Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- Source : https://www.kaggle.com/c/malware-classification/data
- For every malware, we have two files
  1. .asm file (read more: https://www.reviversoft.com/file-extensions/asm)
  2. .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)
- Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:
- **Lots of Data for a single-box/computer.**
- There are total 10,868 .bytes files and 10,868 asm files total 21,736 files
- There are 9 types of malwares (9 classes) in our give data
- Types of Malware:
  1. Ramnit
  2. Lollipop
  3. Kelihos_ver3
  4. Vundo
  5. Simda
  6. Tracur
  7. Kelihos_ver1
  8. Obfuscator.ACY
  9. Gatak

2.1.2. Example Data Point

.asm file

```
.text:00401000                                  assume es:nothing, ss:n
othing, ds:_data,    fs:nothing, gs:nothing
.text:00401000 56                                  push    esi
.text:00401001 8D 44 24    08                          lea     eax, [es
p+8]
.text:00401005 50                                  push    eax
.text:00401006 8B F1                                  mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                          call    ??0ex
ception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08    BB 42 00                       mov     dword
ptr [esi],    offset off_42BB08
.text:00401013 8B C6                                  mov     eax, esi
.text:00401015 5E                                  pop     esi
.text:00401016 C2 04 00                                retn    4
.text:00401016                              ; ---------------------------
-----------------------------------------
.text:00401019 CC CC CC    CC CC CC CC                    align 10h
.text:00401020 C7 01 08    BB 42 00                       mov     dword
ptr [ecx],    offset off_42BB08
.text:00401026 E9 26 1C    00 00                          jmp     sub_4
02C51
.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_4
02C51
.text:0040103E F6 44 24    08 01                          test    byte
ptr    [esp+8], 1
.text:00401043 74 09                                  jz      short loc_401
04E
.text:00401045 56                                  push    esi
.text:00401046 E8 6C 1E    00 00                          call    ??3@Y
AXPAX@Z    ; operator delete(void *)
.text:0040104B 83 C4 04                             add     esp, 4
.text:0040104E
.text:0040104E                              loc_40104E:                ;
CODE XREF: .text:00401043 j
.text:0040104E 8B C6                                  mov     eax, esi
.text:00401050 5E                                  pop     esi
.text:00401051 C2 04 00                                retn    4
.text:00401051                              ; ---------------------------
-----------------------------------------
```

.bytes file

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

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

2.3. Train and Test Dataset

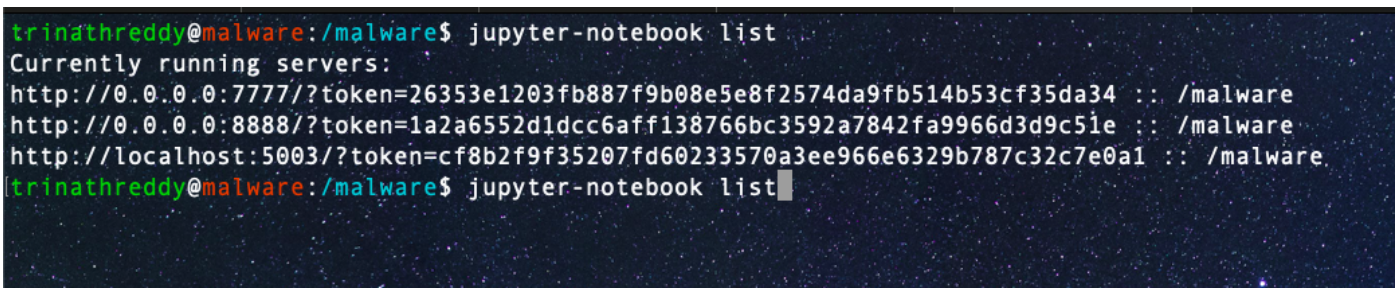
Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

2.4. Useful blogs, videos and reference papers

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

Accessing the notebook from server :

1. jupyter-notebook --ip=0.0.0.0 --port=3333 --no-browser &
2. jupyter-notebook --port 3333 --no-browser



Code Reference's:

1. https://medium.com/kaggle-blog/microsoft-malware-winners-interview-1st-place-no-to-overfitting-ee0b664bfb4c
2. https://towardsdatascience.com/microsoft-malware-prediction-and-its-9-million-machines-22e0fe8c80c8
3. (Optional) https://www.researchgate.net/profile/Mfevzi_Esen/publication/340535548_Multiclass_Classification_with_Decision_Trees_Naive_Bayes_and_Logistic_Regression_An_Application_with_R/links/5e8f2767299bf130798a19fb/Multiclass-Classification-with-Decision-Trees-Naive-Bayes-and-Logistic-Regression-An-Application-with-R.pdf#page=101

3. Exploratory Data Analysis

In [13]:

```
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 [148]:

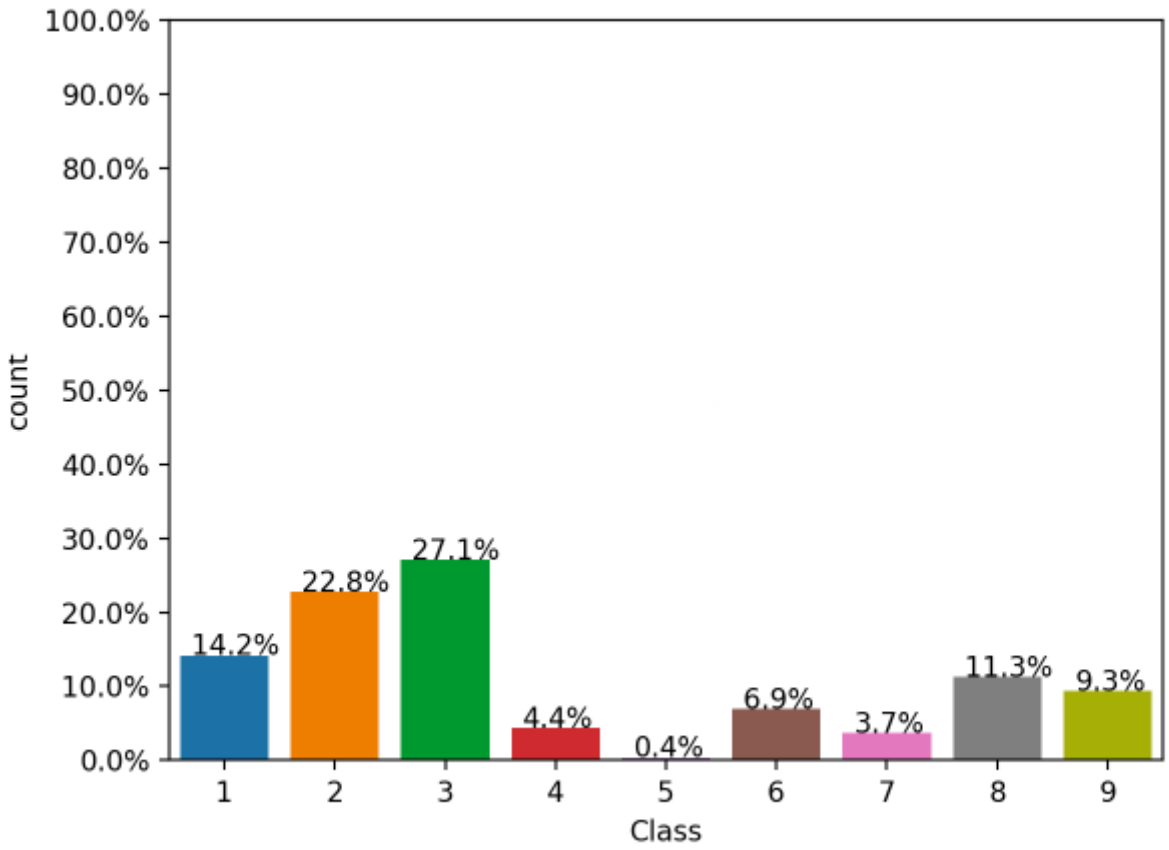
```
separating byte files and asm files

ource = 'train'
estination_1 = 'byteFiles'
estination_2 = 'asmFiles'

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

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

so by the end of this snippet we will separate all the .byte files and .asm files
 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 [7]:

```
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.ge

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

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

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



In [151]:

```
Y
```

Out[151]:

	Id	Class
0	01kcPWA9K2BOxQeS5Rju	1
1	04EjIdbPV5e1XroFOpiN	1
2	05EeG39MTRrl6VY21DPd	1
3	05rJTUWYAKNegBk2wE8X	1
4	0AnoOZDNbPXIr2MRBSCJ	1
...	...	...
10863	KFrZ0Lop1WDGwUtkusCi	9
10864	kg24YRJTB8DNdKMXpwOH	9
10865	kG29BLiFYPgWtpb350sO	9
10866	kGITL4OJxYMWEQ1bKBiP	9
10867	KGorN9J6XAC4bOEkmyup	9

10868 rows × 2 columns

3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [3]:

```
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_r
    # 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.ht
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

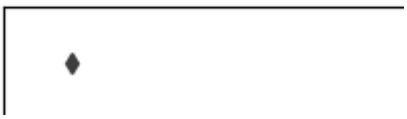
```
                     ID      size  Class
0  IHQoEqklzdpF9J37CNvc  1.359375      2
1  dmWXiZ4u3OA0Bj1Ea9gM  6.703125      3
2  3nZODLNAGgEF76dck14t  0.363281      1
3  ers73VJzBQHoYxilUdt8  3.597656      7
4  dHhYV16mSsuTqrzcAvBE  0.363281      8
```

