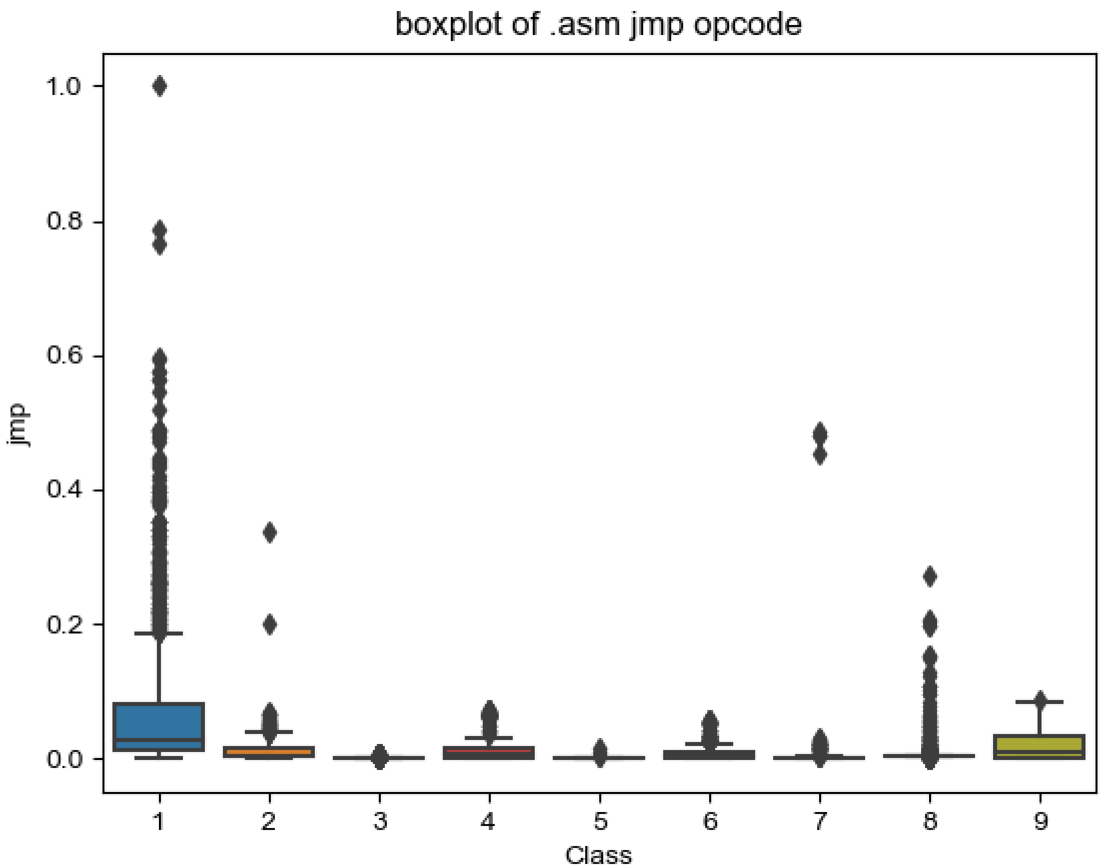
plot between bss segment and class label
very less number of files are having bss segment

In [32]:
```
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
```

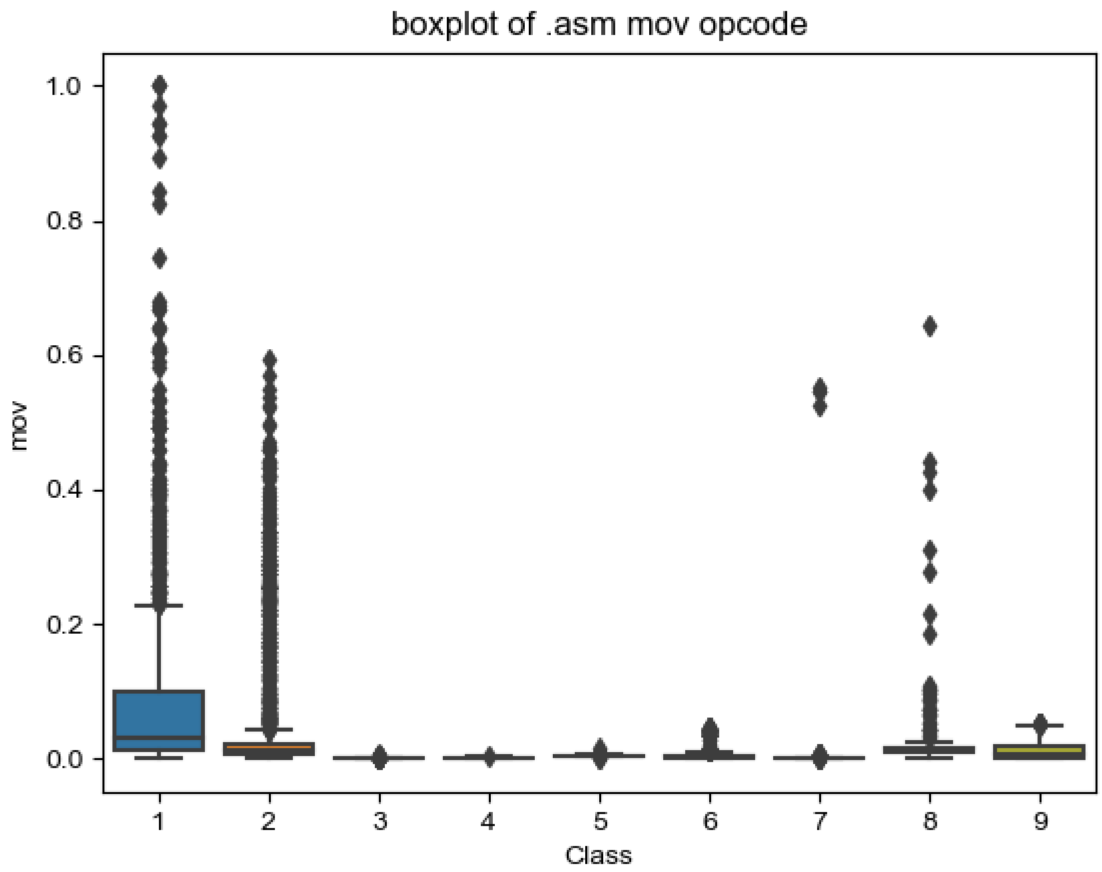
In [33]:
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



plot between jmp and Class label

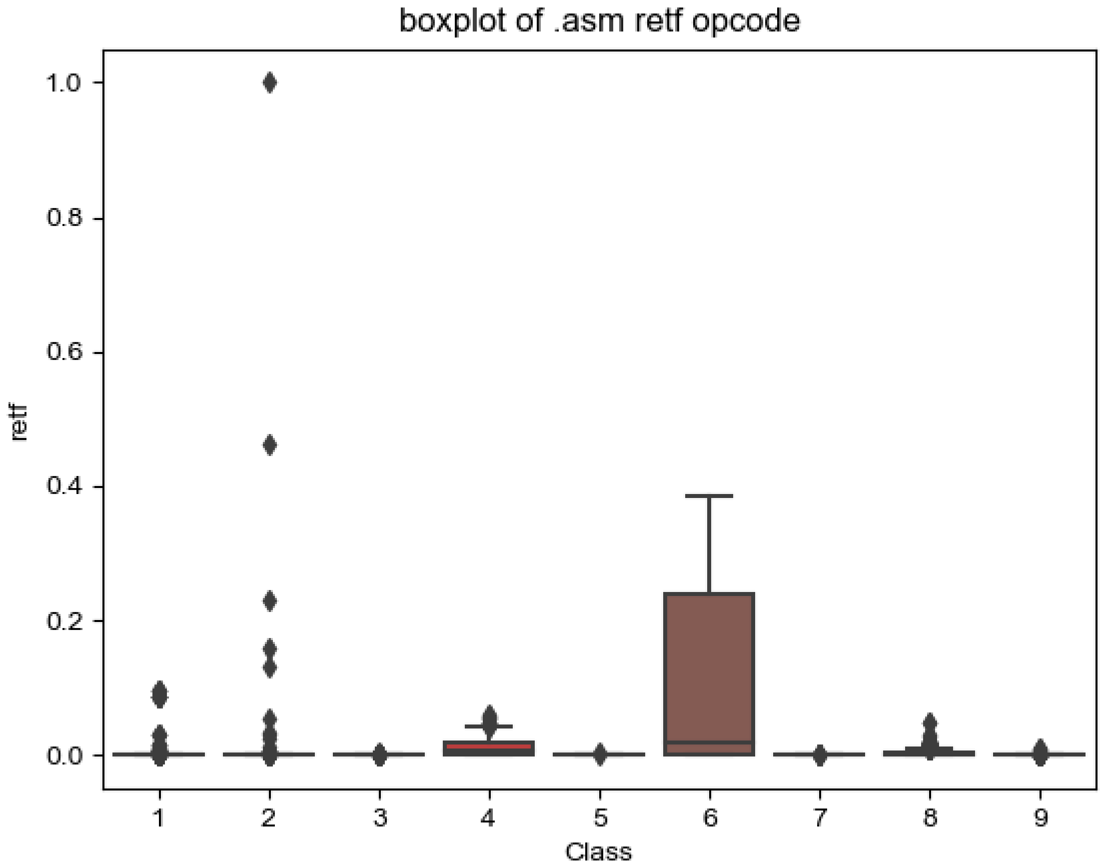
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [34]:
```
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 [35]:
```
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 [36]:
```
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 [37]:
```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic-neighbourhood-embeddin

#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 [38]:
```
# by univariate analysis on the .asm file features we are getting very negligible information from
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing those features
# the plot looks very messy

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

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 [15]:
```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

In [16]:
```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y_train_asm,test_size=0
```

In [21]:
```
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 [46]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighbo
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-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_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 0.05779020348734631 4
log_loss for k =  3 is 0.07108368472140039
log_loss for k =  5 is 0.08171015009225038
log_loss for k =  7 is 0.09100171161638985
log_loss for k =  9 is 0.10049193837065191
log_loss for k =  11 is 0.10781828649465963
log_loss for k =  13 is 0.11483230322117895
log_loss for k =  15 is 0.1211717897157547
log_loss for k =  17 is 0.12763903071616556
log_loss for k =  19 is 0.13346283771702003
```

```
log loss for train data 0.027303568672216888
log loss for cv data 0.057790203487346314
log loss for test data 0.09769263073207214
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 [47]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------


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=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()
```

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

predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  1e-05 is 1.6055761460085234
log_loss for c =  0.0001 is 1.5639380687156217
log_loss for c =  0.001 is 1.3096195040641458
log_loss for c =  0.01 is 1.3378139497181403
log_loss for c =  0.1 is 1.1734216204810437
log_loss for c =  1 is 0.7463950117978878
log_loss for c =  10 is 0.4998489231570535
log_loss for c =  100 is 0.38840431656709146
log_loss for c =  1000 is 0.3104492364519796
```

```
log loss for train data 0.306549337696636
log loss for cv data 0.3104492364519796
log loss for test data 0.3208474965246118
Number of misclassified points  6.39374425022999
-------------------------------------------------- Confusion matrix --------------------------------------------------
```

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

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

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

4.4.3 Random Forest Classifier

In [48]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-constructio
# --------------------------------

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-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_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.027096791679419252
log_loss for c =  50 is 0.02218809258189358
log_loss for c =  100 is 0.021177540593753374
log_loss for c =  500 is 0.020077968877239758
log_loss for c =  1000 is 0.02013453896715885
log_loss for c =  2000 is 0.020197298776636112
log_loss for c =  3000 is 0.020197419050552674
```

```
log loss for train data 0.014349453873898214
log loss for cv data 0.020077968877239758
log loss for test data 0.048885457020057914
Number of misclassified points  1.0579576816927323
```

```
-------------------------------------------------- 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 [49]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGB
# -------------------------
# 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_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# 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, verbose=True, xgb_model=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

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-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_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(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_asm, predict_y)
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_test_asm, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.08783883338746723
log_loss for c =  50 is 0.032060859553591614
log_loss for c =  100 is 0.01942689020123503
log_loss for c =  500 is 0.017420812548639756
log_loss for c =  1000 is 0.016723487733909873
log_loss for c =  2000 is 0.01637432565426884
log_loss for c =  3000 is 0.01637857932744878
```

```
For values of best alpha =  2000 The train log loss is: 0.011485635369330848
For values of best alpha =  2000 The cross validation log loss is: 0.01637432565426884
For values of best alpha =  2000 The test log loss is: 0.03915498819124778
Number of misclassified points  0.5979760809567618
-------------------------------------------------- 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 [50]:
```
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:   17.1s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  2.8min remaining:   18.4s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  3.0min finished
```

Out[50]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [51]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.15, 'colsample_bytree': 0.5}
```

In [52]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGB
# -------------------------
# 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_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# 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, verbose=True, xgb_model=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5,max_depth=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))
```

```
train loss 0.012436477487619937
cv loss 0.016887325969990694
test loss 0.04305581477123653
```

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

4.5.1. Merging both asm and byte file features

In [22]:
```
result.head()
```

Out[22]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.013560	0.013107	0.01
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.001920	0.001147	0.00
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.009804	0.011777	0.01
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.002121	0.001886	0.00
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.001530	0.000853	0.00

5 rows × 260 columns

In [23]:
```
result_asm.head()
```

Out[23]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.0
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.0
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.0
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.0
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.0

5 rows × 54 columns

In [24]:
```
print(result.shape)
print(result_asm.shape)
```

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

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

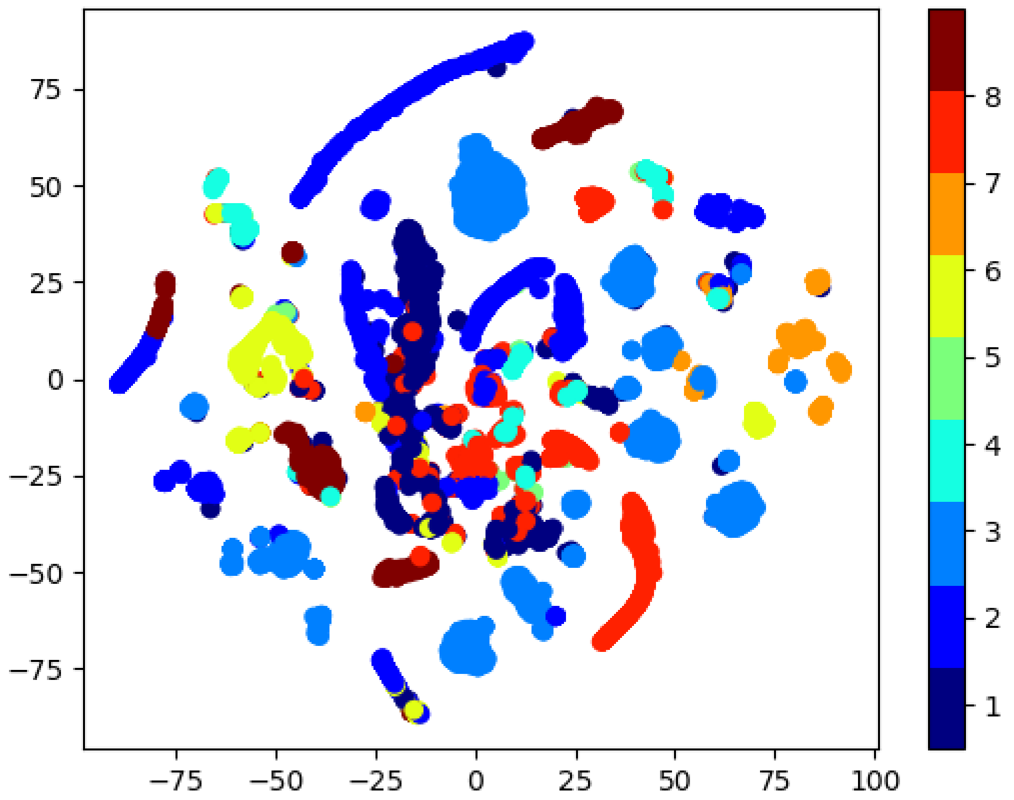
	0	1	2	3	4	5	6	7	8	9	...	:dword	edx	esi	eax	ebx	ecx	ed
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	4371	808	2290	1281	587	701	(
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	1446	260	1090	391	905	420	(
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	903	5	547	5	451	56	(
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	137	18	66	15	43	83	(
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	1220	18	1228	24	1546	107	(

5 rows × 306 columns

4.5.2. Multivariate Analysis on final fearures

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

Figure 2



4.5.3. Train and Test split

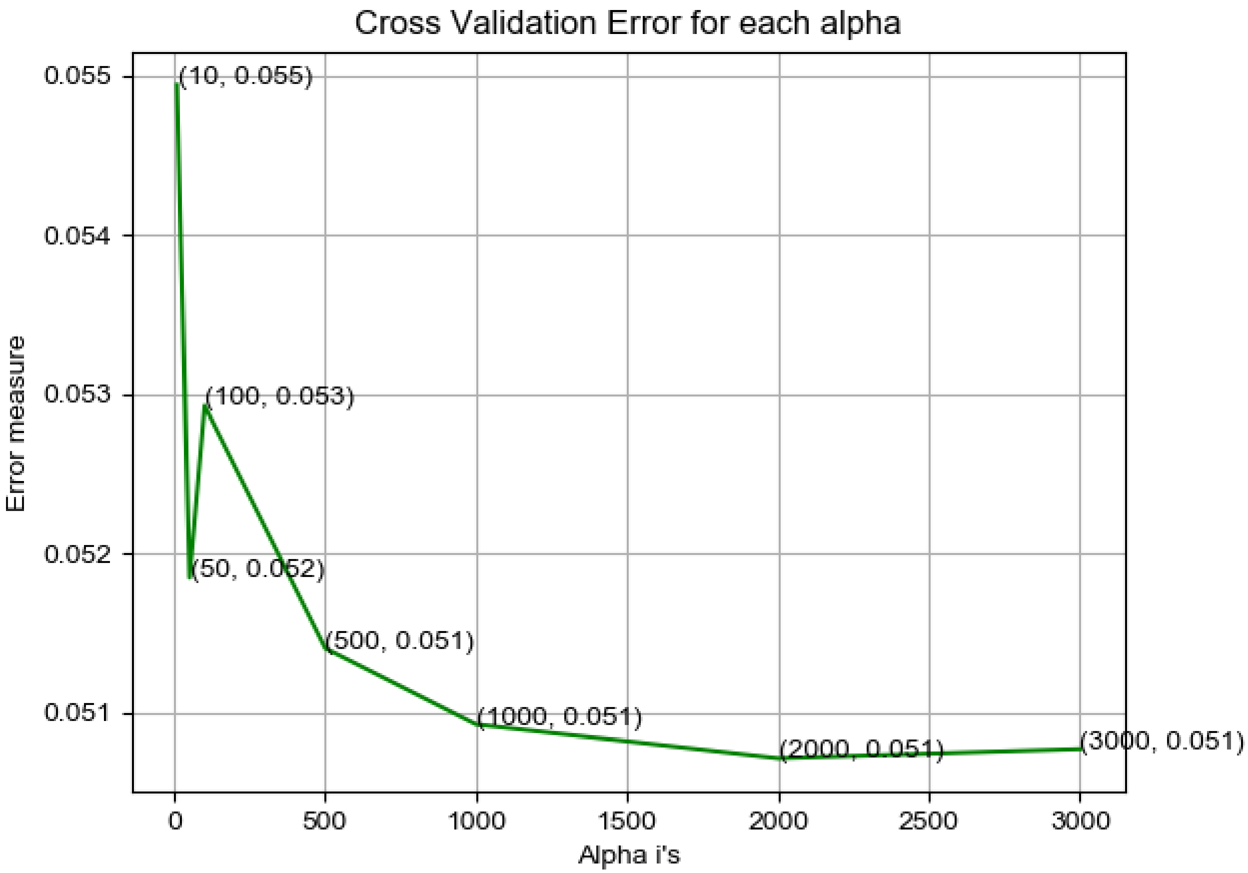
In [ ]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

4.5.4. Random Forest Classifier on final features

In [44]:
```
# ----------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# ----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-constructio
# ----------------------------------