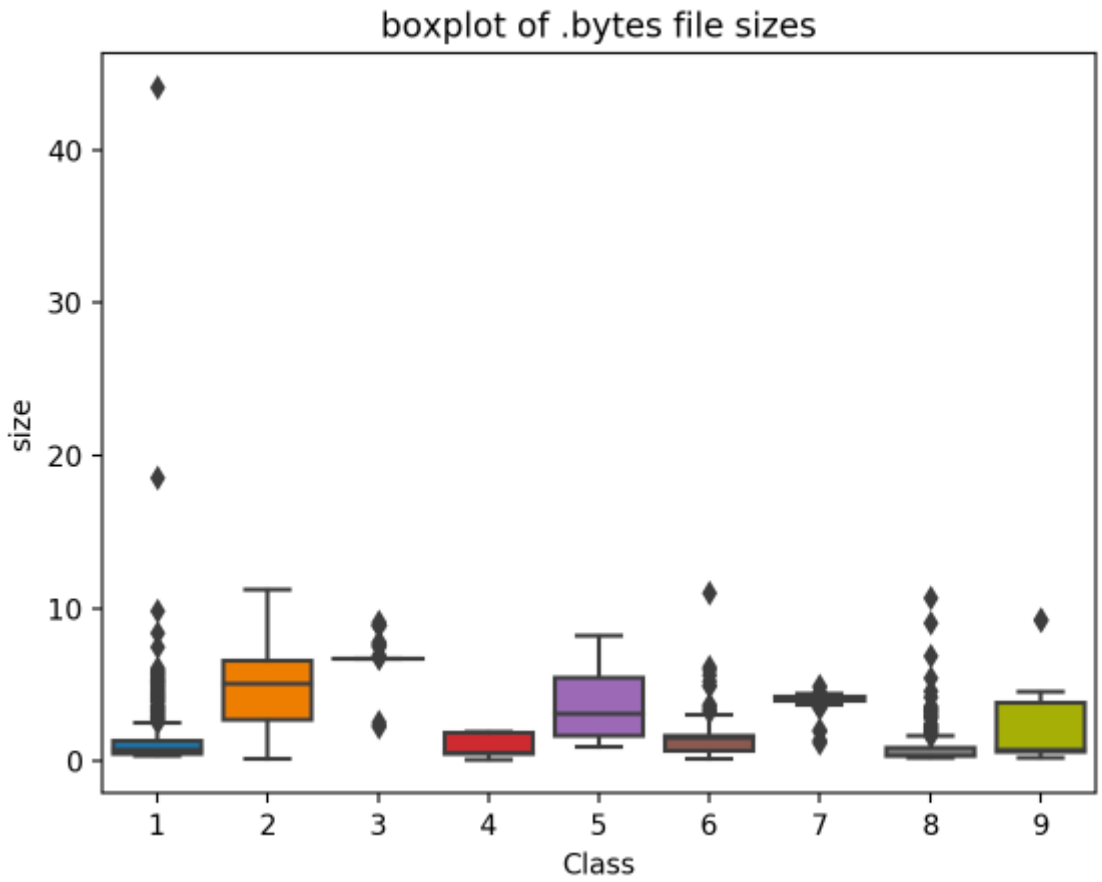
3.2.2 box plots of file size (.byte files) feature

In [4]:

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

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





3.2.3 feature extraction from byte files

In [63]:

```
#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
import tqdm
files = os.listdir('byteFiles')
filenames=[]
array=[]
tnt = 0
for file in files:
    tnt +=1
    print(tnt)
    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,
byte_feature_file.write("\n")
cnt =0
for file in files:
    cnt +=1
    print(cnt)
    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 [73]:

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

Out[73]:

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

2 rows × 258 columns

In [80]:

```
byte_features
```

Out[80]:

	ID	0	1	2	3	4	5	6	7	8	...
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...
...	...	...	...	...	...	...	...	...	...	...	...
10863	loIP1tiwELF9YNZQjSUO	5268	1177	1072	1222	1238	1159	1143	1126	1149	...
10864	LOP6HaJKXpkic5dyuVnT	3032	298	248	293	274	213	203	222	257	...
10865	LOqA6FX02GWguYrI1Zbe	5671	221	270	323	313	155	248	147	261	...
10866	LoWgaidpb2IUM5ACcSGO	3637	437	453	506	511	390	431	407	405	...
10867	lS0lVqXeJrN6Dzi9Pap1	3534	373	385	432	495	399	393	373	399	...

10868 rows × 258 columns

In [72]:

```
get_files
```

```
 'D4UKRvhXzCsW2Aahu61W',
 '8EQSs7XKIkVNHL0Z41au',
 'C2spzDaAf6rwHJ4tBgFv',

 '5QPsHaRAEIt7ibDdVy3F',
 'GyN1TW59jIfadSXxUZFb',
 'a7Niv6pD5WPycnQK1TZ8',
 'f1KTZSymHw5na9Vz4Nsd',
 'h1nZq0BX5LruizPw3EUm',
 'f2n3OzR8yBvu5ScmNQeP',
 '6CbnMfItA5py0odwQjPR',
 'DnSqtl6oksRxOwP8g0AM',
 'g1lGzN3S8fPhx96QVbE7',
 '860EGyhmJVB49WrLgOC3',
 '8APhEd3UCifDsc4zVemR',
 'IkFwMHPbmDB0VvxEWOGX',
 'EdwQ1my8754NBkMaJxGA',
 'ArMnPlD6b7SdcX538UqQ',
 'epFEs6LjYC2nB1Kw7lDP',
 'aISJkXUny0dojZKYNBAP',
 '4XRD81BbiqusJ3OtoS2r',
```

In [81]:

```
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[81]:

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

2 rows × 260 columns

In [85]:

```
byte_features_with_size[byte_features_with_size['ID']=='IAX6uCwTfFUH8vOKkzcV']
```

Out[85]:

	ID	0	1	2	3	4	5	6	7	
9208	IAX6uCwTfFUH8vOKkzcV	140732	117950	109652	110425	10682	13677	12167	10732	1
9209	IAX6uCwTfFUH8vOKkzcV	140732	117950	109652	110425	10682	13677	12167	10732	1

2 rows × 260 columns

In [92]:

```
byte_features_with_size = byte_features_with_size.drop(9209)
```

In [93]:

```
byte_features_with_size.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10868 entries, 0 to 10868
Columns: 260 entries, ID to Class
dtypes: float64(1), int64(258), object(1)
memory usage: 21.6+ MB
```

In [94]:

```
# 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 - mi
    return result1
result = normalize(byte_features_with_size)
```

In [95]:

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

Out[95]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747

2 rows × 260 columns

In [96]:

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

Out[96]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.00205
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.00474
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.00507
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.00031
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.00014

5 rows × 260 columns

In [97]:

```
result.info()
```

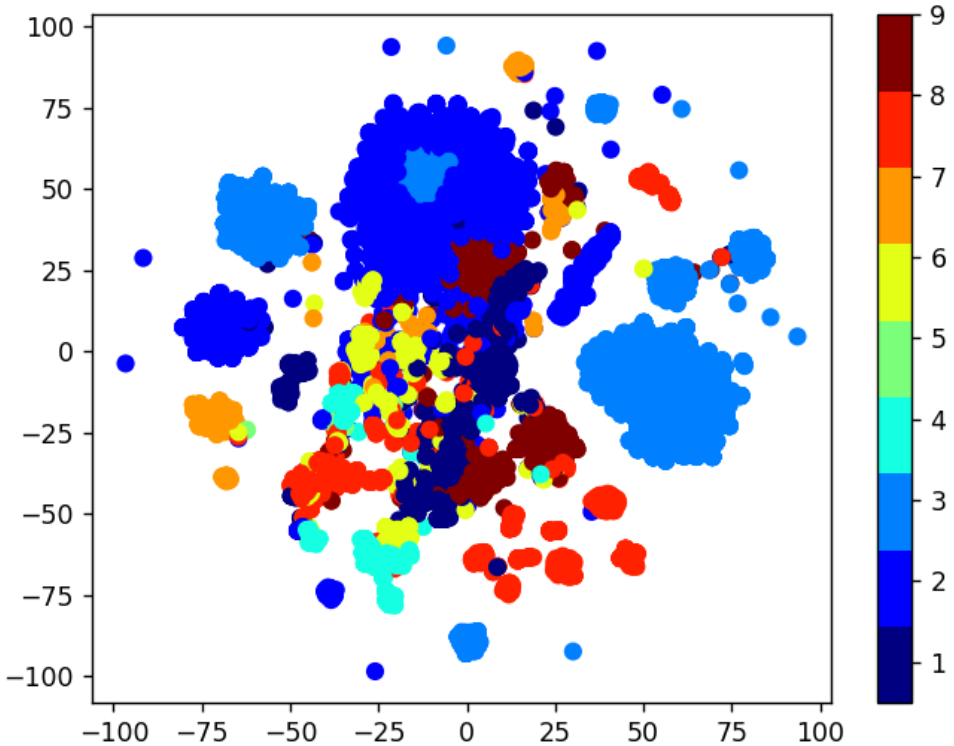
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10868 entries, 0 to 10868
Columns: 260 entries, ID to Class
dtypes: float64(258), int64(1), object(1)
memory usage: 21.6+ MB
```

3.2.4 Multivariate Analysis

In [0]:

```
#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()
```

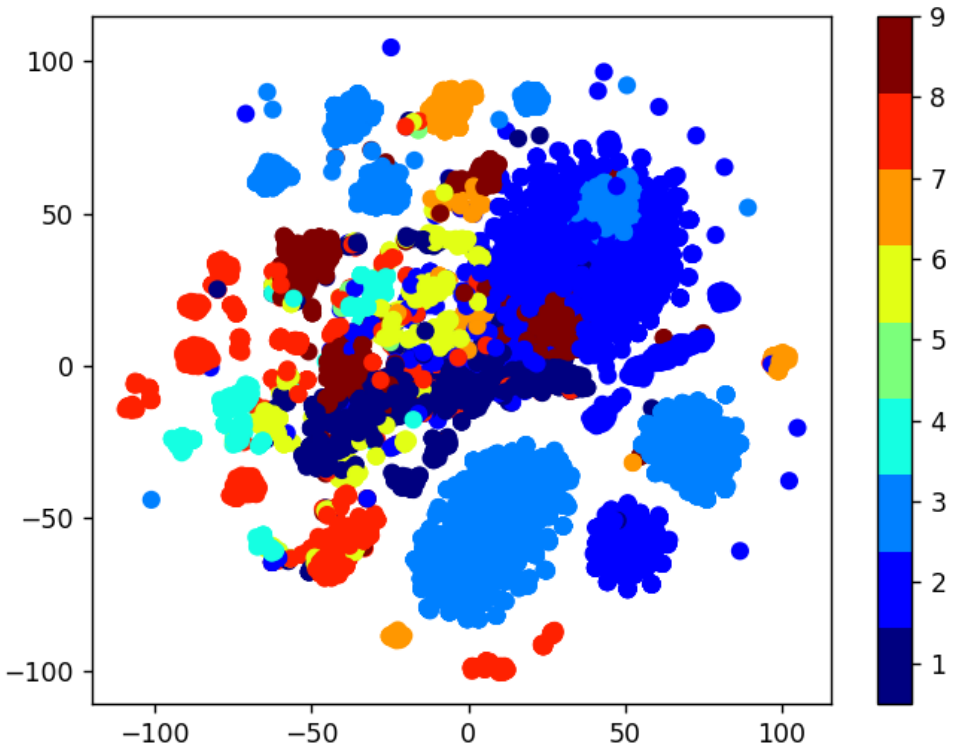
```
<IPython.core.display.Javascript object>
```



In [0]:

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

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



Train Test split

In [98]:

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

In [99]:

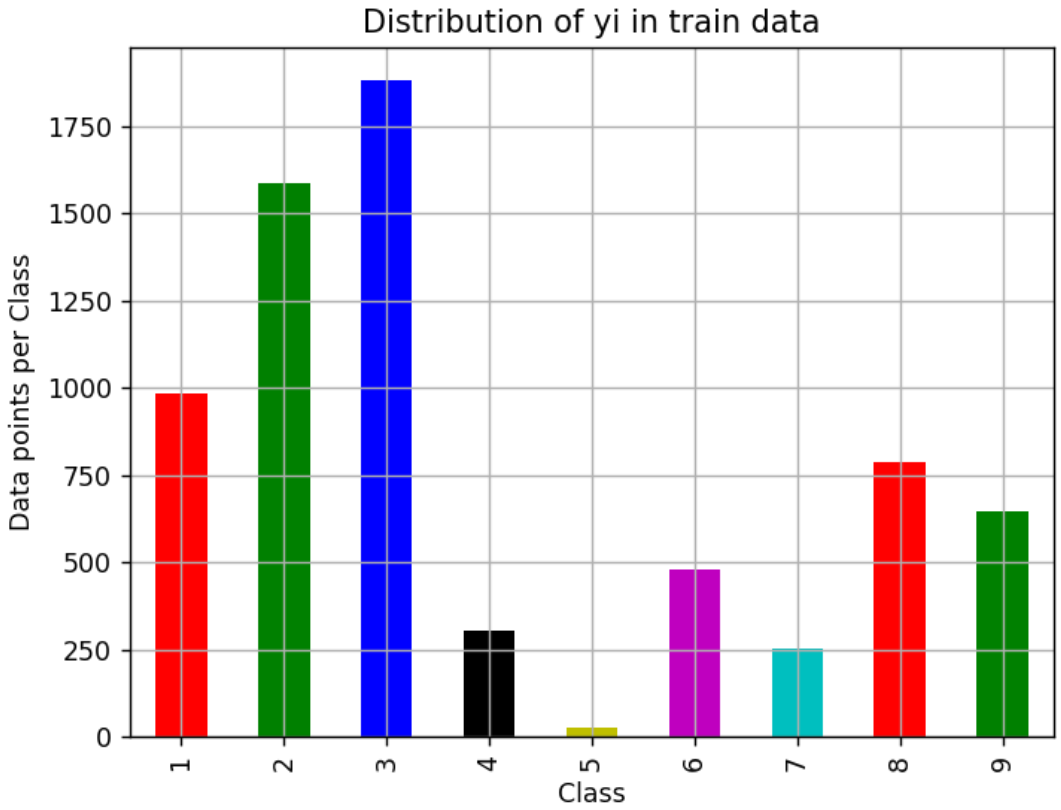
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
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
```

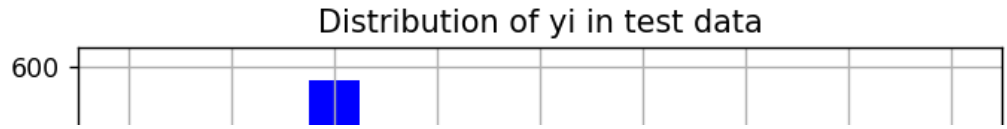
In [0]:

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

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

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


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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


```

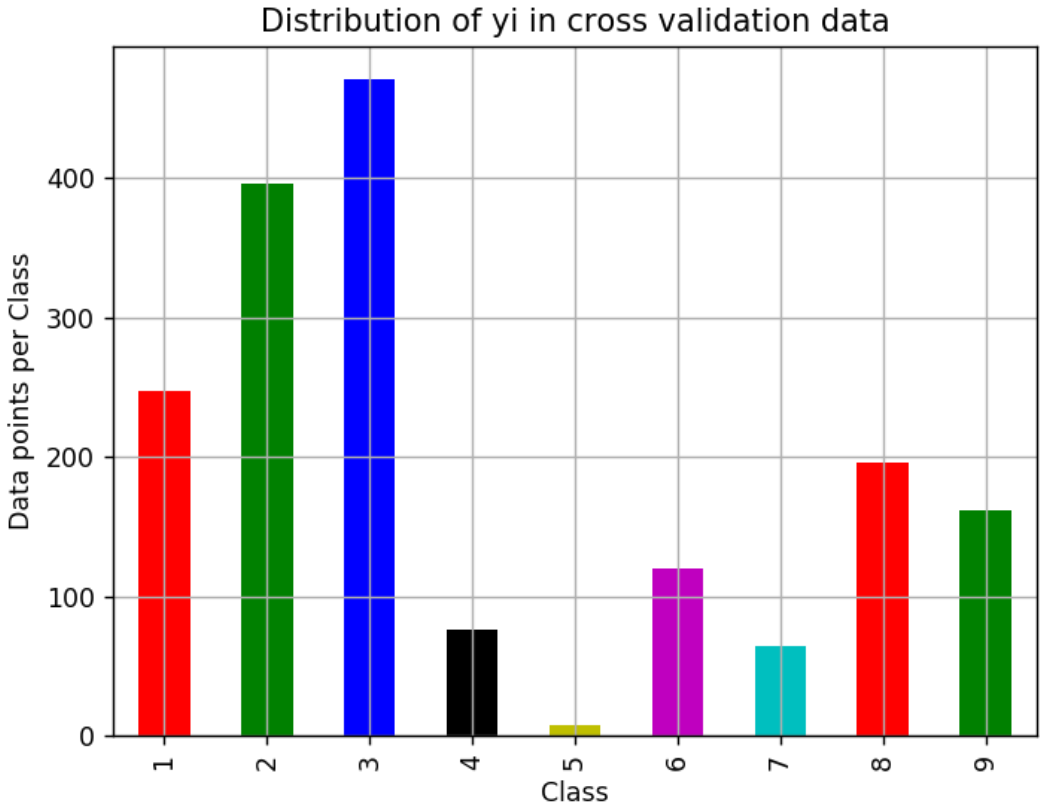
```
<IPython.core.display.Javascript object>
```




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

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




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

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




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

In [124]:

```
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)*10
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pre

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

    # 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 i
    # 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 i
    # 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
    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
    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
    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