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-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_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(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_
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_test_merge, predict_y))
```

```
log_loss for c =  10 is 0.05493948003278959
log_loss for c =  50 is 0.0518475076237933
log_loss for c =  100 is 0.05292678628835201
log_loss for c =  500 is 0.05140306285509439
log_loss for c =  1000 is 0.05092537439335411
log_loss for c =  2000 is 0.0507139563201554
log_loss for c =  3000 is 0.050771525049528164
```




```
For values of best alpha =  2000 The train log loss is: 0.015808306583054568
For values of best alpha =  2000 The cross validation log loss is: 0.0507139563201554
For values of best alpha =  2000 The test log loss is: 0.035518140905318345
```

4.5.5. XgBoost Classifier on final features

In [60]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGB
# -------------------------
# 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_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# 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, verbose=True, xgb_model=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

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-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=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(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_
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_test_merge, predict_y))
```

```
log_loss for c =  10 is 0.08797867027475288
log_loss for c =  50 is 0.047388819790909704
log_loss for c =  100 is 0.04084775603236589
log_loss for c =  500 is 0.040252364121444735
log_loss for c =  1000 is 0.040356675669871014
log_loss for c =  2000 is 0.040372692315599104
log_loss for c =  3000 is 0.04037214332507081
```

```
For values of best alpha =  500 The train log loss is: 0.012394642168665237
For values of best alpha =  500 The cross validation log loss is: 0.040372143325070081
For values of best alpha =  500 The test log loss is: 0.04278914578208932
```

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

In [61]:
```
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  5.3min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  6.9min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 15.4min finished
```

Out[61]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [62]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 0.3}
```

In [63]:

```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGB
# -------------------------
# 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_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# 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, verbose=True, xgb_model=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.3,subsample=1,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(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_
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_test_merge, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
For values of best alpha =  500 The train log loss is: 0.011367740651602436
For values of best alpha =  500 The cross validation log loss is: 0.03913243373907577
For values of best alpha =  500 The test log loss is: 0.044696019027752026
Number of misclassified points  82.38270469181232
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video (https://www.youtube.com/watch?v=VLQTRlLGz5Y#t=13m11s)) and include pixel intensity features to improve the logloss

In [29]:
```
s = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26
```

In [30]:
```
# code source - http://www.albertauyeung.com/post/generating-ngrams-python/
import re

def generate_ngrams(s, n):
    # Convert to lowercases
    s = s.lower()

    # Replace all none alphanumeric characters with spaces
    s = re.sub(r'[^a-zA-Z0-9\s?]', ' ', s)

    # Break sentence in the token, remove empty tokens
    tokens = [token for token in s.split(" ") if token != ""]

    # Use the zip function to help us generate n-grams
    # Concatentate the tokens into ngrams and return
    ngrams = zip(*[tokens[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]
```

In [31]:
```
byte = generate_ngrams(s, 2)
```

In [32]:
```
len(byte)
```

Out[32]:
```
256
```

In [33]:
```
byte
```

Out[33]:
```
['00 01',
 '01 02',
 '02 03',
 '03 04',
 '04 05',
 '05 06',
 '06 07',
 '07 08',
 '08 09',
 '09 0a',
 '0a 0b',
 '0b 0c',
 '0c 0d',
 '0d 0e',
 '0e 0f',
 '0f 10',
 '10 11',
 '11 12',
 '12 13',
```

In [30]:
```
byte_bi_vocab = """00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,
```

In [31]:
```
byte_bigram_vocab = []
def byte_bigram():
    #byte_bigram_vocab = []
    for x, y in enumerate(byte_bi_vocab.split(',')):
        for z in range(0, len(byte_bi_vocab.split(','))):
            byte_bigram_vocab.append(y + ' ' +byte_bi_vocab.split(',')[z])
    return len(byte_bigram_vocab)
```

In [32]:
```
byte_bigram()
```

Out[32]:
```
66049
```

In [33]:
```
byte_bigram_vocab[:3]
```

Out[33]:
```
['00 00', '00 01', '00 02']
```

In [38]:
```
import os
os.getcwd()
```

Out[38]:
```
'D:\\malware'
```

In [39]:
```
%cd D:\\malware
```

```
D:\malware
```

In [40]:
```
os.getcwd()
```

Out[40]:
```
'D:\\malware'
```

Do not re-run below two cell

In [104]:
```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import scipy
vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
bytebigram_vect = scipy.sparse.csr_matrix((10868, 66049))
for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
    f = open('byteFiles/' + file)
    bytebigram_vect[i,:]+= scipy.sparse.csr_matrix(vec.fit_transform([f.read().replace('\n', ' ').lower()]))
    f.close()
```

```
10868it [33:49:34, 11.20s/it]
```

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

In [19]:
```
from sklearn.preprocessing import normalize
import scipy
byte_bi_vec = normalize(scipy.sparse.load_npz('byte_bigram.npz'), axis = 0)
```

In [28]:
```
def imp_features(data, features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    sns.set()
    plt.figure(figsize = (10, 5))
    ax = sns.barplot(x = imp_feature_name, y = imp_value)
    ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
    sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
    plt.title('Important Features')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')
    return imp_feature_indx[:keep]
```

In [34]:
```
byte_bi_indxes = imp_features(normalize(byte_bi_vec, axis = 0), byte_bigram_vocab, 300)
```

In [35]:
```
byte_bi_indxes
```

Out[35]:
```
array([63135, 41932, 19504, 52725, 59502, 60184, 33107,  4119, 38942,
       50950, 65604, 43213, 54966, 55472, 62116, 63088,  2989, 12224,
       62056,  7821, 20280, 63617,     8, 39200, 62089, 63105, 38941,
       10734, 39092, 58506,  7560, 10168, 62334, 36647, 11186, 12223,
       34922, 11224, 38797,  6509, 52215, 56872, 57324, 56923,  6790,
       12225, 26139, 48040, 33017, 11401, 34447, 64165, 58484, 62435,
       53990, 11017, 20277, 33615,  8080, 30828, 10838, 23374, 62234,
       18464, 58482, 62352, 58527, 59019, 61558, 58508, 31385, 58937,
       38850, 62628, 37759, 62316, 12003, 64717, 63619, 37826, 41879,
       36132, 39559, 58504, 37516, 55326, 44703,  6522, 62359, 57129,
       56018, 62631,  7092, 38535, 61545, 26709, 43642, 44771, 44126,
         105, 10404, 57779, 65620, 18248, 35978, 21128, 30590,  3388,
        4637, 50221, 17741, 47802, 35959, 35472, 50254, 34181, 17619,
       65018, 19797, 27242, 22873, 30695,  4128, 58946, 22093,   111,
       12085, 50115, 49833, 26343, 65264,  2512, 52125, 23010, 33021,
       29403,    18, 19801, 17991, 57803, 52776,  2162, 27864, 27254,
       17993, 32353, 52138, 31102, 35723, 65394, 60783, 63690, 50392,
       14742, 65644,   258, 29298, 47286, 43947,   157,   141, 14718,
       23913, 50766, 49858,  8176,  8326, 17749, 11244, 20303, 29337,
       35334, 13246, 33916, 33918,  8056, 65649,  1918, 59883,   182,
       35736, 16642, 14319, 36237, 30101, 63741,  1275, 13409, 27305,
       52428, 38034, 61769,   128, 33667, 20625, 15944, 29553,   205,
       25965, 27250, 64250,  1161,  5143,   137,     5, 49504, 65535,
         171, 65691, 30105, 13326, 40347,   164, 60909, 65687, 46258,
       65648, 26212,     4,   393, 29821, 28784, 36235, 16773, 65706,
       42660,    64, 42403, 64767, 48763, 50886, 35793,   101,  2701,
       65731, 64847,   106,    71, 64775, 14122,  8038, 57019,  2630,
        3087, 47288, 18327, 11056, 46517, 15109, 33153, 19789, 28270,
       35023, 35800,   213,   223, 26728, 33728,  3974,   139,   142,
           0, 35294, 21857,   328, 35816, 57054, 37008,   910,   206,
       50888,   238, 33866, 22704,   158,   199,    25, 40863, 28903,
       10769,    70, 17977, 55769,    83, 61423, 65693,  8340, 53451,
       35808,  4037,  1911,   195, 35791,    72, 15215, 20021, 11927,
       20033, 15145, 40659], dtype=int64)
```

In [36]:
```
np.save('byte_bi_indx', byte_bi_indxes)
```

In [37]:
```
byte_bi_indxes = np.load('byte_bi_indx.npy')
```

In [47]:
```
top_byte_bi = np.zeros((10868, 0))
for i in byte_bi_indxes:
    sliced = byte_bi_vec[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

In [48]:
```
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns = np.take(byte_bigram_vocab, byte_bi_indxes))
```

In [49]:
```
byte_bi_df.to_dense().to_csv('byte_bi.csv')
```

In [53]:
```
byte_bi_df = pd.read_csv('byte_bi.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [54]:
```
byte_bi_df['ID'] = result.ID
```

In [55]:
```
byte_bi_df.head()
```

Out[55]:

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	6d 00	13 8b	f4 38	d6 00
0	0.005295	0.000067	0.008667	0.008217	0.000510	0.000835	0.000307	0.000087	0.000033	0.000144	...	0.012699	0.000902	0.013709	0.010667
1	0.000000	0.000000	0.000000	0.000000	0.000510	0.000000	0.000077	0.000362	0.000265	0.052763	...	0.000233	0.000045	0.000000	0.000017
2	0.005295	0.000056	0.003250	0.005135	0.151488	0.000626	0.000384	0.000050	0.000099	0.000108	...	0.001577	0.001940	0.007834	0.001201
3	0.001059	0.000011	0.002167	0.000000	0.000510	0.000000	0.000537	0.000000	0.000066	0.000000	...	0.000657	0.000541	0.000979	0.000128
4	0.000000	0.000011	0.000000	0.001027	0.001020	0.000104	0.000230	0.000000	0.000000	0.000000	...	0.000010	0.000045	0.000000	0.000000

5 rows × 301 columns

In [56]:
```
byte_bi_df_1 = byte_bi_df.drop("ID", axis=1)
```

In [57]:
```
byte_bi_df_1
```

Out[57]:

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	65 65	6d 00	13 8b	f4
0	0.005295	0.000067	0.008667	0.008217	0.000510	0.000835	0.000307	0.000087	0.000033	0.000144	...	0.000436	0.012699	0.000902	0.013
1	0.000000	0.000000	0.000000	0.000000	0.000510	0.000000	0.000077	0.000362	0.000265	0.052763	...	0.000058	0.000233	0.000045	0.000
2	0.005295	0.000056	0.003250	0.005135	0.151488	0.000626	0.000384	0.000050	0.000099	0.000108	...	0.000054	0.001577	0.001940	0.007
3	0.001059	0.000011	0.002167	0.000000	0.000510	0.000000	0.000537	0.000000	0.000066	0.000000	...	0.000091	0.000657	0.000541	0.000
4	0.000000	0.000011	0.000000	0.001027	0.001020	0.000104	0.000230	0.000000	0.000000	0.000000	...	0.000021	0.000010	0.000045	0.000
5	0.003177	0.000000	0.001083	0.000000	0.000510	0.000209	0.000154	0.000012	0.000017	0.000018	...	0.000008	0.000162	0.000135	0.000
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000768	0.000349	0.000282	0.005210	...	0.000071	0.000061	0.000000	0.000
7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000844	0.000349	0.000282	0.000036	...	0.000054	0.000172	0.000226	0.000
8	0.002118	0.000022	0.001083	0.000000	0.000510	0.000104	0.000077	0.000062	0.000000	0.000000	...	0.000037	0.000435	0.000090	0.002
9	0.000000	0.000011	0.000000	0.000000	0.000000	0.000104	0.046054	0.000349	0.000265	0.010509	...	0.000050	0.000061	0.017594	0.000
10	0.000000	0.000000	0.001083	0.000000	0.000510	0.000104	0.000844	0.000374	0.000265	0.006363	...	0.000033	0.000091	0.003429	0.000
11	0.014826	0.000100	0.011918	0.002054	0.002550	0.000835	0.000768	0.000137	0.000149	0.000126	...	0.000083	0.000870	0.000632	0.013
12	0.014826	0.000266	0.017335	0.014379	0.009181	0.002817	0.001996	0.000299	0.000530	0.000144	...	0.000083	0.000243	0.001985	0.019
13	0.013767	0.000155	0.018418	0.014379	0.005101	0.000939	0.000691	0.000175	0.000215	0.000198	...	0.000046	0.000101	0.000496	0.015
14	0.001059	0.000011	0.001083	0.002054	0.000510	0.000104	0.000307	0.000000	0.000000	0.000018	...	0.000046	0.001082	0.000451	0.000
15	0.000000	0.000000	0.002167	0.000000	0.000000	0.000000	0.000077	0.000012	0.000017	0.000000	...	0.000025	0.000061	0.000090	0.000
16	0.001059	0.000000	0.001083	0.001027	0.000000	0.000104	0.000000	0.000012	0.000000	0.000018	...	0.000037	0.000516	0.000135	0.000
17	0.010590	0.000133	0.011918	0.015406	0.008161	0.001252	0.001228	0.000112	0.000232	0.000288	...	0.000033	0.000162	0.000361	0.008
18	0.008472	0.000122	0.008667	0.015406	0.007651	0.001356	0.001151	0.000112	0.000182	0.000306	...	0.000075	0.000131	0.000316	0.010
19	0.000000	0.000011	0.000000	0.000000	0.000000	0.000000	0.000844	0.000349	0.000282	0.005246	...	0.000033	0.000586	0.000406	0.000
20	0.000000	0.000011	0.003250	0.001027	0.000510	0.000000	0.000000	0.000025	0.000033	0.000018	...	0.000075	0.000364	0.000677	0.001
21	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000844	0.000349	0.000265	0.000018	...	0.000046	0.000061	0.000361	0.000
22	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.023334	0.000349	0.000265	0.008851	...	0.000033	0.000394	0.016421	0.000
23	0.005295	0.000089	0.003250	0.007190	0.001020	0.000626	0.000921	0.000100	0.000017	0.000090	...	0.000046	0.002659	0.000406	0.012

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	65 65	6d 00	13 8b	f4
24	0.002118	0.000033	0.001083	0.000000	0.001530	0.000209	0.000921	0.000237	0.000232	0.000072	...	0.000095	0.003529	0.001444	0.004
25	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000921	0.000349	0.000265	0.005282	...	0.000083	0.000091	0.000000	0.000
26	0.002118	0.000000	0.000000	0.000000	0.000000	0.000104	0.000844	0.000437	0.000298	0.008490	...	0.000042	0.000243	0.008842	0.000
27	0.011649	0.000233	0.009751	0.010271	0.008671	0.001148	0.000844	0.000150	0.000215	0.000180	...	0.000075	0.000212	0.000586	0.010
28	0.013767	0.000100	0.014084	0.012325	0.006631	0.001252	0.001151	0.000337	0.000199	0.000198	...	0.000075	0.000202	0.000496	0.010
29	0.011649	0.000122	0.015168	0.013352	0.007141	0.001565	0.000768	0.000150	0.000149	0.000234	...	0.000075	0.000222	0.000812	0.011
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10838	0.000000	0.000000	0.000000	0.000000	0.000510	0.000104	0.000000	0.000012	0.000000	0.000000	...	0.000017	0.000030	0.000000	0.000
10839	0.001059	0.000011	0.001083	0.001027	0.000510	0.000104	0.000000	0.000012	0.000066	0.000000	...	0.000008	0.000010	0.000045	0.000
10840	0.000000	0.000011	0.001083	0.001027	0.001530	0.000000	0.000154	0.000025	0.000017	0.000000	...	0.000012	0.000081	0.000000	0.000
10841	0.000000	0.000011	0.001083	0.001027	0.000000	0.000000	0.000154	0.000012	0.000033	0.000000	...	0.000021	0.000030	0.000090	0.000
10842	0.000000	0.000000	0.000000	0.000000	0.001530	0.000000	0.000000	0.000012	0.000017	0.000000	...	0.000012	0.000010	0.000000	0.000
10843	0.003177	0.000022	0.000000	0.002054	0.000510	0.000313	0.000154	0.000012	0.000033	0.000000	...	0.000004	0.000020	0.000000	0.005
10844	0.002118	0.000011	0.002167	0.000000	0.000510	0.000000	0.000230	0.000025	0.000033	0.000000	...	0.000000	0.000000	0.000090	0.001
10845	0.000000	0.000022	0.000000	0.003081	0.001020	0.000104	0.000154	0.000012	0.000000	0.000036	...	0.000012	0.000334	0.000045	0.001
10846	0.001059	0.000111	0.004334	0.001027	0.002040	0.000522	0.000384	0.000000	0.000033	0.000018	...	0.000012	0.000051	0.000180	0.004
10847	0.006354	0.000033	0.002167	0.005135	0.001530	0.000522	0.000230	0.000037	0.000133	0.000054	...	0.000029	0.000101	0.000090	0.003
10848	0.004236	0.000033	0.003250	0.002054	0.002040	0.000000	0.000077	0.000025	0.000017	0.000000	...	0.000004	0.000020	0.000090	0.000
10849	0.000000	0.000000	0.001083	0.002054	0.001020	0.000000	0.000077	0.000012	0.000017	0.000036	...	0.000000	0.000030	0.000090	0.001
10850	0.002118	0.000022	0.006500	0.005135	0.002040	0.000417	0.000154	0.000025	0.000099	0.000144	...	0.000046	0.000415	0.000045	0.002
10851	0.000000	0.000000	0.001083	0.001027	0.000000	0.000000	0.000000	0.000000	0.000017	0.000000	...	0.000000	0.000010	0.000000	0.000
10852	0.000000	0.000011	0.002167	0.003081	0.001530	0.000209	0.000000	0.000000	0.000017	0.000036	...	0.000012	0.000020	0.000000	0.001
10853	0.001059	0.000000	0.001083	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000010	0.000000	0.000
10854	0.000000	0.000022	0.001083	0.000000	0.000510	0.000209	0.000154	0.000025	0.000017	0.000000	...	0.000021	0.000334	0.000000	0.000
10855	0.002118	0.000000	0.000000	0.000000	0.001530	0.000209	0.000077	0.000000	0.000000	0.000036	...	0.000000	0.000020	0.000090	0.000
10856	0.002118	0.000011	0.001083	0.001027	0.000000	0.000000	0.000154	0.000000	0.000000	0.000000	...	0.000008	0.000061	0.000045	0.000

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	65 65	6d 00	13 8b	f4
10857	0.001059	0.000000	0.001083	0.005135	0.001530	0.000104	0.000077	0.000037	0.000000	0.000036	...	0.000012	0.000030	0.000045	0.001
10858	0.006354	0.000011	0.006500	0.008217	0.001020	0.000104	0.000461	0.000050	0.000066	0.000090	...	0.000033	0.000061	0.000090	0.002
10859	0.001059	0.000000	0.002167	0.001027	0.000510	0.000104	0.000077	0.000025	0.000066	0.000000	...	0.000017	0.000010	0.000045	0.000
10860	0.001059	0.000000	0.002167	0.000000	0.001020	0.000209	0.000000	0.000012	0.000000	0.000036	...	0.000012	0.000324	0.000045	0.002
10861	0.003177	0.000000	0.002167	0.001027	0.000000	0.000104	0.000000	0.000037	0.000033	0.000018	...	0.000004	0.000344	0.000045	0.002
10862	0.002118	0.000033	0.003250	0.006163	0.002040	0.000522	0.000000	0.000050	0.000066	0.000090	...	0.000017	0.000071	0.000135	0.004
10863	0.002118	0.000022	0.003250	0.003081	0.003570	0.000730	0.000384	0.000025	0.000033	0.000036	...	0.000025	0.000030	0.000316	0.003
10864	0.001059	0.000000	0.000000	0.001027	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000061	0.000000	0.000
10865	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000012	0.000000	0.000000	...	0.000000	0.000313	0.000000	0.000
10866	0.000000	0.000000	0.001083	0.000000	0.000510	0.000209	0.000077	0.000025	0.000017	0.000018	...	0.000008	0.000040	0.000090	0.000
10867	0.000000	0.000011	0.001083	0.000000	0.000000	0.000104	0.000154	0.000012	0.000017	0.000036	...	0.000004	0.000010	0.000000	0.000

10868 rows × 300 columns

Bi-grams on ASM FIles

In [43]:
```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 's
```

In [44]:
```
asm_bigram_vocab = []
def asm_bigram():
    #byte_bigram_vocab = []
    for x, y in enumerate(opcodes):
        for z in range(0, len(opcodes)):
            asm_bigram_vocab.append(y + ' ' + opcodes[z])
    return len(asm_bigram_vocab)
```

In [45]:
```
asm_bigram()
```

Out[45]:
```
676
```

In [46]:
```
asm_bigram_vocab[:3]
```

Out[46]:
```
['jmp jmp', 'jmp mov', 'jmp retf']
```

In [115]:
```
os.getcwd()
```

Out[115]:
```
'D:\\malware'
```

In [117]:
```
def opcode_collect():
    op_file = open("opcode_file.txt", "w+")
    for asmfile in tqdm(os.listdir('asmFiles')):
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as file:
            for lines in file:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
opcode_collect()
```

```
100%|██████████████████████████████████████████████████| 10868/10868 [5:23:26<00:00,  1.79s/it]
```

In [122]:
```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import scipy
vec = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=asm_bigram_vocab)
asm_bi_vec = scipy.sparse.csr_matrix((10868, len(asm_bigram_vocab)))
for i in tqdm(range(10868)):
    raw = open('opcode_file.txt').read().split('\n')
    asm_bi_vec[i, :] += scipy.sparse.csr_matrix(vec.transform([raw[i]]))
```

```
100%|██████████████████████████████████████████████████| 10868/10868 [8:30:25<00:00,  2.82s/it]
```

In [123]:
```
scipy.sparse.save_npz('asm_bigram.npz', asm_bi_vec)
```

In [47]:
```
from sklearn.preprocessing import normalize
import scipy
asm_bi_vec = normalize(scipy.sparse.load_npz('asm_bigram.npz'), axis = 0)
```

In [48]:
```
asm_bi_indx = imp_features(normalize(asm_bi_vec, axis = 0), asm_bigram_vocab, 300)
```

In [49]:
```
asm_bi_indx
```

Out[49]:
```
array([ 30, 432, 458, 159, 373,  26, 573, 290, 438, 437, 287, 302, 274,
        27,  50,  37, 625, 252, 131, 224, 477, 136,   3, 453,  51,  48,
        43, 588,  10, 442, 209, 232, 445, 417, 108, 115, 492, 110, 466,
       157, 628, 268,   1,  34, 235,  81,   5, 446, 658, 580, 134, 292,
       547, 365,  16,  36,  32, 130,  42,  41, 133, 627, 156, 211, 112,
       464, 146,   0, 447, 416, 596, 448,  79, 641, 443, 261,  96, 226,
       650, 440,  29, 581,  22, 380, 664, 250, 575, 161,  38, 594, 469,
       666, 655, 238, 382,  24, 456, 234, 582, 208, 303, 468, 248, 405,
       421, 107, 289, 218, 427,  86,   8, 102,  95, 313, 121,  35, 172,
       589, 430, 341,  11,  25, 638,  92, 310, 139, 635, 364, 225, 367,
       441, 375, 219, 572,  58, 105, 583, 419, 242, 651,  82, 404,  89,
       648, 264, 213, 297,  31, 632,  88, 369, 138,  54, 212, 300, 675,
       646,  83, 284,  44, 425,  40, 118, 155,  17, 562,  87, 640,   4,
       412, 222, 286, 158, 577,  94,  53,  84, 338, 342, 109, 141, 276,
       294, 162,  91, 296, 217,  78,   2, 126, 315,  68, 233, 661, 586,
       426, 154, 370, 451, 340, 471, 262, 467,  63, 258, 401, 597, 164,
       236, 180, 216, 455, 498, 144, 239, 173, 653, 251, 386,  28, 237,
       624, 570, 308, 649, 629, 639, 372, 584, 282, 170,   9, 551,  98,
       167, 263, 563,   6, 266, 420, 178, 120, 326,  52, 143, 100, 587,
       389,  55,  61, 656, 391, 288, 230, 104,  56, 424, 414, 388,  97,
        13, 642, 186, 183, 106, 344, 479, 556, 459, 472,  45, 406, 366,
       376, 672, 339, 114, 291,  14, 295, 549, 450, 662, 278, 103, 128,
       528, 396, 674, 394, 576, 147, 505,  62, 555, 298, 243, 422, 174,
       452], dtype=int64)
```

In [50]:
```
np.save('asm_bi_indx',asm_bi_indx)
```

In [21]:
```
asm_bi_indxes = np.load('asm_bi_indx.npy')
```

In [52]:
```
top_asm_bi = np.zeros((10868, 0))
for i in asm_bi_indxes:
    sliced = asm_bi_vec[:, i].todense()
    top_asm_bi = np.hstack([top_asm_bi, sliced])
```

In [53]:
```
asm_bi_df = pd.SparseDataFrame(top_asm_bi, columns = np.take(asm_bigram_vocab, asm_bi_indxes))
```

In [54]:
```
asm_bi_df.to_dense().to_csv('asm_bi.csv')
```

In [22]:
```
asm_bi_df = pd.read_csv('asm_bi.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [23]:
```
asm_bi_df['ID'] = result.ID
```

In [24]:
```
asm_bi_df.head()
```

Out[24]:

	mov pop	cmp cmp	call cmp	retn push	or inc	mov jmp	jz mov	add pop	cmp jz	cmp jnb	...	xor call	ror add	retf dec	jnb inc	add imul	inc inc
0	0.004581	0.000922	0.008045	0.002184	0.0	0.003770	0.005517	0.001096	0.000126	0.000000	...	0.005794	0.0	0.00	0.0	0.000565	0.000000
1	0.001497	0.000503	0.000380	0.000898	0.0	0.000777	0.000298	0.000274	0.000461	0.003746	...	0.000658	0.0	0.00	0.0	0.000000	0.000038
2	0.000000	0.002432	0.000326	0.000748	0.0	0.000000	0.000000	0.000000	0.000587	0.000000	...	0.000132	0.0	0.01	0.0	0.000000	0.000000
3	0.000091	0.000084	0.000109	0.000150	0.0	0.000069	0.000099	0.000000	0.000063	0.000489	...	0.000658	0.0	0.00	0.0	0.000000	0.000000
4	0.000499	0.000168	0.000054	0.000569	0.0	0.001165	0.001290	0.000183	0.001613	0.002443	...	0.000000	0.0	0.00	0.0	0.000000	0.000000

5 rows × 301 columns

In [25]:
```
asm_bi_df = asm_bi_df.drop("ID", axis=1)
```

In [26]:
```
asm_bi_df.head()
```

Out[26]:

	mov pop	cmp cmp	call cmp	retn push	or inc	mov jmp	jz mov	add pop	cmp jz	cmp jnb	...	jz pop	xor call	ror add	retf dec	jnb inc	add imul
0	0.004581	0.000922	0.008045	0.002184	0.0	0.003770	0.005517	0.001096	0.000126	0.000000	...	0.000568	0.005794	0.0	0.00	0.0	0.000565
1	0.001497	0.000503	0.000380	0.000898	0.0	0.000777	0.000298	0.000274	0.000461	0.003746	...	0.000000	0.000658	0.0	0.00	0.0	0.000000
2	0.000000	0.002432	0.000326	0.000748	0.0	0.000000	0.000000	0.000000	0.000587	0.000000	...	0.000000	0.000132	0.0	0.01	0.0	0.000000
3	0.000091	0.000084	0.000109	0.000150	0.0	0.000069	0.000099	0.000000	0.000063	0.000489	...	0.000000	0.000658	0.0	0.00	0.0	0.000000
4	0.000499	0.000168	0.000054	0.000569	0.0	0.001165	0.001290	0.000183	0.001613	0.002443	...	0.000000	0.000000	0.0	0.00	0.0	0.000000

5 rows × 300 columns

In [60]:
```
asm_trigram_vocab = []
def asm_trigram():
    for x, y in enumerate(opcodes):
        for w in range(0, len(opcodes)):
            for z in range(0, len(opcodes)):
                asm_trigram_vocab.append(y + ' ' + opcodes[w] + ' ' + opcodes[z])
    return len(asm_trigram_vocab)
```

In [61]:
```
asm_trigram()
```

Out[61]: 17576

In [127]:
```
tri_vec = CountVectorizer(lowercase=False,ngram_range=(3,3), vocabulary=asm_trigram_vocab)
asm_tri_vec = scipy.sparse.csr_matrix((10868, len(asm_trigram_vocab)))
for i in tqdm(range(10868)):
    raw = open('opcode_file.txt').read().split('\n')
    asm_tri_vec[i, :] += scipy.sparse.csr_matrix(tri_vec.transform([raw[i]]))
```