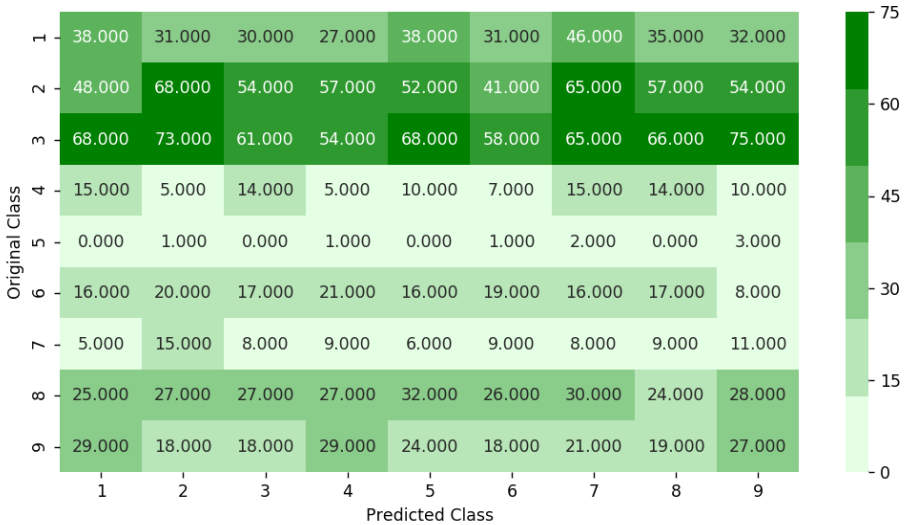
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

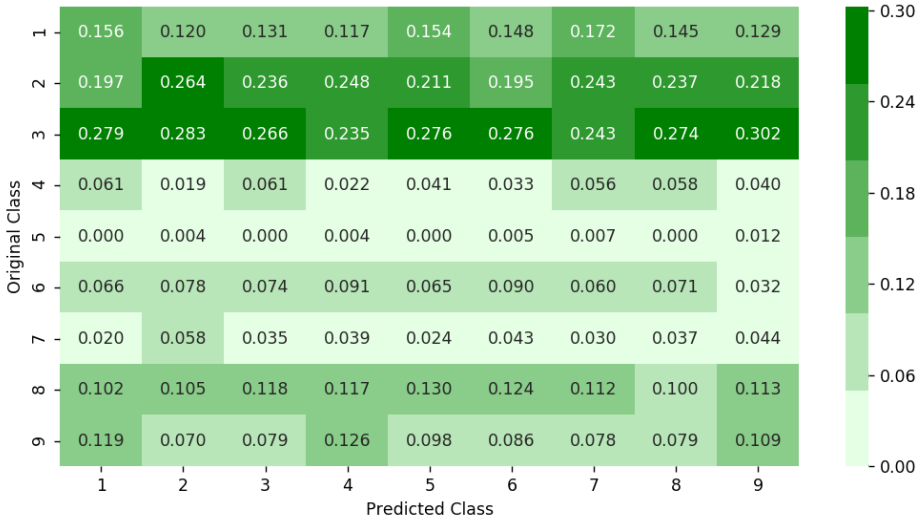
In [0]:

```
# 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_predic


# 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, e

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

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

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

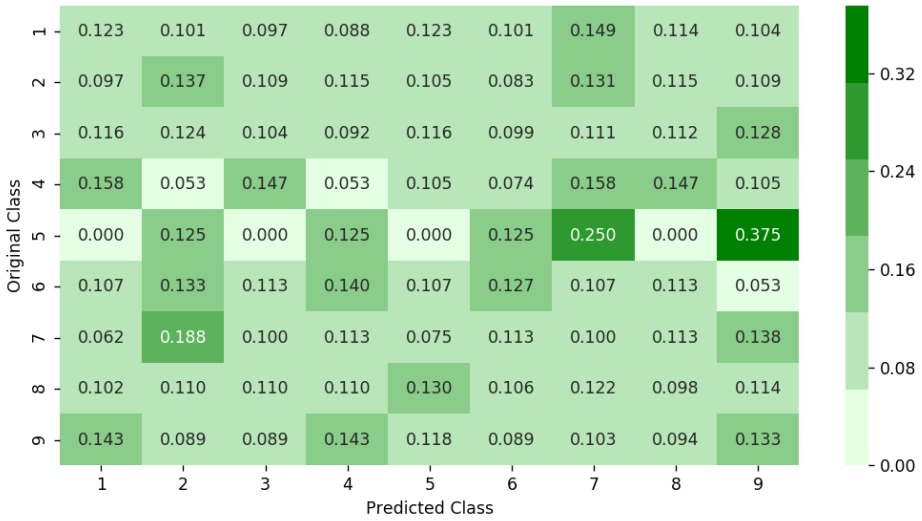



```
---------------------------------------------------- Precision matrix --
--------------------------------------------------

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




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

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



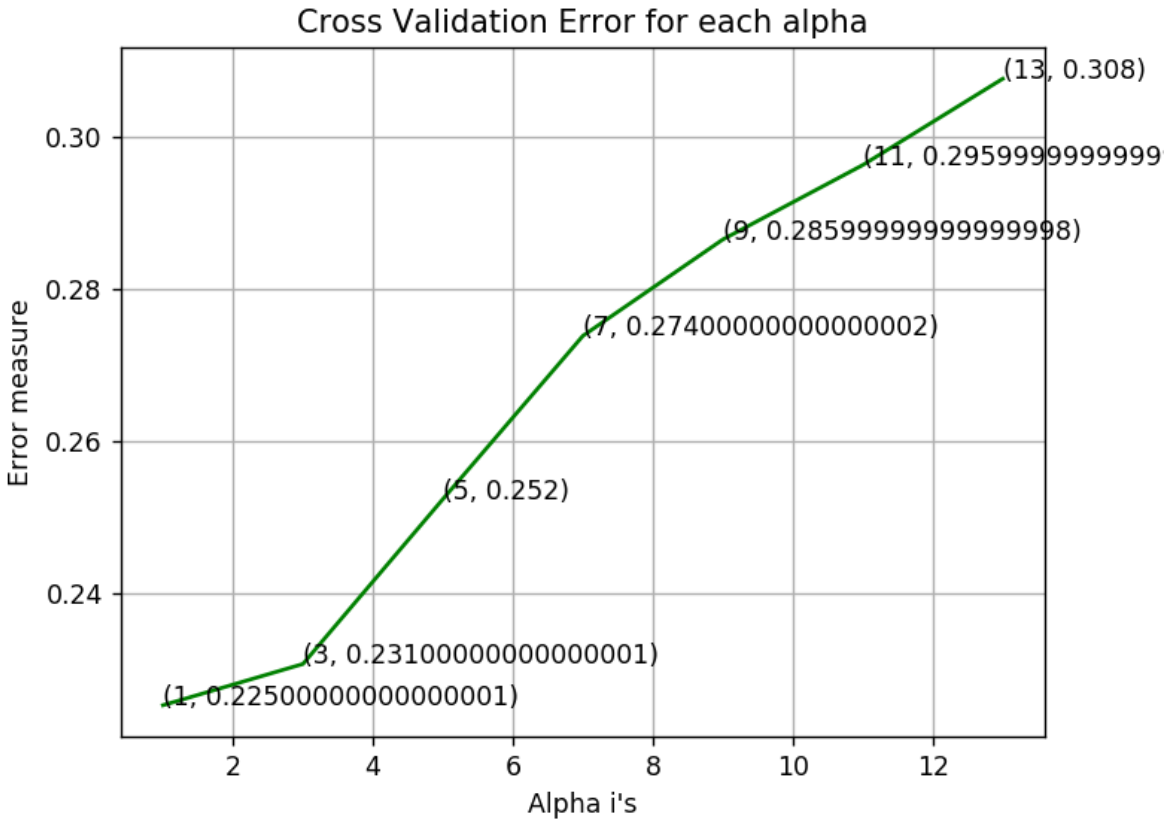

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.2. K Nearest Neighbour Classification

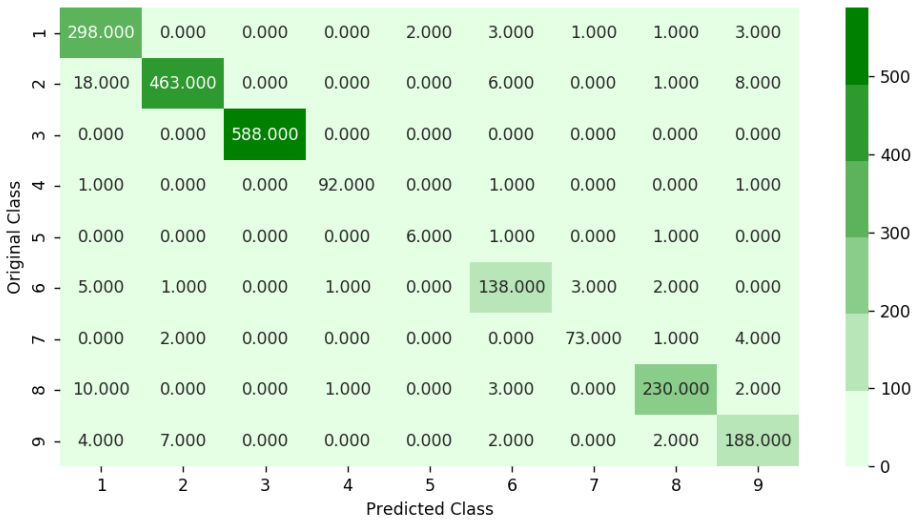
In [0]:

```
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=1

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

best_alpha = np.argmin(cv_log_error_array)

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

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

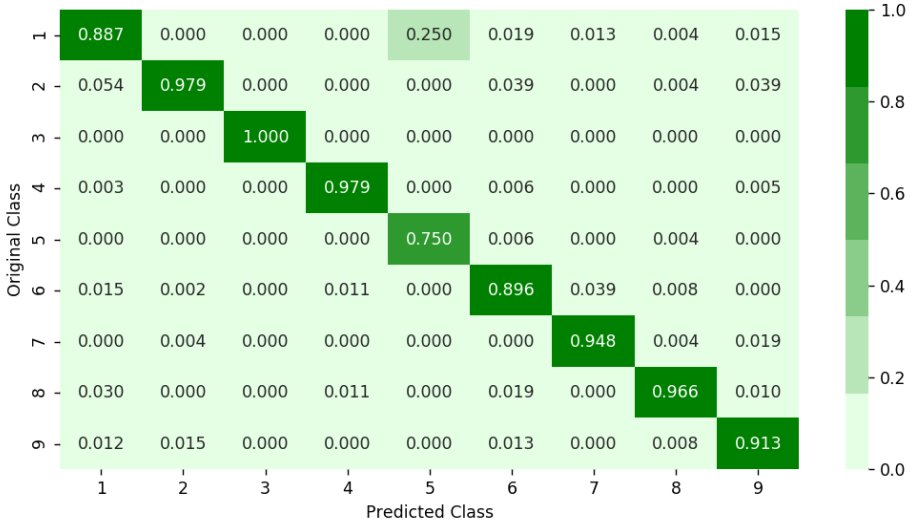
```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154