```
100%|██████████████████████████████████████████████████| 10868/10868 [8:52:35<00:00,  2.94s/it]
```

In [128]:
```
scipy.sparse.save_npz('asm_trigram.npz', asm_tri_vec)
```

In [62]:
```
from sklearn.preprocessing import normalize
import scipy
asm_tri_vec = normalize(scipy.sparse.load_npz('asm_trigram.npz'), axis = 0)
```

In [63]:
```
asm_tri_indx = imp_features(normalize(asm_tri_vec, axis = 0), asm_trigram_vocab, 300)
```

In [64]:
```
asm_tri_indx
```

Out[64]:
```
array([  713,  2863,   703,   908, 11782,  7463, 14899,  7543,   677,
       11363,  4085, 15310,   755,  2486,   797,  6971,  2520,  2783,
       11404,   719,  1058,   438, 11389, 11540,  4190, 14244,  5669,
        6035,  1301,  2814,   718,  2481,  9712, 14967,  1093,  2861,
         786,  5547, 14953,  2994,    37,  3157,   963,   784,   717,
         707,   960,  4135, 14898, 11253,  2071, 12184,  1251,  2055,
        2928,  1265, 11602,  6043,  2109,  6576, 16260,  1113,  7546,
        6037, 14914,   708,  7557,  4018,  7385,   720,  1153,  5542,
         807,  3492,  6048,   706,   885,  6297,   432,  6085,  7473,
         425,  3597,  6040, 11396,  2237,   596,  4083, 14909,   727,
        2799,  2130,  5435, 16253,  2492,  2149,  7879,  2478, 11542,
       14920,  2107, 17566,  3799,   765,  2474,  7710,  1317,   675,
         726,  4088,   705, 11649,  3279, 15336, 10847, 15325,  6046,
       12202,  1134,   926,  2921,  2812,  7853,  9706,  2752,   686,
        2317,  3336,   681, 11600, 16435,  4004, 12078,  9933,  2470,
        6033,  2070,  1121,  1334,  1041,  6553,   887,  3797,  7649,
       16357,  2062,   812,  4499,  2475,  7487,  2140,  7868,  3501,
        3536,  2471,  1108,  4166,   417, 17265, 11519, 16979,   757,
        5845, 14915,    42, 14233,   762,   754, 15304, 10817,  7177,
          30,  7478, 11778,  5866, 11412,  2484,  2110, 10970, 11618,
        7486,  5437, 10897,  1405, 11518,   758, 11393,  2884, 11571,
       15499,  9545,   973, 16274,  6501, 11066, 10842, 12088,  5434,
       11930, 17033, 14901,  5660, 11651, 11522,  9932,  2473,   771,
        9465, 11587,  2270,     3,  2132,     0, 11623, 10911, 11534,
        1114, 16643, 11934, 12080, 14223,  2065,  1273,  5867, 16305,
        2655,  2838,   679, 16272, 12119,  4140,   702, 16319,  2865,
         724, 12208,  2123, 10859,    51,   806, 11639,  7517,  1272,
       16266,  7541,   979, 15133,    81, 14639,   692,  2239,  2733,
       11521,  2156,  4086,  5851, 17316, 16363, 16981,    48,  4192,
        4767, 16849,  2808, 16306, 11599, 12793,  2806,  2476,  2078,
       11532,  9701,  6522,  6449,  8139,  5825,  8155,  5489,  2057,
       11497,  5438,   978,   714,  1249,   710, 11932, 11717, 10895,
        2114, 11092,  1303,  2111, 14873,   822, 11242, 10898, 11779,
        9907, 15115,  1119], dtype=int64)
```

In [65]:
```
np.save('asm_tri_indx',asm_tri_indx)
```

In [66]:
```
top_asm_tri = np.zeros((10868, 0))
for i in asm_tri_indx:
    sliced = asm_tri_vec[:, i].todense()
    top_asm_tri = np.hstack([top_asm_tri, sliced])
```

In [67]:
```
asm_tri_df = pd.SparseDataFrame(top_asm_tri, columns = np.take(asm_trigram_vocab, asm_tri_indx))
```

In [68]:
```
asm_tri_df.to_dense().to_csv('asm_tri.csv')
```

In [27]:
```
asm_tri_df = pd.read_csv('asm_tri.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [28]:
```
asm_tri_df['ID'] = result.ID
```

In [29]:
```
asm_tri_df =  asm_tri_df.drop("ID", axis=1)
```

In [30]:
```
asm_tri_df.head()
```

Out[30]:

	mov mov add	pop retn push	mov mov mov	mov sub lea	call add pop	add mov mov	jz mov mov	add pop push	mov jmp mov	cmp jnb mov	...	mov lea push	push push xor	jz jmp mov	mov xor cmp
0	0.005770	0.001802	0.001211	0.001002	0.000000	0.003593	0.003869	0.000000	0.004309	0.000000	...	0.000750	0.011470	0.000000	0.005202
1	0.003435	0.001073	0.001544	0.002506	0.000251	0.002705	0.000221	0.000000	0.001048	0.004019	...	0.000346	0.000000	0.000000	0.000000
2	0.000000	0.000129	0.000000	0.003007	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000115	0.003036	0.000000	0.000000
3	0.000000	0.000000	0.000010	0.000000	0.000000	0.000000	0.000111	0.000000	0.000039	0.000670	...	0.000115	0.000000	0.001077	0.000000
4	0.013433	0.000815	0.001047	0.000501	0.000000	0.014009	0.001105	0.000441	0.001825	0.005024	...	0.000231	0.000000	0.002154	0.000000

5 rows × 300 columns

In [73]:
```
asm_4gram_vocab = []
def asm_4gram():
    for x, y in enumerate(opcodes):
        for v in range(0, len(opcodes)):
            for w in range(0, len(opcodes)):
                for z in range(0, len(opcodes)):
                    asm_4gram_vocab.append(y + ' ' + opcodes[v] + ' ' + opcodes[w] + ' ' + opcodes[z])
    return len(asm_4gram_vocab)
```

In [74]:
```
asm_4gram()
```

Out[74]: 456976

In [133]:
```
from tqdm.notebook import tqdm
import scipy
from sklearn.feature_extraction.text import CountVectorizer
four_vec = CountVectorizer(lowercase=False, ngram_range=(4,4), vocabulary=asm_4gram_vocab)
asm_4_vec = scipy.sparse.csr_matrix((10868, len(asm_4gram_vocab)))
for i in tqdm(range(10868)):
    raw = open('opcode_file.txt').read().split('\n')
    asm_4_vec[i, :] += scipy.sparse.csr_matrix(four_vec.transform([raw[i]]))
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

In [134]:
```
scipy.sparse.save_npz('asm_fourgram.npz', asm_4_vec)
```

In [75]:
```
asm_4_vec = normalize(scipy.sparse.load_npz('asm_fourgram.npz'), axis = 0)
```

In [76]:
```
asm_4_indx = imp_features(normalize(asm_4_vec, axis = 0), asm_4gram_vocab, 300)
```

In [77]:
```
asm_4_indx
```

Out[77]:
```
array([ 23613,  82086,  23405,  19638,  27508,  23611, 106211, 370033,
        53636,  34555, 292579, 296323, 106159,  18279,  54783, 306335,
        23619,  25038, 429341, 301653,  71009,  29485, 306333,  23622,
       147161,  54851, 306349, 370361,  54785, 196667,  73169,  55220,
        18289, 438129, 108946, 249881,  23616,   6971,  99133, 299906,
        88583, 296183, 107563,  53441, 456738, 196135, 247120, 306648,
       432667, 314291,  72785,  19699,  28527,   2109,  18411, 387351,
        18827, 292054,  55209, 398466,  56212, 196197,  71612,  28980,
       246772,  34914, 258248,  58166,  24961, 247209, 252230,  53615,
        20388,  17603, 283687,  93540, 309635, 423140, 313759, 306338,
        64795,  23609, 156861,  23010,  85257,  53850, 306254,  18253,
       283372,  17566,  18690,  74387,  71075,  18513,  74389,  29771,
        32488,  18281, 142717, 427313, 107527, 196142,  29095, 301786,
        55201,  90600,  91032, 185466,  23697, 156882,  21399,  55215,
        19686, 423933,  72444, 282046, 387375, 299604,  91685, 281893,
        20439, 107931, 107511,  17610,  18334,  25039,  54835, 299495,
        20062,  71088,  18288, 423986,  10843,  54031,  19633, 107513,
       387479,  17655,  73113, 194429,  55885,    713,  64658,  18476,
       300915, 274145, 370059, 194639, 306645, 422932,  53875, 196121,
        54787,  33879,  18903,  18352, 306258,  55658,  55212,  73118,
       108332,  64346,  72413,  18294, 369853, 296131,  20310,  74782,
         2471,  17657,  21058,  70721,  32498,  53490,  69369,  19685,
       141365,  23013,  64637,  23011,  18266,    757,  53485,  18295,
       287249, 200858,  34217,  73167,  92015,  32830, 370191, 296513,
        34527,  55203, 317078,  23065, 292396,  64506, 264681, 450789,
        27067,  32485,  53977,  11389,  34503, 397926,  87907, 247177,
        64250, 158221, 194093, 296754,  64325, 181251,  56540,  32901,
        18338, 106213, 281243,  64322,   1113, 157121, 194286,  18538,
         1342, 369981, 299516,  74441, 298819,    765, 301655, 385422,
       194042,  18287,  29058,  64431, 397590, 301604,     37, 424479,
        11404, 389142, 281940,  29116,  64270, 450216,  69033, 106345,
        34232,  29641, 141314,  64330,  28426,  64246, 369822, 194049,
       193362, 422501, 106180,    963, 313691,     81,  34892,  56189,
       144083, 423879,  54967, 299209, 296125, 317253, 432721,  54806,
        27082, 141328, 246767,  20050, 195445, 450253, 194039,  32890,
       398760, 106234, 281926,  72959, 301265,  89989, 387921,  28422,
       427493, 152559, 185461,  25048, 158896,  53613,  74444, 387374,
       309650, 295441, 106172,  20047], dtype=int64)
```

In [78]:
```
np.save('asm_4_indx', asm_4_indx)
```

In [79]:
```
asm_4_indx = np.load('asm_4_indx.npy')
```

In [80]:
```
top_asm_four = np.zeros((10868, 0))
for i in asm_4_indx:
    sliced = asm_4_vec[:, i].todense()
    top_asm_four = np.hstack([top_asm_four, sliced])
```

In [81]:
```
asm_4_df = pd.SparseDataFrame(top_asm_four, columns = np.take(asm_4gram_vocab, asm_4_indx))
```

In [83]:
```
asm_4_df.to_dense().to_csv('asm_4_df.csv')
```

In [31]:
```
asm_4_df = pd.read_csv('asm_4_df.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [32]:
```
asm_4_df['ID'] = result.ID
```

In [33]:
```
asm_4_df = asm_4_df.drop("ID", axis=1)
```

In [34]:
```
asm_4_df.head()
```

Out[34]:

	mov sub lea xor	pop call add pop	mov sub cmp xor	mov push mov sub	mov or shl jmp	mov sub lea push	retn mov push mov	jnb mov dec mov	push mov sub lea	mov movzx push mov	...	dec or inc push	mov add mov dec	inc mov mov dec	push mov sub mov	pop retn push retn	jz mov mov jmp	call cmp mov cmp	cmp jnb mov push	re
0	0.005011	0.0	0.0	0.001617	0.0	0.000000	0.000286	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.000886	0.0	0.0	0.0	0.000000	0.
1	0.000000	0.0	0.0	0.000147	0.0	0.000000	0.000906	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.013954	0.
2	0.000000	0.0	0.0	0.000000	0.0	0.009915	0.000000	0.0	0.003646	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.
3	0.000000	0.0	0.0	0.000147	0.0	0.000000	0.000000	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.000665	0.0	0.0	0.0	0.000000	0.
4	0.000000	0.0	0.0	0.000000	0.0	0.001652	0.000000	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.003989	0.0	0.0	0.0	0.003489	0.

5 rows × 300 columns

In [74]:
```
result_x1 = byte_bi_df_1
result_y = data_y
```

Train & test split

In [75]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x1, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

In [76]:
```
X_train.shape, y_train.shape
```

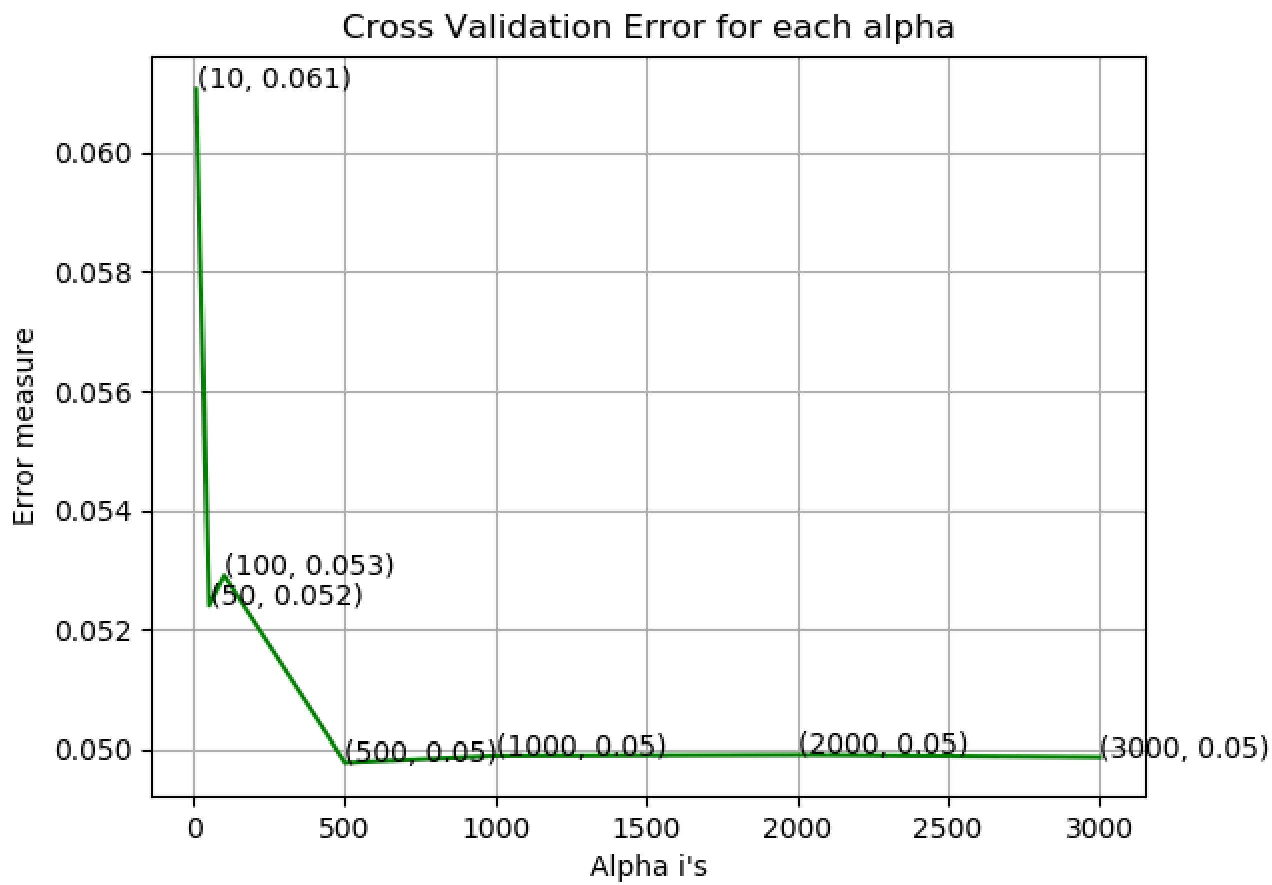
Out[76]:
```
((8694, 300), (8694,))
```

Random Forest Classifier on byte bi-grams Files

In [77]:
```
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-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()
```

```
log_loss for c =  10 is 0.06105454136327116
log_loss for c =  50 is 0.05240544371971374
log_loss for c =  100 is 0.052910707678592864
log_loss for c =  500 is 0.049784618923179064
log_loss for c =  1000 is 0.04989981654166991
log_loss for c =  2000 is 0.049916409716371374
log_loss for c =  3000 is 0.049878502512113045
```

Figure 10



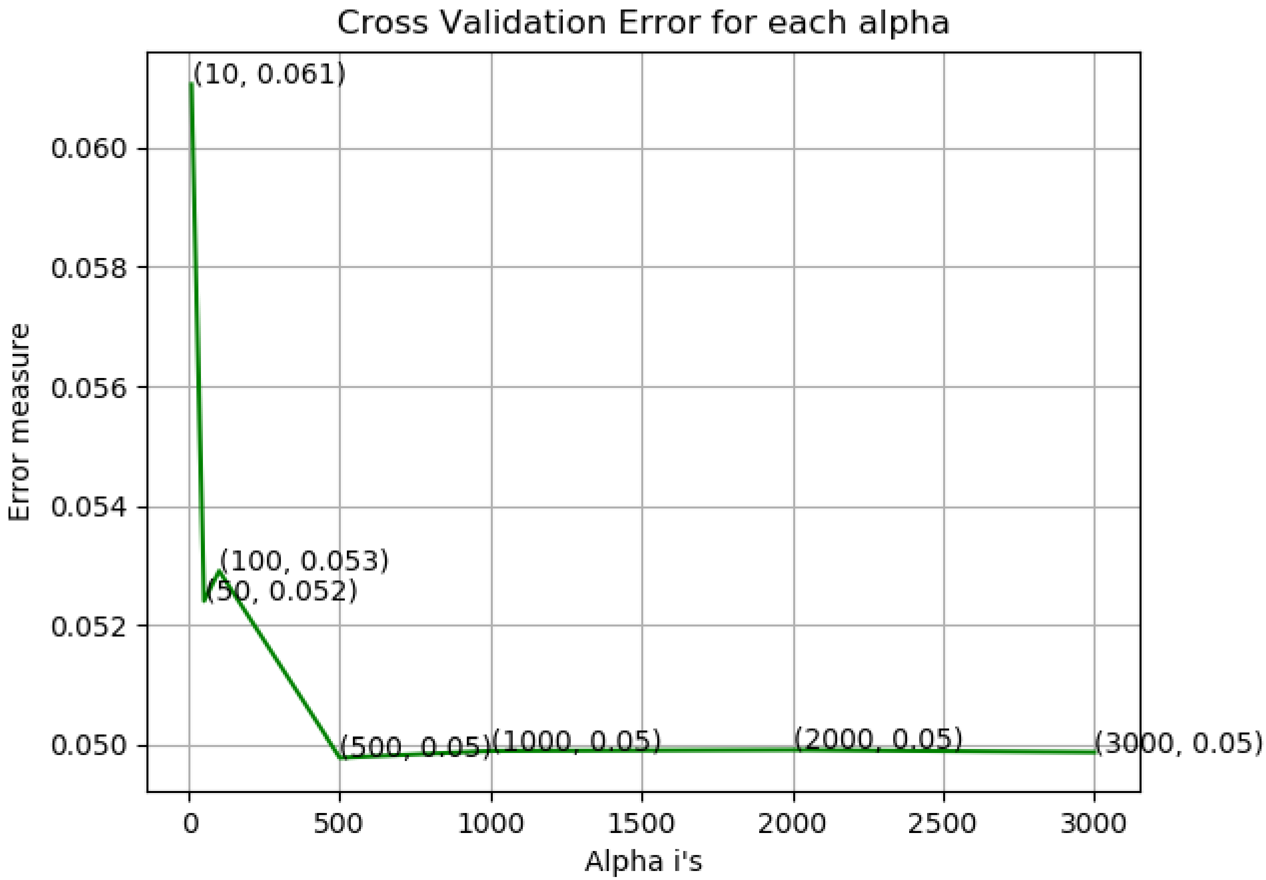
x=2916.27 y=0.0542391

Random Forest Classifier On Bytes Bi-gram

In [78]:
```
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_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-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()
```

```
log_loss for c =  10 is 0.06105454136327116
log_loss for c =  50 is 0.05240544371971374
log_loss for c =  100 is 0.052910707678592864
log_loss for c =  500 is 0.049784618923179064
log_loss for c =  1000 is 0.04989981654166991
log_loss for c =  2000 is 0.049916409716371374
log_loss for c =  3000 is 0.049878502512113045
```

Figure 11


Cross Validation Error for each alpha
(10, 0.061)
(100, 0.053)
(50, 0.052)
(500, 0.05)
(1000, 0.05)
(2000, 0.05)
(3000, 0.05)
Error measure
0.060
0.058
0.056
0.054
0.052
0.050
0
500
1000
1500
2000
2500
3000
Alpha i's

In [79]:
```
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 ('log loss for train data',(log_loss(y_train_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_merge)
print ('log loss for cv data',(log_loss(y_cv_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_merge)
print ('log loss for test data',(log_loss(y_test_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
log loss for train data 0.017321593486531485
log loss for cv data 0.049784618923179064
log loss for test data 0.042919980811460784
Number of misclassified points  0.8279668813247469
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

HyperParameter Tuned XGBoost on Byte Bi-gram Files

In [80]:
```
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  5.6min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  9.2min remaining:  1.0min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 10.9min finished
```

Out[80]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [81]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.2, 'colsample_bytree': 0.5}
```

In [82]:

```
x_cfl=XGBClassifier(n_estimators=500, learning_rate=0.2, colsample_bytree=0.5, max_depth=5, subsample=0.5, n_jobs=-1)
x_cfl.fit(X_train_merge, y_train_merge)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_merge, y_train_merge)

predict_y = c_cfl.predict_proba(X_train_merge)
print ('train loss',log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))
```

```
train loss 0.015682084702440886
cv loss 0.04869234506218092
test loss 0.04340585792521129
```

In [83]:

```
plt.close()
plot_confusion_matrix(y_test_merge,c_cfl.predict(X_test_merge))
```

```
Number of misclassified points  0.6899724011039559
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

Extracing Image Features from ASM Files

Do not re-run this cell

In [69]:
```
#code source - https://towardsdatascience.com/malware-classification-using-machine-learning-7c648fb1da79

import time,array
import imageio


for afile in tqdm(os.listdir("asmFiles"), position=0, leave=True):
        start_time = time.time()
        asm_file_name = afile.split('.')[0]
        file = codecs.open("asmFiles/" + afile, 'rb')
        asm_file_len = os.path.getsize("asmFiles/" + afile)

        width = int(asm_file_len ** 0.5)
        rem = int(asm_file_len / width)

        imgarr = array.array('B')
        imgarr.frombytes(file.read())

        file.close()
        re_img = np.reshape(imgarr[:width * width], (width, width))
        re_img = np.uint8(re_img)

        imageio.imwrite('asmImage/' + asm_file_name + '.png',re_img)
print('File conversion Successful!!!')
```

```
100%|██████████████████████████████████████████████████| 10868/10868 [3:10:03<00:00,  1.05s/it]

File conversion Successful!!!
```

In [61]:
```
!pip install pillow
```

```
Requirement already satisfied: pillow in c:\users\hims1\anaconda3\lib\site-packages (5.1.0)

thinc 6.10.3 requires msgpack<1.0.0,>=0.5.6, which is not installed.
msgpack-numpy 0.4.4.3 requires msgpack>=0.5.2, which is not installed.
distributed 1.21.8 requires msgpack, which is not installed.
spacy 2.0.13 has requirement msgpack-numpy<0.29,<0.4.4.0murmurhash>=0.28, but you'll have msgpack-numpy 0.4.4.3 which i
s incompatible.
spacy 2.0.13 has requirement numpy>=1.15.0, but you'll have numpy 1.14.3 which is incompatible.
spacy 2.0.13 has requirement regex==2018.01.10, but you'll have regex 2017.11.9 which is incompatible.
jupyterlab-server 1.0.0 has requirement jsonschema>=3.0.1, but you'll have jsonschema 2.6.0 which is incompatible.
You are using pip version 10.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [88]:
```
from PIL import Image
%matplotlib inline

rows = 5
path= 'D:\malware/asmImage'

os.chdir(path)
files=os.listdir(path)
files=files[:15]


plt.figure(figsize=(25,50))
for num, f in enumerate(files):
  try:
    img = PIL.Image.open(path+'/'+f)
    plt.subplot(rows,3,num+1)
    plt.axis('off')
    plt.imshow(img)

  except Exception as e:  # for clean output
    print(e)
    pass
```

```
Traceback (most recent call last):
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py", line 388, in process
    proxy(*args, **kwargs)
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py", line 228, in __call__
    return mtd(*args, **kwargs)
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\backends\backend_nbagg.py", line 241, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\_pylab_helpers.py", line 58, in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py", line 388, in process
    proxy(*args, **kwargs)
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py", line 228, in __call__
    return mtd(*args, **kwargs)
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\backends\backend_nbagg.py", line 241, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "C:\Users\hims1\Anaconda3\lib\site-packages\matplotlib\_pylab_helpers.py", line 58, in destroy
```

```
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list

name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined
name 'PIL' is not defined

<Figure size 1800x3600 with 0 Axes>
```

In [38]:
```
import cv2
```

In [39]:
```
import os
os.getcwd()
```

Out[39]:
```
'D:\\malware'
```

In [40]:
```
%cd D:\\malware
```

```
D:\malware
```

In [41]:
```
image_fts = np.zeros((10868, 200))
```

In [42]:
```
from tqdm import tqdm
for i, asmfile in tqdm(enumerate(os.listdir("asmFiles"))):
    img = cv2.imread("asmImage/" + asmfile.split('.')[0] + '.png')
    img_arr = img.flatten()[:200]
    image_fts[i, :] += img_arr
```

```
10868it [27:24,  6.61it/s]
```

In [43]:
```
#extracting the column names of first 200 pixels.
imgfeatures_name = []
for i in range(200):
    imgfeatures_name.append('pix' + str(i))
img_new = pd.DataFrame(normalize(image_fts, axis = 0), columns = imgfeatures_name)
img_new['ID'] = result.ID
```

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

Out[44]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix191	pix192	pix193	pix194
0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009593
1	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009593
2	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009593
3	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009593
4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009593

5 rows × 201 columns

In [45]:
```
from sklearn.externals import joblib
joblib.dump(img_new, 'img_fin')
```

Out[45]: ['img_fin']

In [46]:
```
os.getcwd()
```

Out[46]: 'D:\\malware'

In [47]:
```
%cd D:\\malware
```

```
D:\malware
```

In [48]:
```
from sklearn.externals import joblib
joblib.load('img_fin')
```

Out[48]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix191	pix192	pix193	pix
0	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
1	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
2	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
3	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
5	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
6	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
7	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
8	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
9	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
10	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
11	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
12	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
13	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
14	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
15	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
16	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
17	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
18	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
19	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
20	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
21	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
22	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
23	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix191	pix192	pix193	pix
24	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
25	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
26	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.009593	0.009593	0.009593	0.009
27	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
28	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
29	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10838	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10839	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10840	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10841	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10842	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10843	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10844	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10845	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10846	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10847	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10848	0.016401	0.016401	0.016401	0.011758	0.011758	0.011758	0.013183	0.013183	0.013183	0.005585	...	0.009593	0.009593	0.009593	0.009
10849	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10850	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10851	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10852	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10853	0.016401	0.016401	0.016401	0.011758	0.011758	0.011758	0.013183	0.013183	0.013183	0.005585	...	0.009593	0.009593	0.009593	0.009
10854	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10855	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10856	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix191	pix192	pix193	pix
10857	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10858	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10859	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10860	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10861	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10862	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10863	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10864	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10865	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10866	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009
10867	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.009593	0.009593	0.009593	0.009

10868 rows × 201 columns

Combining byte files unigram, asm features & asm image feature

In [84]:

```
comb_data = pd.concat([result, result_asm, img_new], axis = 1, join = 'inner')
comb_data.head()
```

Out[84]:

	ID	0	1	2	3	4	5	6	7	8	...	pix191	pix192	pi
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.009593	0.009593	0.00
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.009593	0.009593	0.00
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.009593	0.009593	0.00
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.009593	0.009593	0.00
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.009593	0.009593	0.00

5 rows × 514 columns

In [85]:
```
comb_data_1 = comb_data.drop(['ID', 'Class'], axis=1)
comb_data_1.head()
```

Out[85]:

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	pix193
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.009593	0.009593	0.009593	0.009593
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.009593	0.009593	0.009593	0.009593
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.009593	0.009593	0.009593	0.009593
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.009593	0.009593	0.009593	0.009593
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.009593	0.009593	0.009593	0.009593

5 rows × 509 columns

In [86]:
```
type(comb_data_1)
```

Out[86]:
```
pandas.core.frame.DataFrame
```

In [87]:
```
comb_data_1.fillna(0)
```

Out[87]:

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	p
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.009593	0.009593	0.009593	0.0
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.009593	0.009593	0.009593	0.0
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.009593	0.009593	0.009593	0.0
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.009593	0.009593	0.009593	0.0
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.009593	0.009593	0.009593	0.0
5	0.037152	0.000583	0.000189	0.000179	0.000214	0.000183	0.000171	0.000275	0.000291	0.000401	...	0.009593	0.009593	0.009593	0.0
6	0.021138	0.031231	0.007688	0.007696	0.151561	0.008529	0.007359	0.011977	0.009622	0.000243	...	0.009593	0.009593	0.009593	0.0
7	0.009431	0.001943	0.000416	0.000578	0.000700	0.000328	0.000233	0.000595	0.003302	0.000735	...	0.009593	0.009593	0.009593	0.0
8	0.014470	0.000605	0.000173	0.000221	0.000252	0.000187	0.000217	0.000794	0.000307	0.000416	...	0.009593	0.009593	0.009593	0.0
9	0.038002	0.079385	0.033004	0.029590	0.007601	0.005878	0.006948	0.011057	0.011021	0.000229	...	0.009593	0.009593	0.009593	0.0
10	0.020150	0.016701	0.006024	0.005192	0.006263	0.005336	0.007957	0.009828	0.010777	0.008890	...	0.009593	0.009593	0.009593	0.0
11	0.005401	0.003262	0.001174	0.001192	0.001278	0.001200	0.001187	0.002012	0.002157	0.002171	...	0.009593	0.009593	0.009593	0.0
12	0.006165	0.011838	0.003406	0.003303	0.003813	0.003525	0.003579	0.005778	0.005439	0.006801	...	0.009593	0.009593	0.009593	0.0
13	0.002770	0.004292	0.001694	0.001678	0.001920	0.001754	0.001734	0.002741	0.002841	0.003425	...	0.009593	0.009593	0.009593	0.0
14	0.017478	0.002788	0.000682	0.000644	0.001207	0.000620	0.000542	0.000695	0.002141	0.000606	...	0.009593	0.009593	0.009593	0.0
15	0.011195	0.001021	0.000169	0.000255	0.000180	0.000191	0.000112	0.000196	0.000214	0.000220	...	0.009593	0.009593	0.009593	0.0
16	0.014740	0.002065	0.000526	0.000363	0.000904	0.000581	0.000520	0.000602	0.001306	0.000550	...	0.009593	0.009593	0.009593	0.0
17	0.003307	0.005989	0.001709	0.001661	0.001894	0.001756	0.001766	0.002884	0.002751	0.003423	...	0.009593	0.009593	0.009593	0.0
18	0.004991	0.007705	0.001753	0.001775	0.001996	0.001816	0.001853	0.003046	0.002832	0.003584	...	0.009593	0.009593	0.009593	0.0
19	0.022479	0.014481	0.004830	0.004235	0.007200	0.004278	0.005007	0.006456	0.009336	0.000609	...	0.009593	0.009593	0.009593	0.0
20	0.008479	0.001344	0.000386	0.000354	0.000509	0.000262	0.000209	0.000469	0.000744	0.000495	...	0.009593	0.009593	0.009593	0.0
21	0.006622	0.001477	0.000315	0.000346	0.000477	0.000213	0.000269	0.000280	0.001038	0.000202	...	0.009593	0.009593	0.009593	0.0
22	0.033997	0.071684	0.026956	0.180296	0.005323	0.004026	0.003538	0.006098	0.006134	0.000292	...	0.009593	0.009593	0.009593	0.0
23	0.075334	0.004912	0.001476	0.001916	0.001573	0.001509	0.001347	0.002095	0.001831	0.002558	...	0.009593	0.009593	0.009593	0.0