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



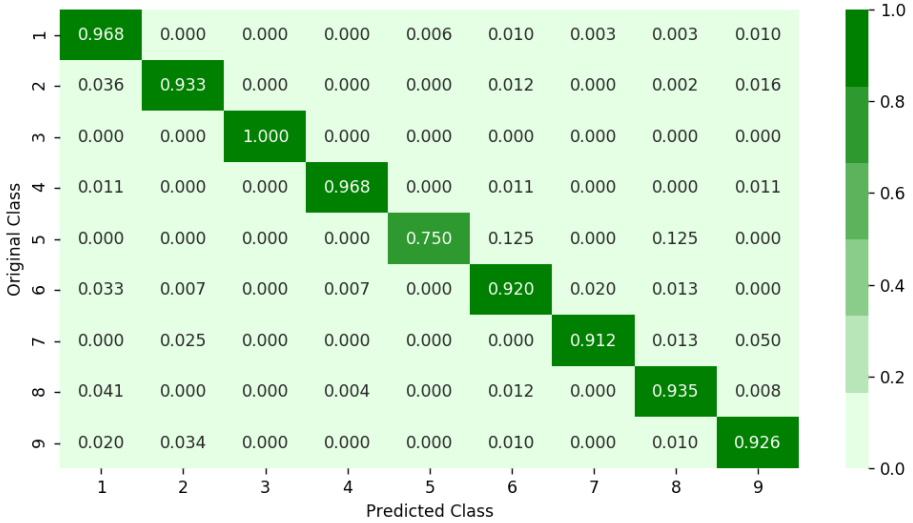

```
For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.2253
86237304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
---------------------------------------------------- Confusion matrix --
--------------------------------------------------

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




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

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




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

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



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

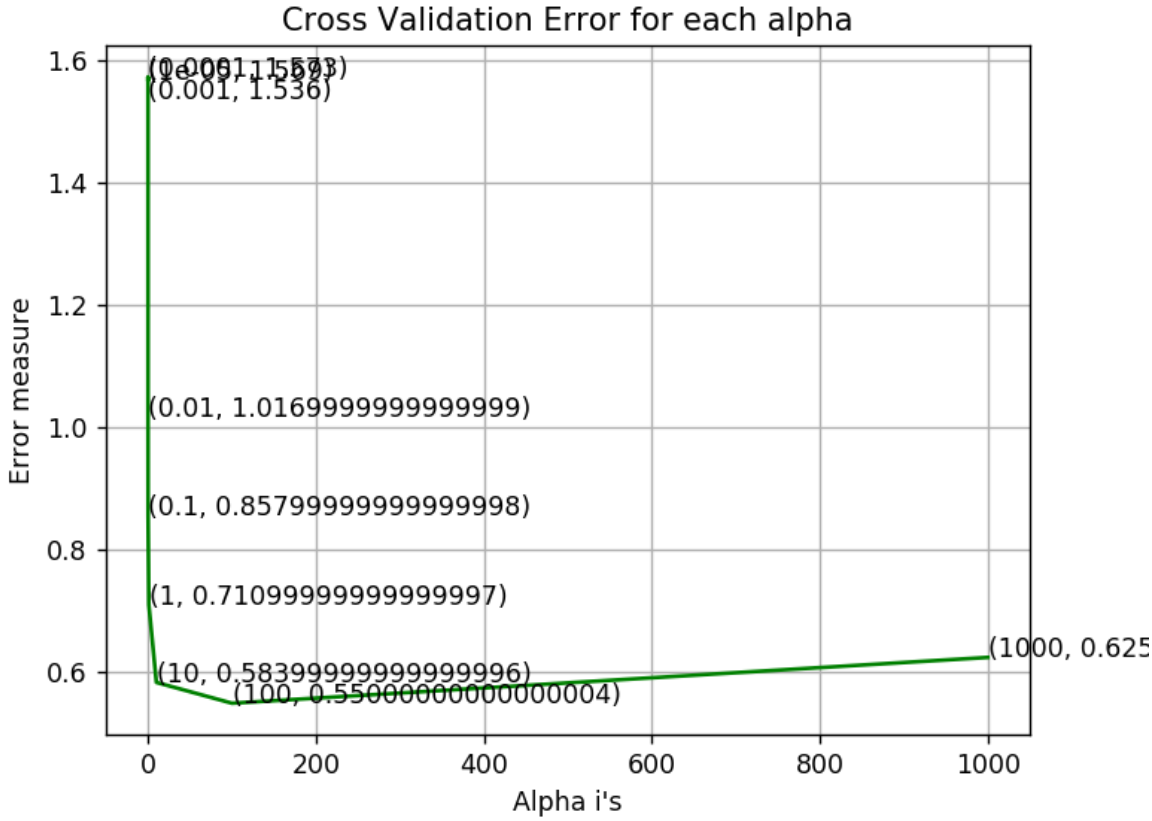
4.1.3. Logistic Regression

In [0]:

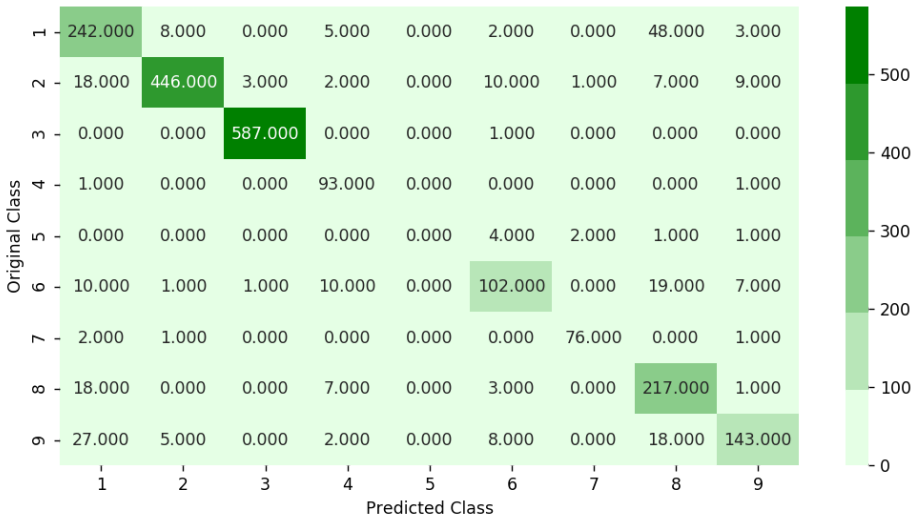
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_interce
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='
# 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 Gradien
# predict(X)    Predict class labels for samples in X.

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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
```

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

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

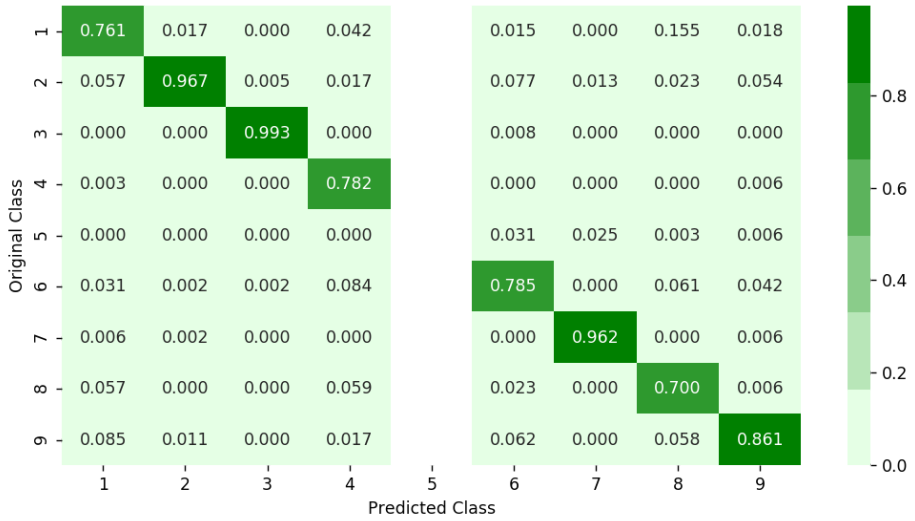



```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
---------------------------------------------------- Confusion matrix --
-----------------------------------------------

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

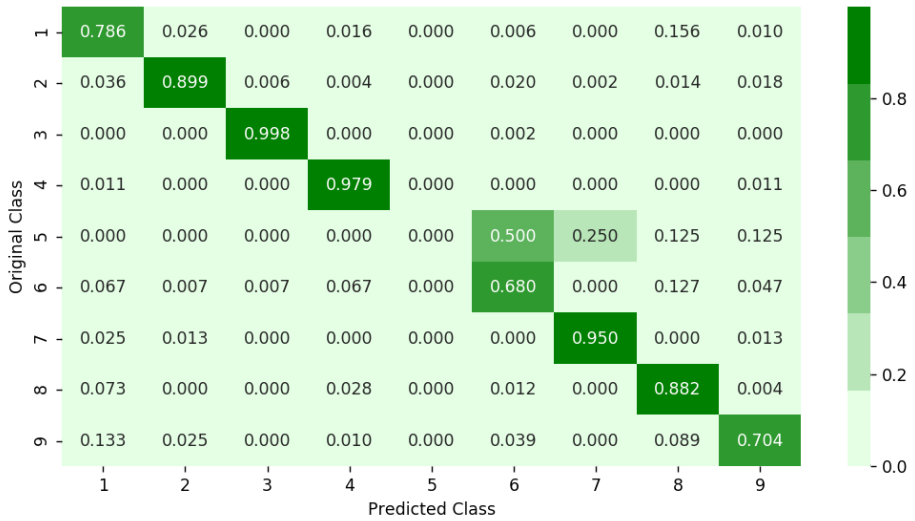



```
---------------------------------------------------- Precision matrix --
```

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




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

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




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

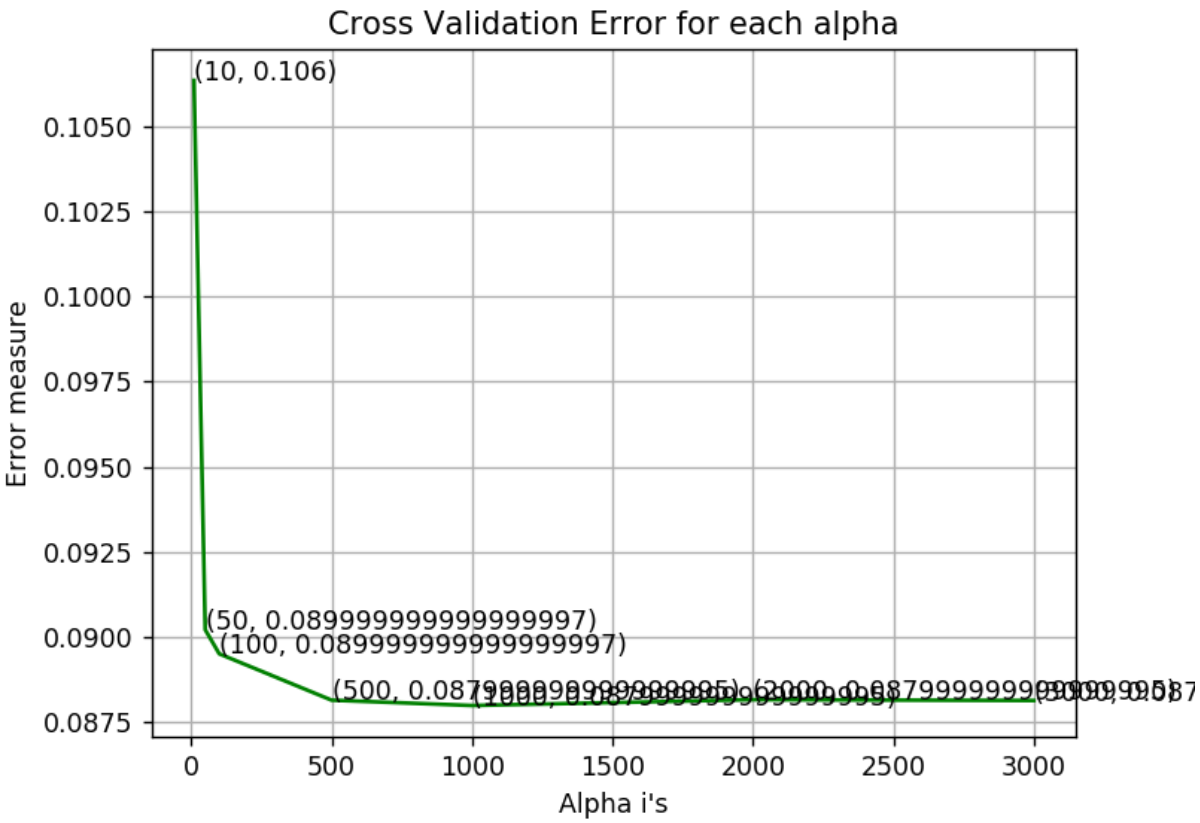
4.1.4. Random Forest Classifier

In [0]:

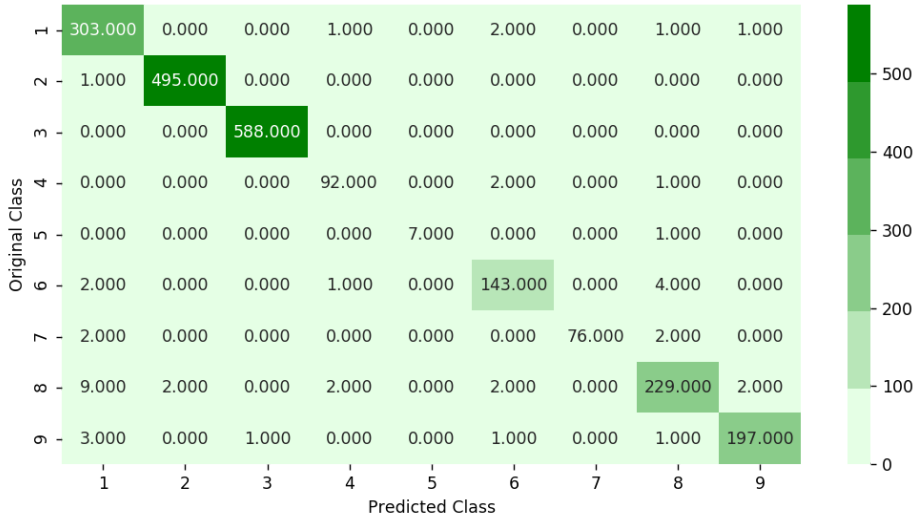
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_dep
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_no
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N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

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training da
# 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/lesson
# --------------------------------

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=1

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-
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
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
predict_y = sig_clf.predict_proba(X_test)
```

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

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

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

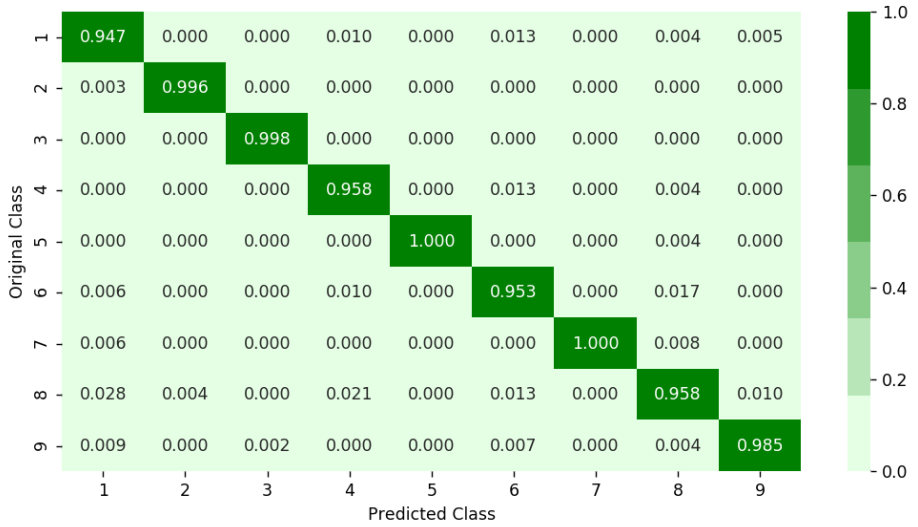



```
For values of best alpha =  1000 The train log loss is: 0.026647629180
1
For values of best alpha =  1000 The cross validation log loss is: 0.0
879849524621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix --
--------------------------------------------------

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




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

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




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

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

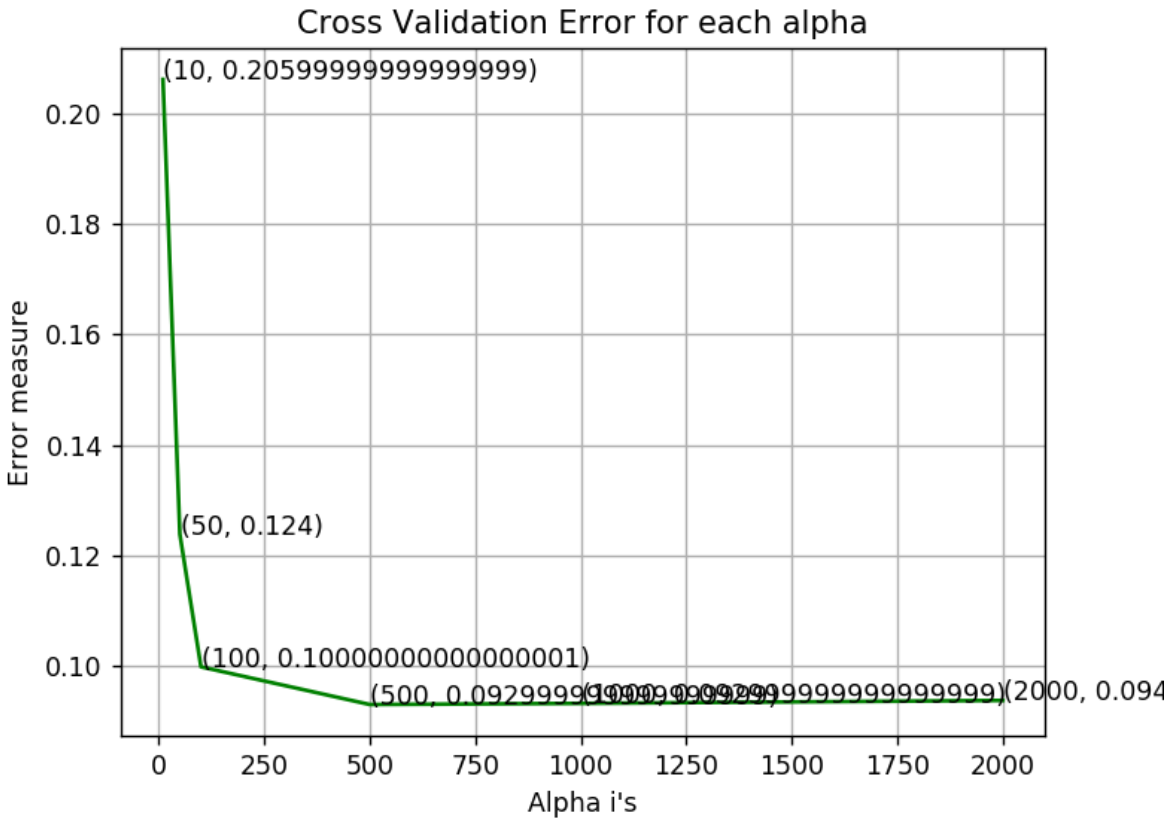

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.5. XgBoost Classification

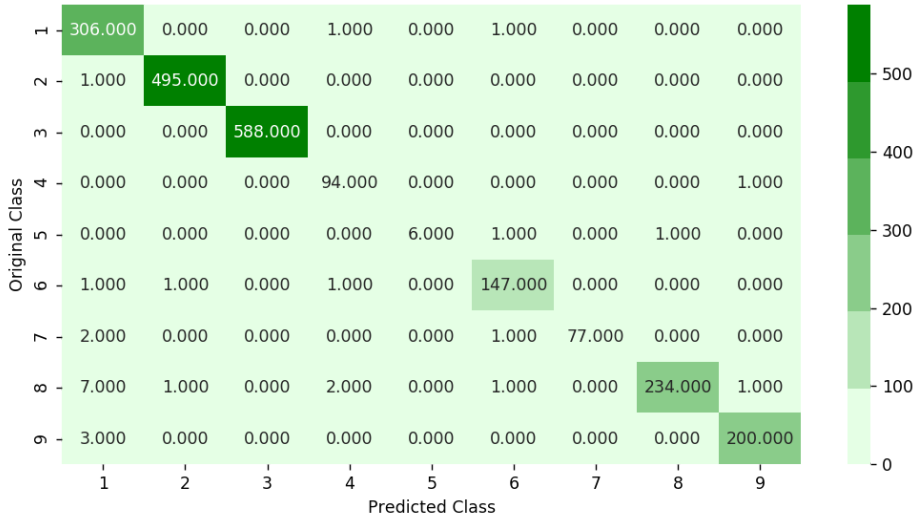
In [0]:

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

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

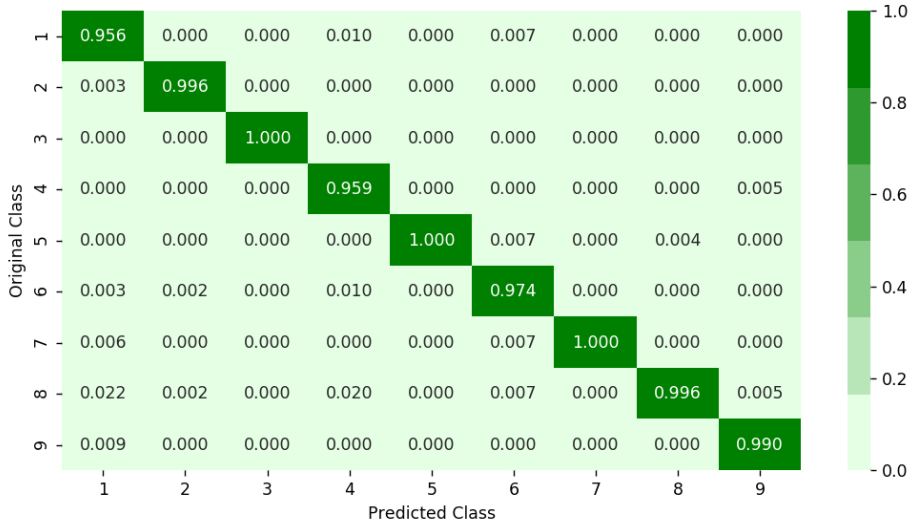



```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.09
31035681289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
---------------------------------------------------- Confusion matrix --
--------------------------------------------------

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



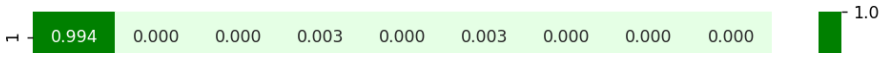

```
---------------------------------------------------- Precision matrix --
-----------------------------------------------
```

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




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

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




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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [0]:

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

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

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  9.3min remainin
g:  5.4min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 10.1min remainin
g:  3.1min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remainin
g:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 14.2min finished
```

Out[75]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1,
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, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.
1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_dept
h': [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=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

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

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

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

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

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

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In parall
el 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/vide

os/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 [0]:

```
#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 [0]:

```
#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:','.ed
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

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

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


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

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

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


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

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


def main():
    #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 [11]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the
# 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[11]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [12]:

```
#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_r
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=151963852
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.ht
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

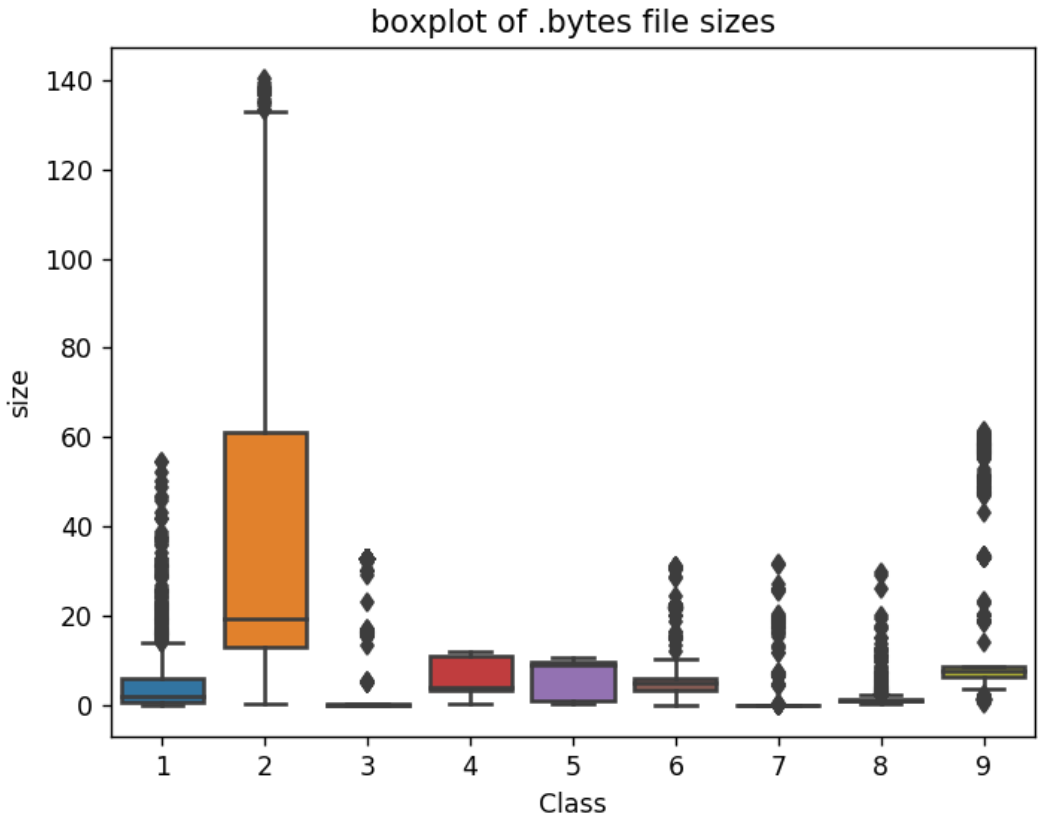
```
                     ID       size  Class
0  k0DVI4svgBRA8pjlf9M1   4.291086      1
1  57Y1sP8cgERT6OQyCJjN  33.062117      9
2  3X2jQ6SzeKCxA7WmlqoZ   0.171789      3
3  JMngbfY3PC5WkjA2iSvH   0.805081      1
4  iSUhzuAXpcy1ofT7xW0d   0.970914      8
```

4.2.1.2 Distribution of .asm file sizes

In [0]:

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

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



In [13]:

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

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

Out[13]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 54 columns

In [14]:

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

Out[14]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.eda
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	

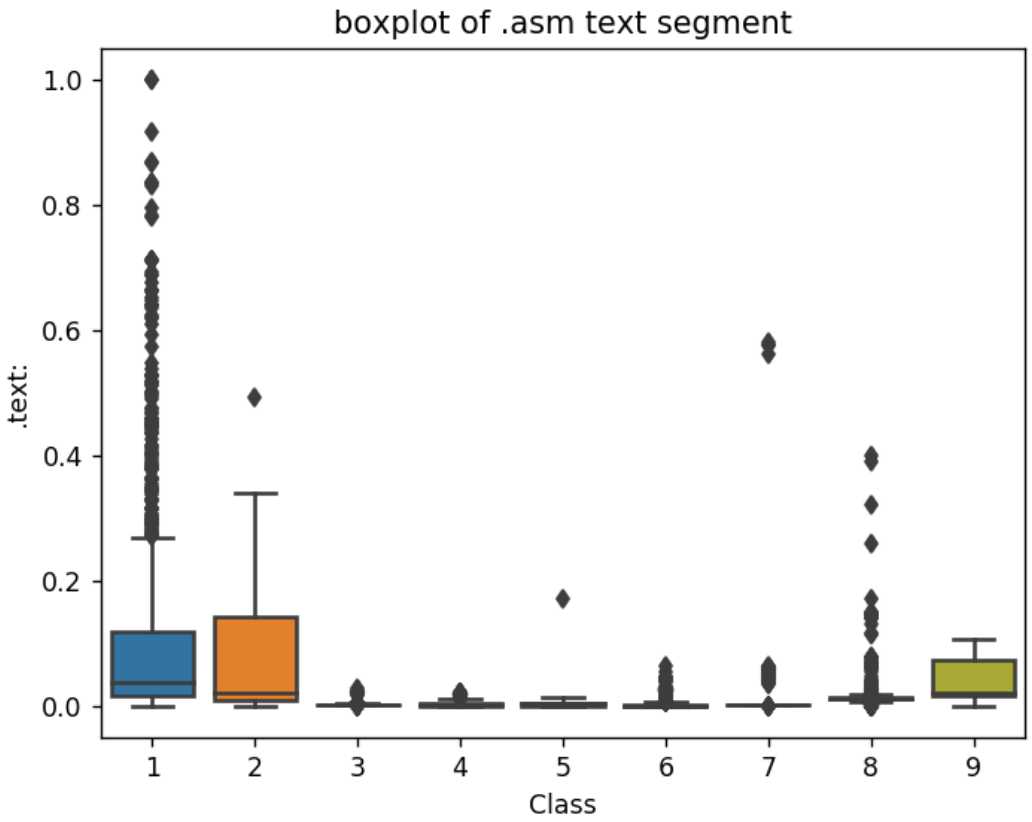
5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [0]:

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

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

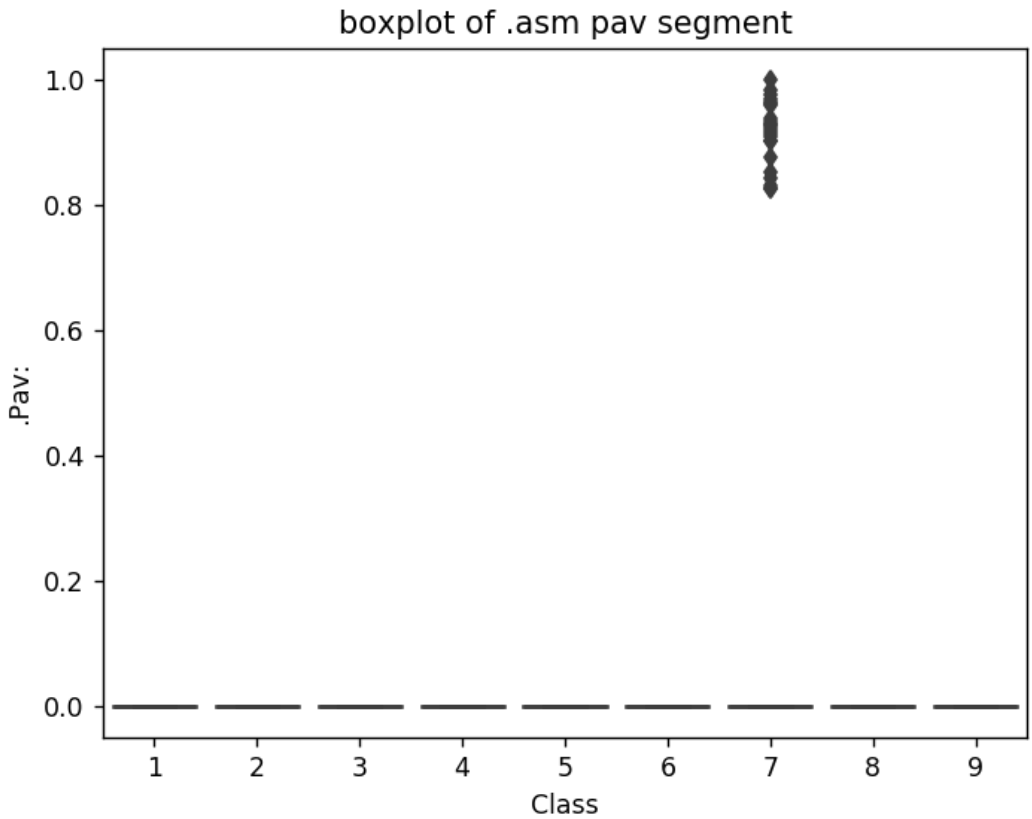


The plot is between Text and class
Class 1,2 and 9 can be easly separated

In [0]:

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

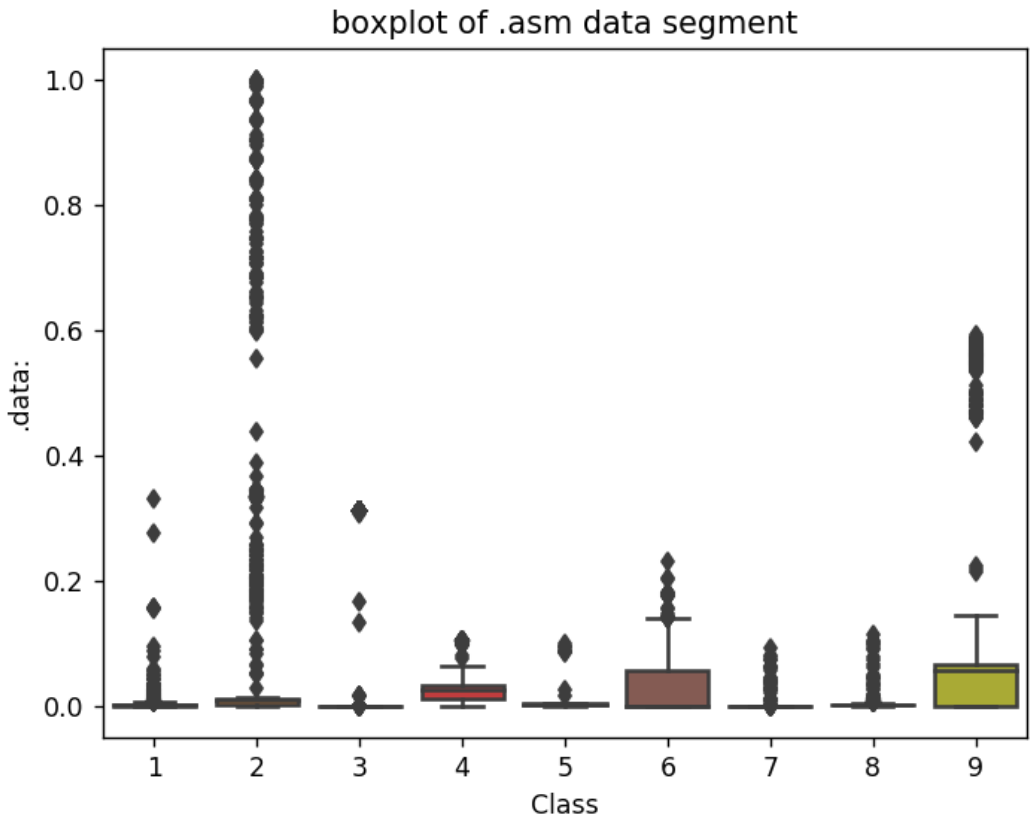
```
<IPython.core.display.Javascript object>
```



In [0]:

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

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