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	
24	0.034741	0.009073	0.001749	0.001358	0.003050	0.001032	0.001198	0.002728	0.007386	0.001685	...	0.009593	0.009593	0.009593	0.0
25	0.020717	0.011554	0.005100	0.003957	0.005683	0.005060	0.003948	0.010136	0.008203	0.000587	...	0.009593	0.009593	0.009593	0.0
26	0.029327	0.033820	0.015668	0.016655	0.016494	0.015194	0.014941	0.023670	0.024791	0.021647	...	0.009593	0.009593	0.009593	0.0
27	0.004689	0.007856	0.001781	0.001804	0.002010	0.001855	0.001864	0.003064	0.002846	0.003574	...	0.009593	0.009593	0.009593	0.0
28	0.005003	0.007746	0.001813	0.001804	0.002103	0.001902	0.001823	0.002947	0.002980	0.003643	...	0.009593	0.009593	0.009593	0.0
29	0.003720	0.005903	0.001684	0.001671	0.001906	0.001806	0.001783	0.002924	0.002746	0.003454	...	0.009593	0.009593	0.009593	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10838	0.001845	0.000241	0.000074	0.000109	0.000113	0.000077	0.000072	0.000116	0.000129	0.000118	...	0.009593	0.009593	0.009593	0.0
10839	0.001664	0.000507	0.000181	0.000284	0.000232	0.000201	0.000197	0.000310	0.000300	0.000354	...	0.009593	0.009593	0.009593	0.0
10840	0.001371	0.000607	0.000201	0.000225	0.000257	0.000205	0.000192	0.000335	0.000359	0.000405	...	0.009593	0.009593	0.009593	0.0
10841	0.001458	0.000593	0.000256	0.000245	0.000284	0.000256	0.000219	0.000377	0.000369	0.000489	...	0.009593	0.009593	0.009593	0.0
10842	0.001664	0.000259	0.000078	0.000180	0.000122	0.000081	0.000076	0.000138	0.000149	0.000167	...	0.009593	0.009593	0.009593	0.0
10843	0.001412	0.000607	0.000241	0.000252	0.000291	0.000227	0.000214	0.000401	0.000363	0.000447	...	0.009593	0.009593	0.009593	0.0
10844	0.001443	0.000577	0.000235	0.000270	0.000289	0.000226	0.000215	0.000375	0.000351	0.000430	...	0.009593	0.009593	0.009593	0.0
10845	0.002417	0.000624	0.000242	0.000262	0.000320	0.000183	0.000254	0.000295	0.000413	0.000380	...	0.009593	0.009593	0.009593	0.0
10846	0.001632	0.001519	0.000619	0.000647	0.000871	0.000604	0.000594	0.000931	0.000898	0.001148	...	0.009593	0.009593	0.009593	0.0
10847	0.001568	0.001768	0.000675	0.000667	0.000730	0.000675	0.000669	0.001056	0.001019	0.001329	...	0.009593	0.009593	0.009593	0.0
10848	0.000227	0.000604	0.000224	0.000229	0.000248	0.000246	0.000228	0.000412	0.000344	0.000473	...	0.009593	0.009593	0.009593	0.0
10849	0.001754	0.000538	0.000198	0.000259	0.000315	0.000192	0.000194	0.000333	0.000319	0.000345	...	0.009593	0.009593	0.009593	0.0
10850	0.002851	0.001509	0.000669	0.000669	0.000752	0.000598	0.000688	0.000901	0.001073	0.001239	...	0.009593	0.009593	0.009593	0.0
10851	0.001276	0.000210	0.000058	0.000120	0.000282	0.000050	0.000052	0.000084	0.000101	0.000091	...	0.009593	0.009593	0.009593	0.0
10852	0.000801	0.000432	0.000187	0.000192	0.000226	0.000164	0.000175	0.000323	0.000311	0.000307	...	0.009593	0.009593	0.009593	0.0
10853	0.000085	0.000156	0.000058	0.000047	0.000062	0.000055	0.000048	0.000093	0.000070	0.000096	...	0.009593	0.009593	0.009593	0.0
10854	0.002146	0.000764	0.000314	0.000319	0.000378	0.000267	0.000311	0.000441	0.000507	0.000470	...	0.009593	0.009593	0.009593	0.0
10855	0.001580	0.000496	0.000223	0.000252	0.000293	0.000231	0.000209	0.000334	0.000319	0.000435	...	0.009593	0.009593	0.009593	0.0
10856	0.001261	0.000653	0.000223	0.000231	0.000257	0.000192	0.000189	0.000317	0.000349	0.000386	...	0.009593	0.009593	0.009593	0.0

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	
10857	0.001984	0.000596	0.000202	0.000235	0.000318	0.000231	0.000227	0.000352	0.000376	0.000399	...	0.009593	0.009593	0.009593	0.0
10858	0.002255	0.001632	0.000618	0.000646	0.000764	0.000627	0.000636	0.000964	0.000989	0.001325	...	0.009593	0.009593	0.009593	0.0
10859	0.000980	0.000436	0.000148	0.000230	0.000343	0.000157	0.000148	0.000250	0.000269	0.000308	...	0.009593	0.009593	0.009593	0.0
10860	0.002489	0.000897	0.000256	0.000416	0.000329	0.000315	0.000259	0.000521	0.000424	0.000660	...	0.009593	0.009593	0.009593	0.0
10861	0.002493	0.000757	0.000274	0.000367	0.000340	0.000278	0.000283	0.000445	0.000459	0.000500	...	0.009593	0.009593	0.009593	0.0
10862	0.001771	0.001787	0.000670	0.000697	0.000729	0.000653	0.000639	0.001049	0.001039	0.001229	...	0.009593	0.009593	0.009593	0.0
10863	0.002300	0.001657	0.000596	0.000659	0.000758	0.000656	0.000644	0.001036	0.001022	0.001257	...	0.009593	0.009593	0.009593	0.0
10864	0.001324	0.000420	0.000138	0.000158	0.000168	0.000121	0.000114	0.000204	0.000229	0.000258	...	0.009593	0.009593	0.009593	0.0
10865	0.002476	0.000311	0.000150	0.000174	0.000192	0.000088	0.000140	0.000135	0.000232	0.000172	...	0.009593	0.009593	0.009593	0.0
10866	0.001588	0.000615	0.000252	0.000273	0.000313	0.000221	0.000243	0.000375	0.000360	0.000454	...	0.009593	0.009593	0.009593	0.0
10867	0.001543	0.000525	0.000214	0.000233	0.000303	0.000226	0.000222	0.000343	0.000355	0.000376	...	0.009593	0.009593	0.009593	0.0

10868 rows × 509 columns

In [88]:
```
np.isnan(comb_data_1)
```

Out[88]:

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	pix193	pix194	pix195	pix196	pix197	pix
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
5	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
6	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
7	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
8	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
9	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
11	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
12	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
13	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
14	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
15	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
16	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
17	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
18	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
19	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
20	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
21	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
22	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
23	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	pix193	pix194	pix195	pix196	pix197	pi
24	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
25	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
26	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
27	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
28	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
29	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10838	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10839	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10840	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10841	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10842	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10843	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10844	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10845	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10846	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10847	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10848	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10849	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10850	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10851	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10852	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10853	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10854	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10855	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10856	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	pix193	pix194	pix195	pix196	pix197	pi
10857	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10858	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10859	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10860	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10861	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10862	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10863	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10864	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10865	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10866	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F
10867	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	F

10868 rows × 509 columns

In [89]:
```
np.where(np.isnan(comb_data_1))
```

Out[89]:
```
(array([], dtype=int64), array([], dtype=int64))
```

In [90]:
```
comb_data_1 = np.nan_to_num(comb_data_1)
```

Train & Test Split

In [91]:
```
result_y = data_y
```

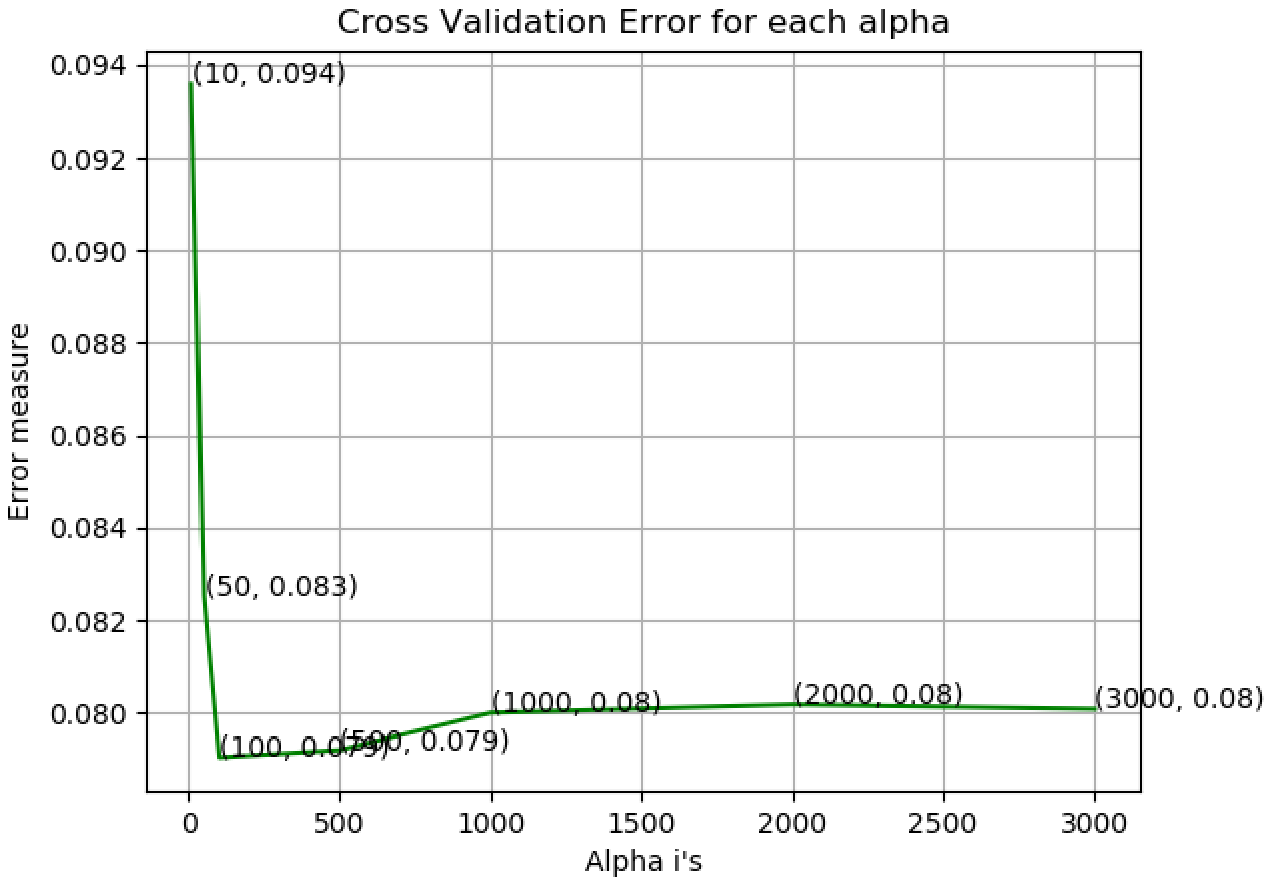
In [92]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(comb_data_1, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

Random Forest classifier on byte files unigram, asm unigram features & asm image feature

In [93]:
```
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-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()
```

```
log_loss for c =  10 is 0.09358462413319721
log_loss for c =  50 is 0.08253160971468469
log_loss for c =  100 is 0.07904491338488688
log_loss for c =  500 is 0.0792040133741543
log_loss for c =  1000 is 0.08000360801340127
log_loss for c =  2000 is 0.08017864546787751
log_loss for c =  3000 is 0.08008285980316453
```

Figure 18


Cross Validation Error for each alpha
Error measure
Alpha i's
0.094
0.092
0.090
0.088
0.086
0.084
0.082
0.080
0
500
1000
1500
2000
2500
3000
(10, 0.094)
(50, 0.083)
(100, 0.079)
(500, 0.079)
(1000, 0.08)
(2000, 0.08)
(3000, 0.08)

In [94]:
```
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 ('log loss for train data',(log_loss(y_train_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_merge)
print ('log loss for cv data',(log_loss(y_cv_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_merge)
print ('log loss for test data',(log_loss(y_test_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
log loss for train data 0.024819755576891478
log loss for cv data 0.07904491338488688
log loss for test data 0.07168657134325582
Number of misclassified points  1.609935602575897
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

HyperParameter Tuned XGBoost on Byte files unigram, ASM features & ASM image feature

In [95]:
```
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  5.3min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  8.5min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 11.3min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.7min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 16.4min finished
```

Out[95]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

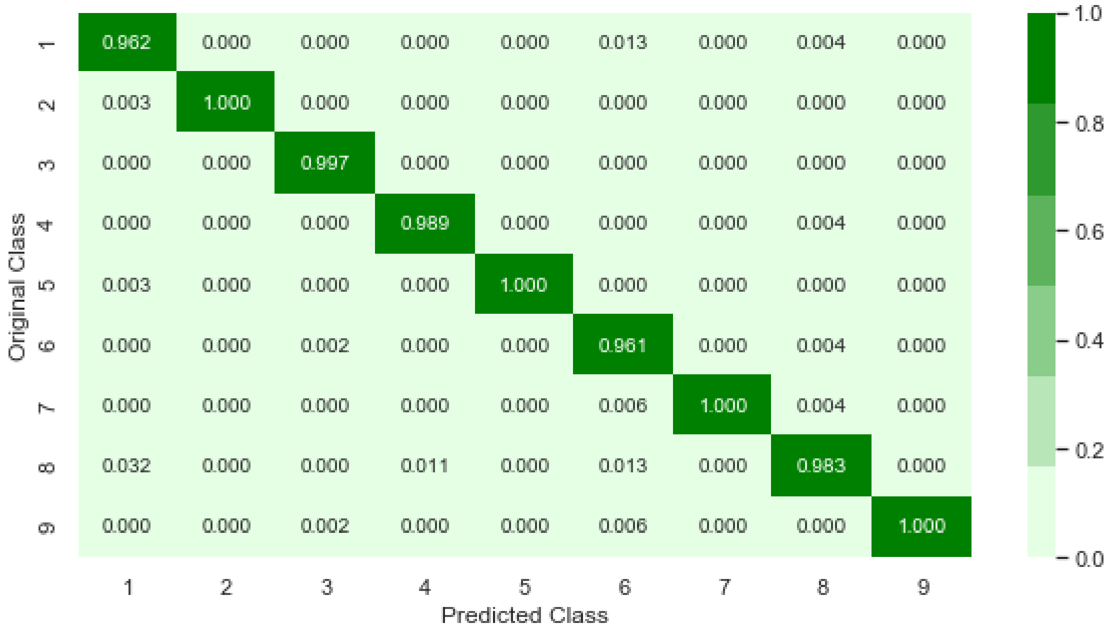
In [96]:
```
print(random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 0.3}
```

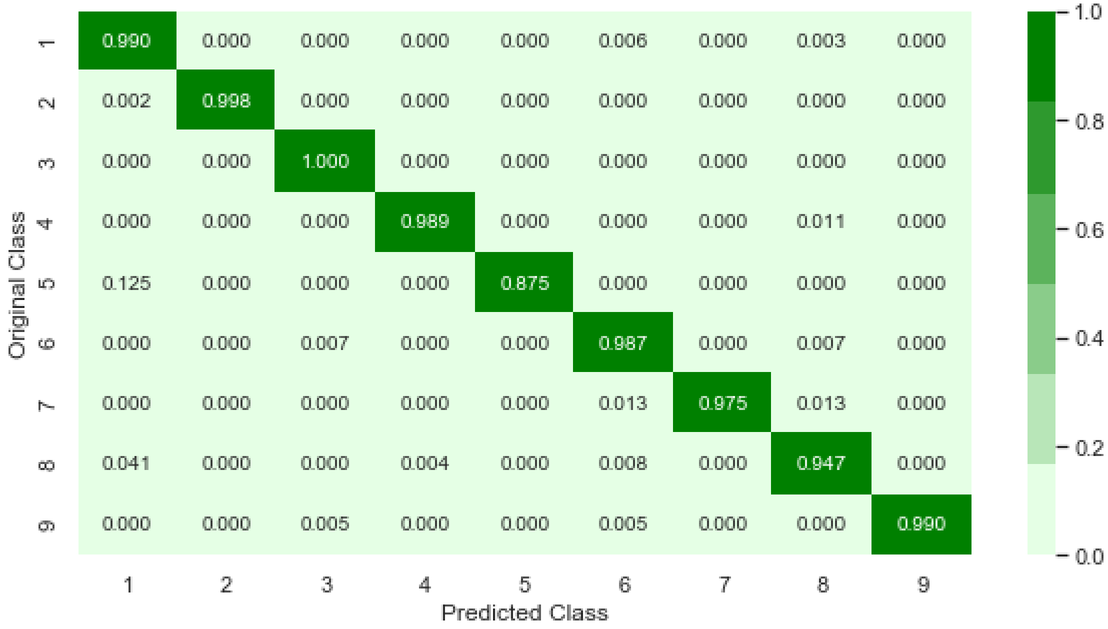
In [100]:
```
x_cfl=XGBClassifier(n_estimators=1000, learning_rate=0.03, colsample_bytree=0.3, max_depth=3, subsample=1, n_jobs=-1)
x_cfl.fit(X_train_merge, y_train_merge)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_merge, y_train_merge)

predict_y = c_cfl.predict_proba(X_train_merge)
print ('train loss',log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))

plt.close()
plot_confusion_matrix(y_test_merge,c_cfl.predict(X_test_merge))
```

```
train loss 0.017555537143338063
cv loss 0.049817892852966346
test loss 0.06643020013696387
Number of misclassified points  1.1499540018399264
-------------------------------------------------- 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.]
```

Combining byte files bi-gram, asm features & asm image feature

In [97]:
```
comb_data = pd.concat([byte_bi_df_1, result_asm, img_new], axis = 1, join = 'inner')
comb_data.head()
```

Out[97]:

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	pix191	pix192	pix193	pix194
0	0.005295	0.000067	0.008667	0.008217	0.000510	0.000835	0.000307	0.000087	0.000033	0.000144	...	0.009593	0.009593	0.009593	0.009593
1	0.000000	0.000000	0.000000	0.000000	0.000510	0.000000	0.000077	0.000362	0.000265	0.052763	...	0.009593	0.009593	0.009593	0.009593
2	0.005295	0.000056	0.003250	0.005135	0.151488	0.000626	0.000384	0.000050	0.000099	0.000108	...	0.009593	0.009593	0.009593	0.009593
3	0.001059	0.000011	0.002167	0.000000	0.000510	0.000000	0.000537	0.000000	0.000066	0.000000	...	0.009593	0.009593	0.009593	0.009593
4	0.000000	0.000011	0.000000	0.001027	0.001020	0.000104	0.000230	0.000000	0.000000	0.000000	...	0.009593	0.009593	0.009593	0.009593

5 rows × 554 columns

In [98]:
```
comb_data_2 = comb_data.drop(['ID', 'Class'], axis=1)
comb_data_2.head()
```

Out[98]:

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	pix190	pix191	pix192	pix193
0	0.005295	0.000067	0.008667	0.008217	0.000510	0.000835	0.000307	0.000087	0.000033	0.000144	...	0.009593	0.009593	0.009593	0.009593
1	0.000000	0.000000	0.000000	0.000000	0.000510	0.000000	0.000077	0.000362	0.000265	0.052763	...	0.009593	0.009593	0.009593	0.009593
2	0.005295	0.000056	0.003250	0.005135	0.151488	0.000626	0.000384	0.000050	0.000099	0.000108	...	0.009593	0.009593	0.009593	0.009593
3	0.001059	0.000011	0.002167	0.000000	0.000510	0.000000	0.000537	0.000000	0.000066	0.000000	...	0.009593	0.009593	0.009593	0.009593
4	0.000000	0.000011	0.000000	0.001027	0.001020	0.000104	0.000230	0.000000	0.000000	0.000000	...	0.009593	0.009593	0.009593	0.009593

5 rows × 551 columns

In [99]:
```
comb_data_2 = np.nan_to_num(comb_data_2)
```

Train & Test Split

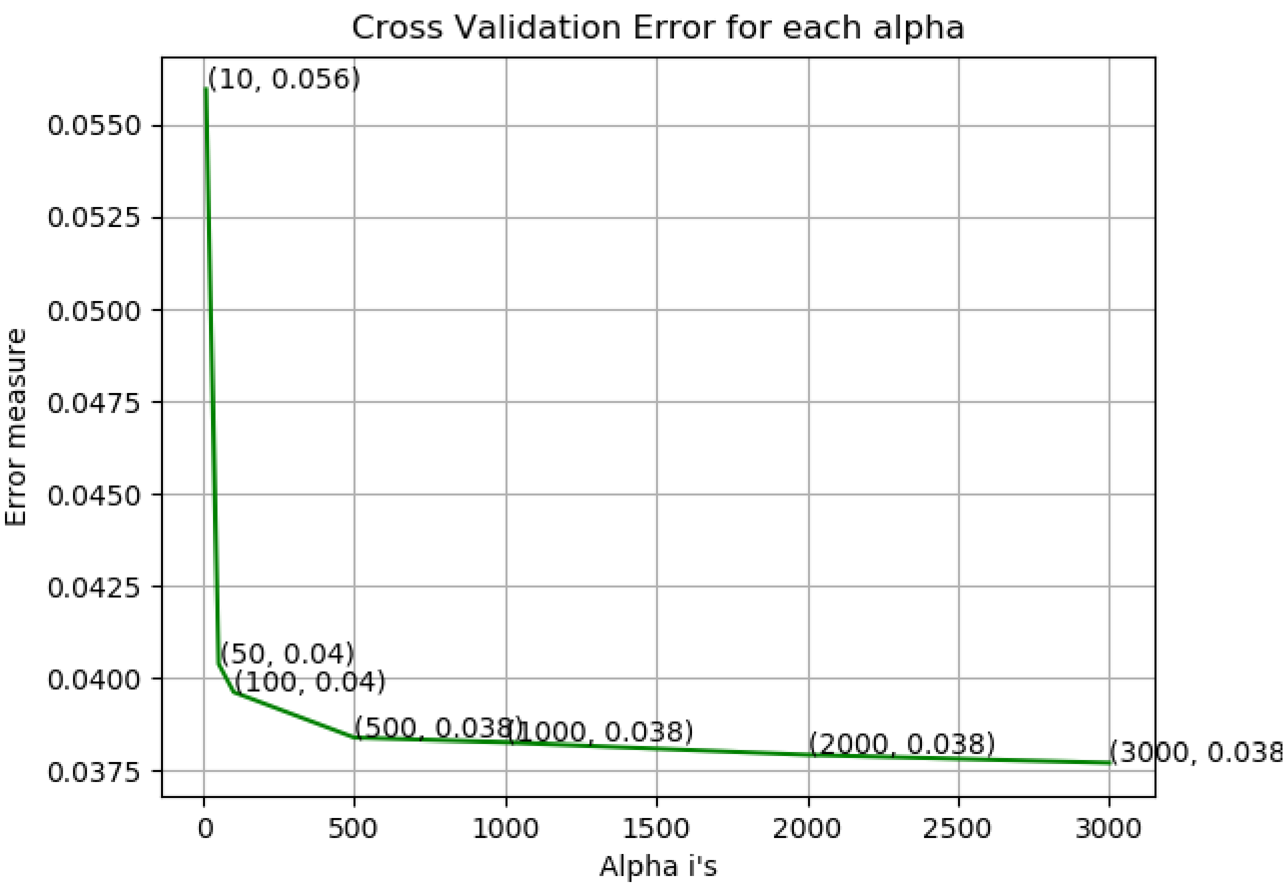
In [100]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(comb_data_2, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

Random Forest classifier on byte files bi-gram, asm features & asm image feature

In [101]:
```
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-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()
```

```
log_loss for c =  10 is 0.055944327234702865
log_loss for c =  50 is 0.040390245593763065
log_loss for c =  100 is 0.03962488636566595
log_loss for c =  500 is 0.038392230651333265
log_loss for c =  1000 is 0.03827118455193932
log_loss for c =  2000 is 0.03793790929235758
log_loss for c =  3000 is 0.037721819688108546
```

Figure 22


Cross Validation Error for each alpha
(10, 0.056)
(50, 0.04)
(100, 0.04)
(500, 0.038)
(1000, 0.038)
(2000, 0.038)
(3000, 0.038
0.0550
0.0525
0.0500
0.0475
0.0450
0.0425
0.0400
0.0375
Error measure
0
500
1000
1500
2000
2500
3000
Alpha i's

In [102]:

```
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 ('log loss for train data',(log_loss(y_train_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_merge)
print ('log loss for cv data',(log_loss(y_cv_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_merge)
print ('log loss for test data',(log_loss(y_test_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
log loss for train data 0.015077519806199778
log loss for cv data 0.037721819688108546
log loss for test data 0.035607556170l3001
Number of misclassified points  0.8739650413983441
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

HyperParameter Tuned XGBoost on Byte files bi-gram, ASM features & ASM image feature

In [103]:
```
x_cfl=XGBClassifier()

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:   37.8s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  8.1min remaining:   53.8s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  9.1min finished
```

Out[103]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [104]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 10, 'learning_rate': 0.15, 'colsample_bytree': 0.3}
```

In [105]:
```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.15, colsample_bytree=0.3, max_depth=10, subsample=1, n_jobs=-1)
x_cfl.fit(X_train_merge, y_train_merge)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_merge, y_train_merge)

predict_y = c_cfl.predict_proba(X_train_merge)
print ('train loss',log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))
```

```
train loss 0.01429945926818971
cv loss 0.043889325406506634
test loss 0.04778832106704075
```

In [106]:
```
plt.close()
plot_confusion_matrix(y_test_merge,c_cfl.predict(X_test_merge))
```

```
Number of misclassified points  0.8739650413983441
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

Combining byte files bi-gram & asm image feature

In [107]:
```
comb_data_3 = pd.concat([byte_bi_df_1, img_new], axis = 1, join = 'inner')
comb_data_3.head()
```

Out[107]:

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	pix191	pix192	pix193	pix194
0	0.005295	0.000067	0.008667	0.008217	0.000510	0.000835	0.000307	0.000087	0.000033	0.000144	...	0.009593	0.009593	0.009593	0.009593
1	0.000000	0.000000	0.000000	0.000000	0.000510	0.000000	0.000077	0.000362	0.000265	0.052763	...	0.009593	0.009593	0.009593	0.009593
2	0.005295	0.000056	0.003250	0.005135	0.151488	0.000626	0.000384	0.000050	0.000099	0.000108	...	0.009593	0.009593	0.009593	0.009593
3	0.001059	0.000011	0.002167	0.000000	0.000510	0.000000	0.000537	0.000000	0.000066	0.000000	...	0.009593	0.009593	0.009593	0.009593
4	0.000000	0.000011	0.000000	0.001027	0.001020	0.000104	0.000230	0.000000	0.000000	0.000000	...	0.009593	0.009593	0.009593	0.009593

5 rows × 501 columns

In [108]:
```
comb_data_3 = comb_data_3.drop(['ID'], axis=1)
comb_data_3.head()
```

Out[108]:

	f1 86	1f 71	f1 b3	a2 ea	cb 1e	ee 58	31 89	44 8e	0a 2d	12 0b	...	pix190	pix191	pix192	pix193
0	0.005295	0.000067	0.008667	0.008217	0.000510	0.000835	0.000307	0.000087	0.000033	0.000144	...	0.009593	0.009593	0.009593	0.009593
1	0.000000	0.000000	0.000000	0.000000	0.000510	0.000000	0.000077	0.000362	0.000265	0.052763	...	0.009593	0.009593	0.009593	0.009593
2	0.005295	0.000056	0.003250	0.005135	0.151488	0.000626	0.000384	0.000050	0.000099	0.000108	...	0.009593	0.009593	0.009593	0.009593
3	0.001059	0.000011	0.002167	0.000000	0.000510	0.000000	0.000537	0.000000	0.000066	0.000000	...	0.009593	0.009593	0.009593	0.009593
4	0.000000	0.000011	0.000000	0.001027	0.001020	0.000104	0.000230	0.000000	0.000000	0.000000	...	0.009593	0.009593	0.009593	0.009593

5 rows × 500 columns

Train & Test Split

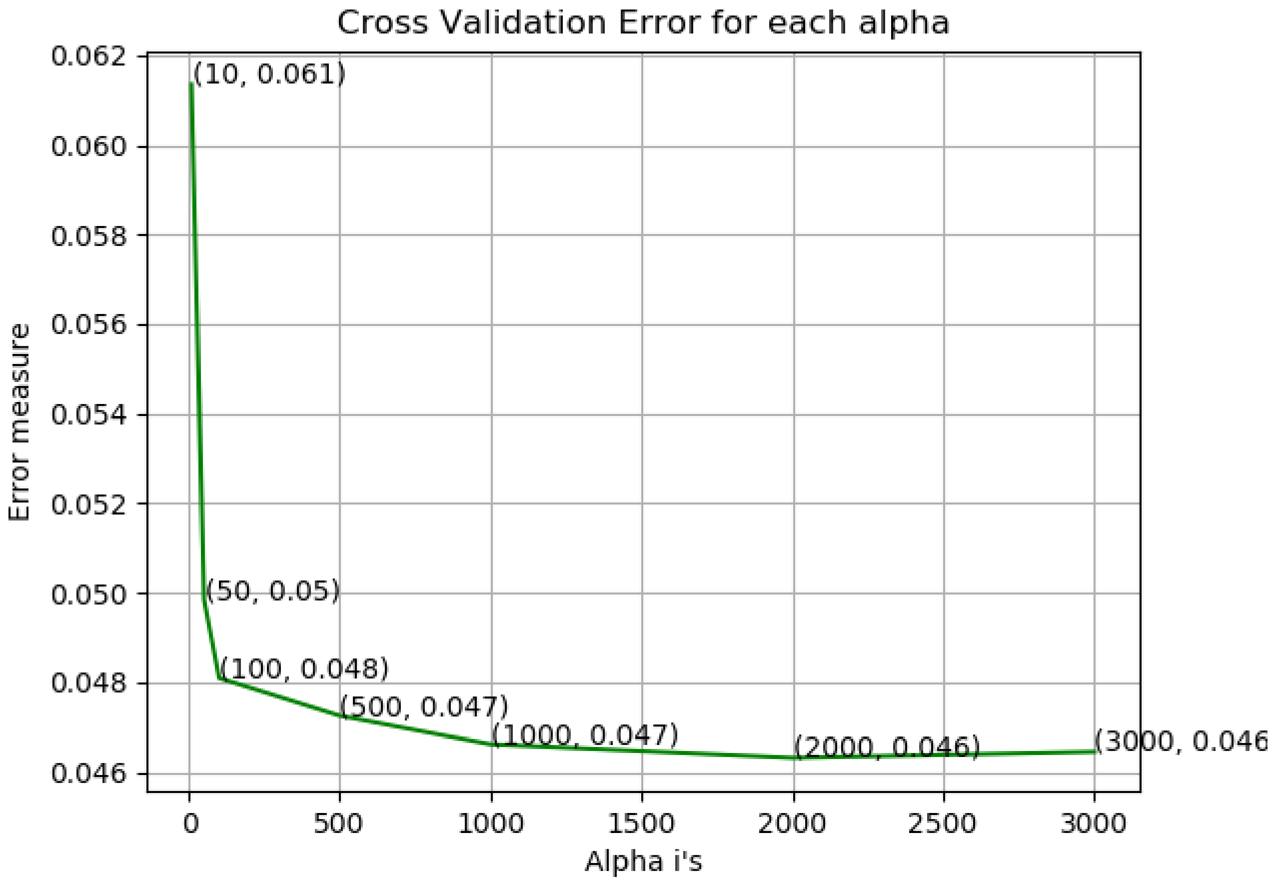
In [109]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(comb_data_3, result_y,stratify=result_y,test_size=0.30)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.30)
```

Random Forest Classifier on byte bi-grams Files & ASM Image Feature

In [110]:
```
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-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()
```

```
log_loss for c =  10 is 0.06134983721742508
log_loss for c =  50 is 0.04984443751257351
log_loss for c =  100 is 0.04809199786436746
log_loss for c =  500 is 0.047252918158990326
log_loss for c =  1000 is 0.046618925102466575
log_loss for c =  2000 is 0.04631974297558685
log_loss for c =  3000 is 0.046452960130821785
```

Figure 29


Cross Validation Error for each alpha
Error measure
Alpha i's
0.062
0.060
0.058
0.056
0.054
0.052
0.050
0.048
0.046
0
500
1000
1500
2000
2500
3000
(10, 0.061)
(50, 0.05)
(100, 0.048)
(500, 0.047)
(1000, 0.047)
(2000, 0.046)
(3000, 0.046

In [111]:
```
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 ('log loss for train data',(log_loss(y_train_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_merge)
print ('log loss for cv data',(log_loss(y_cv_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_merge)
print ('log loss for test data',(log_loss(y_test_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

-

HyperParameter Tuned XGBoost on byte bi-grams Files & ASM Image Feature

In [112]:
```
prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  7.8min remaining:   51.8s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  8.8min finished
```

Out[112]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.3, gamma=0,
       learning_rate=0.15, max_delta_step=0, max_depth=10,
       min_child_weight=1, missing=None, n_estimators=2000, n_jobs=-1,
       nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [113]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 0.3, 'n_estimators': 2000, 'max_depth': 10, 'learning_rate': 0.15, 'colsample_bytree': 1}
```

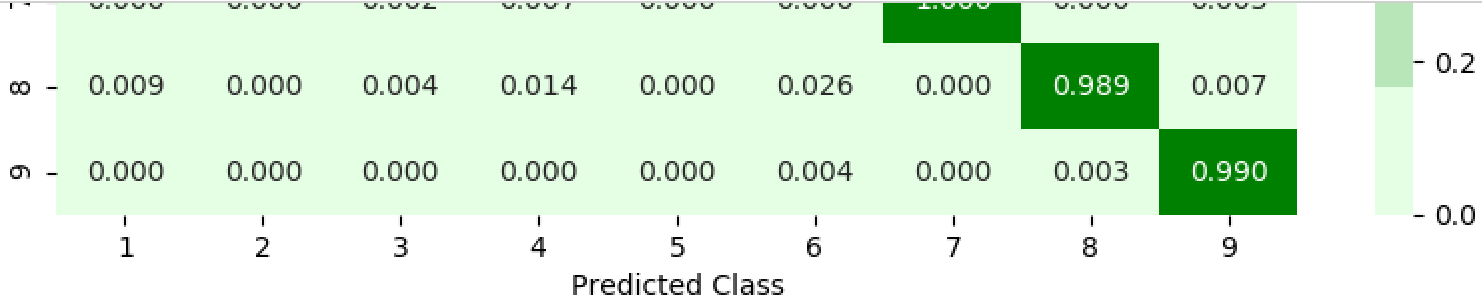
In [114]:

```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.1, colsample_bytree=1, max_depth=10, subsample=0.3, n_jobs=-1)
x_cfl.fit(X_train_merge, y_train_merge)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_merge, y_train_merge)

predict_y = c_cfl.predict_proba(X_train_merge)
print ('train loss',log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))
```

```
train loss 0.016400968681616872
cv loss 0.04860737454938231
test loss 0.04776613242624286
```

In [115]:

```
plt.close()
plot_confusion_matrix(y_test_merge,c_cfl.predict(X_test_merge))
```




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

Combining all the Advanced Features

Concatenating all the Features

In [116]:
```
adv_data = pd.concat([result_x, byte_bi_df_1, asm_bi_df, asm_tri_df, asm_4_df, img_new], axis = 1, join = 'inner')
adv_data.head()
```

Out[116]:

	0	1	2	3	4	5	6	7	8	9	...	pix191	pix192	pix193	pix194
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.009593	0.009593	0.009593	0.009593
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.009593	0.009593	0.009593	0.009593
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.009593	0.009593	0.009593	0.009593
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.009593	0.009593	0.009593	0.009593
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.009593	0.009593	0.009593	0.009593

5 rows × 1707 columns

In [117]:
```
adv_data = adv_data.drop(['ID'], axis=1)
adv_data.head()
```

Out[117]:

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	pix193
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.009593	0.009593	0.009593	0.009593
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.009593	0.009593	0.009593	0.009593
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.009593	0.009593	0.009593	0.009593
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.009593	0.009593	0.009593	0.009593
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.009593	0.009593	0.009593	0.009593

5 rows × 1706 columns

Train & Test Split

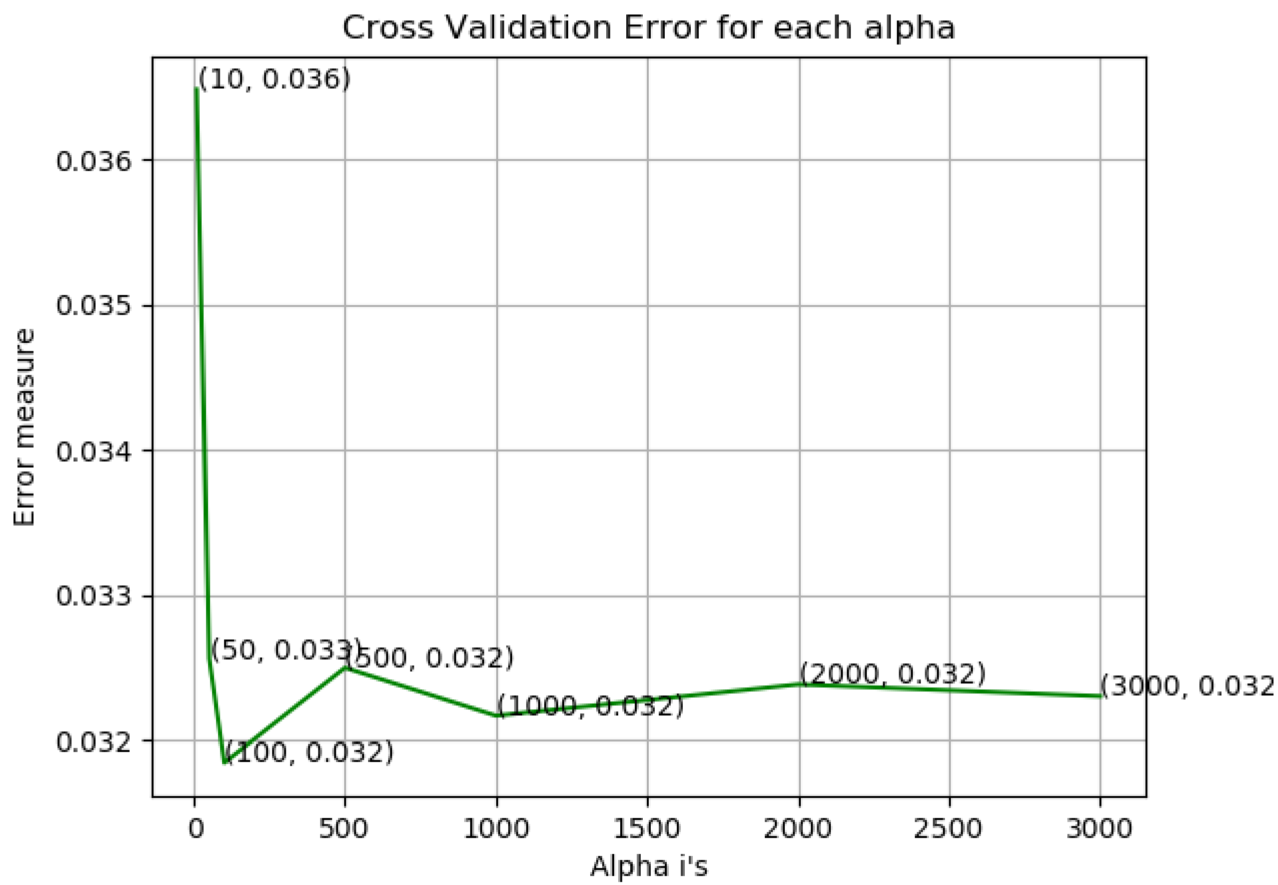
In [118]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(adv_data, result_y,stratify=result_y,test_size=0.30)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.30)
```

Random Forest classifier on all advanced features

In [119]:
```
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-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()
```

```
log_loss for c =  10 is 0.03648352226602319
log_loss for c =  50 is 0.03256154225867737
log_loss for c =  100 is 0.03184520269188538
log_loss for c =  500 is 0.03249782536338217
log_loss for c =  1000 is 0.032167328408604916
log_loss for c =  2000 is 0.032382960498833296
log_loss for c =  3000 is 0.03230170029356027
```

Figure 36


Cross Validation Error for each alpha
(10, 0.036)
(50, 0.033)
(100, 0.032)
(500, 0.032)
(1000, 0.032)
(2000, 0.032)
(3000, 0.032
0.036
0.035
0.034
0.033
0.032
Error measure
0
500
1000
1500
2000
2500
3000
Alpha i's

In [120]:
```
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 ('log loss for train data',(log_loss(y_train_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_merge)
print ('log loss for cv data',(log_loss(y_cv_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_merge)
print ('log loss for test data',(log_loss(y_test_merge, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
log loss for train data 0.014884465260867705
log loss for cv data 0.03184520269188538
log loss for test data 0.03762214127356666
Number of misclassified points  0.5519779208831647
-------------------------------------------------- Confusion matrix -------------------------------------------------
-
```

HyperParameter Tuned XGBoost on advanced features

In [121]:
```
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':[3,5,7],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 32.1min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 40.3min remaining:  4.5min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 44.9min finished
```

Out[121]:
```
RandomizedSearchCV(cv=None, error_score='raise',
          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, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 100
0, 2000], 'max_depth': [3, 5, 7], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [122]:
```
print(random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.01, 'colsample_bytree': 1}
```

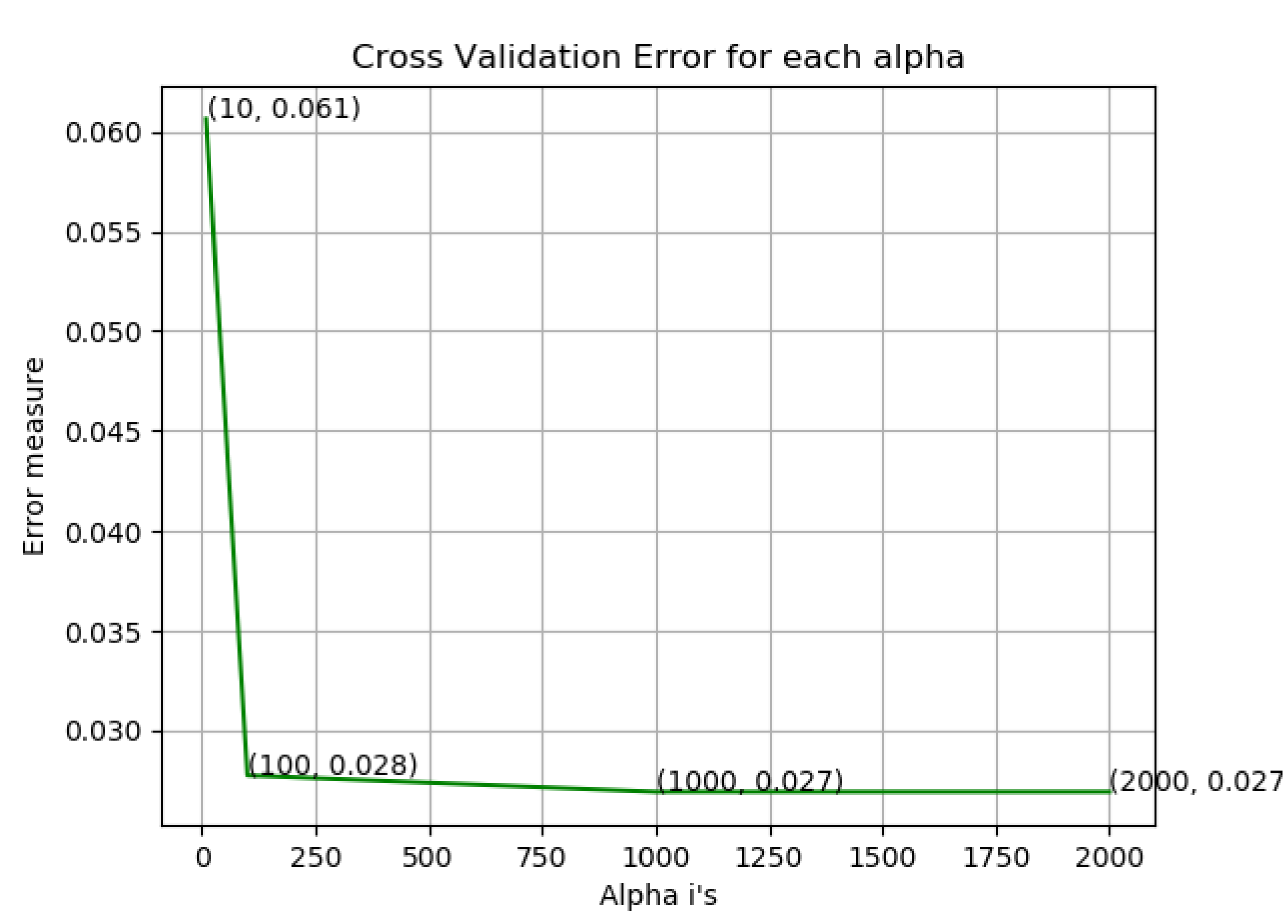
In [123]:
```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.01, colsample_bytree=1, max_depth=3, subsample=0.5, n_jobs=-1)
x_cfl.fit(X_train_merge, y_train_merge)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_merge, y_train_merge)

predict_y = c_cfl.predict_proba(X_train_merge)
print ('train loss',log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))

plt.close()
plot_confusion_matrix(y_test_merge,c_cfl.predict(X_test_merge))
```

```
train loss 0.01232078475973484
cv loss 0.028189404850559405
test loss 0.03037071551588438
Number of misclassified points  0.5519779208831647
-------------------------------------------------- Confusion matrix --------------------------------------------------
-
```

In [124]:
```
alpha=[10,100,1000,2000]
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-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()
```

```
log_loss for c =  10 is 0.060628142812704555
log_loss for c =  100 is 0.027764052794134443
log_loss for c =  1000 is 0.026939908450589377
log_loss for c =  2000 is 0.026939980272149756
```

Figure 43


Cross Validation Error for each alpha
(10, 0.061)
(100, 0.028)
(1000, 0.027)
(2000, 0.027
Error measure
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0
250
500
750
1000
1250
1500
1750
2000
Alpha i's

In [125]:
```
x_cfl=XGBClassifier(n_estimators=2000, n_jobs=-1)
x_cfl.fit(X_train_merge, y_train_merge)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_merge, y_train_merge)

predict_y = c_cfl.predict_proba(X_train_merge)
print ('train loss',log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print ('cv loss',log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print ('test loss',log_loss(y_test_merge, predict_y))

plt.close()
plot_confusion_matrix(y_test_merge,c_cfl.predict(X_test_merge))
```

```
train loss 0.012409313961248418
cv loss 0.026939980272149756
test loss 0.030838210768687875
Number of misclassified points  0.45998160073597055
-------------------------------------------------- Confusion matrix ---------------------------------------------------
-
```

In [ ]:
```
!pip install PrettyTable
```

In [126]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names  = ['Model', 'Features', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', '% Misclassified(Points)']

x.add_row(['Random Forest Classifier', 'Bytes-Bigram', '0.01', '0.03', '0.05', '1.24'])
x.add_row(['XGBoost Classifier', 'Bytes-Bigram', '0.01', '0.07', '0.04', '0.82'])
x.add_row(['Random Forest Classifier', 'Bytes-Bigram + ASM Image', '0.01', '0.05', '0.04', '0.64'])
x.add_row(['XGBoost Classifier', 'Bytes-Bigram + ASM Image', '0.01', '0.05', '0.03', '0.59'])
x.add_row(['Random Forest', 'ASM Files + ASM Images + Bytes Unigram', '0.02', '0.05', '0.07', '1.65'])
x.add_row(['Xgboost Classifier', 'ASM Files + ASM Images + Bytes Unigram', '0.02', '0.06', '0.06', '1.28'])
x.add_row(['Random Forest Classifier', 'ASM Files + ASM Images + Bytes Bi-gram', '0.01', '0.03', '0.04', '1.01'])
x.add_row(['Xgboost Classifier', 'ASM Files + ASM Images + Bytes Bi-gram', '0.01', '0.04', '0.03', '0.87'])
x.add_row(['Random Forest Classifier', 'All Features', '0.01', '0.02', '0.03', '0.55'])
x.add_row(['Xgboost Classifier', 'All Features', '0.01', '0.02', '0.03', '0.45'])
print(x)
```

```
+--------------------------+-----------------------------------------+----------------+-------------+---------------+-
------------------------+
|          Model           |                 Features                | Train Log Loss | CV Log Loss | Test Log Loss |
% Misclassified(Points) |
+--------------------------+-----------------------------------------+----------------+-------------+---------------+-
------------------------+
| Random Forest Classifier |               Bytes-Bigram              |      0.01      |     0.03    |      0.05     |
1.24          |
|    XGBoost Classifier    |               Bytes-Bigram              |      0.01      |     0.07    |      0.04     |
0.82          |
| Random Forest Classifier |         Bytes-Bigram + ASM Image        |      0.01      |     0.05    |      0.04     |
0.64          |
|    XGBoost Classifier    |         Bytes-Bigram + ASM Image        |      0.01      |     0.05    |      0.03     |
0.59          |
|      Random Forest       | ASM Files + ASM Images + Bytes Unigram |      0.02      |     0.05    |      0.07     |
1.65          |
|    Xgboost Classifier    | ASM Files + ASM Images + Bytes Unigram |      0.02      |     0.06    |      0.06     |
1.28          |
| Random Forest Classifier | ASM Files + ASM Images + Bytes Bi-gram |      0.01      |     0.03    |      0.04     |
1.01          |
|    Xgboost Classifier    | ASM Files + ASM Images + Bytes Bi-gram |      0.01      |     0.04    |      0.03     |
0.87          |
| Random Forest Classifier |               All Features              |      0.01      |     0.02    |      0.03     |
0.55          |
```

```
|    Xgboost Classifier    |              All Features              |     0.01     |    0.02     |     0.03     |
0.45          |
+--------------------------+----------------------------------------+--------------+-------------+--------------+-
------------------------+
```

References :-

In [ ]:
```
#https://www.youtube.com/watch?v=VLQTRlLGz5Y#t=13m11s
#https://towardsdatascience.com/malware-classification-using-machine-learning-7c648fb1da79
#https://github.com/mayank171986/Microsoft-Malware-Detection/blob/master/microsoft-malware-detection.ipynb
#https://github.com/matheen777/Microsoft_malware_detection/blob/master/MMD_final.ipynb
#https://www.kaggle.com/c/malware-classification/overview
##https://www.kaggle.com/c/microsoft-malware-pre diction/discussion/74639
```

In [ ]: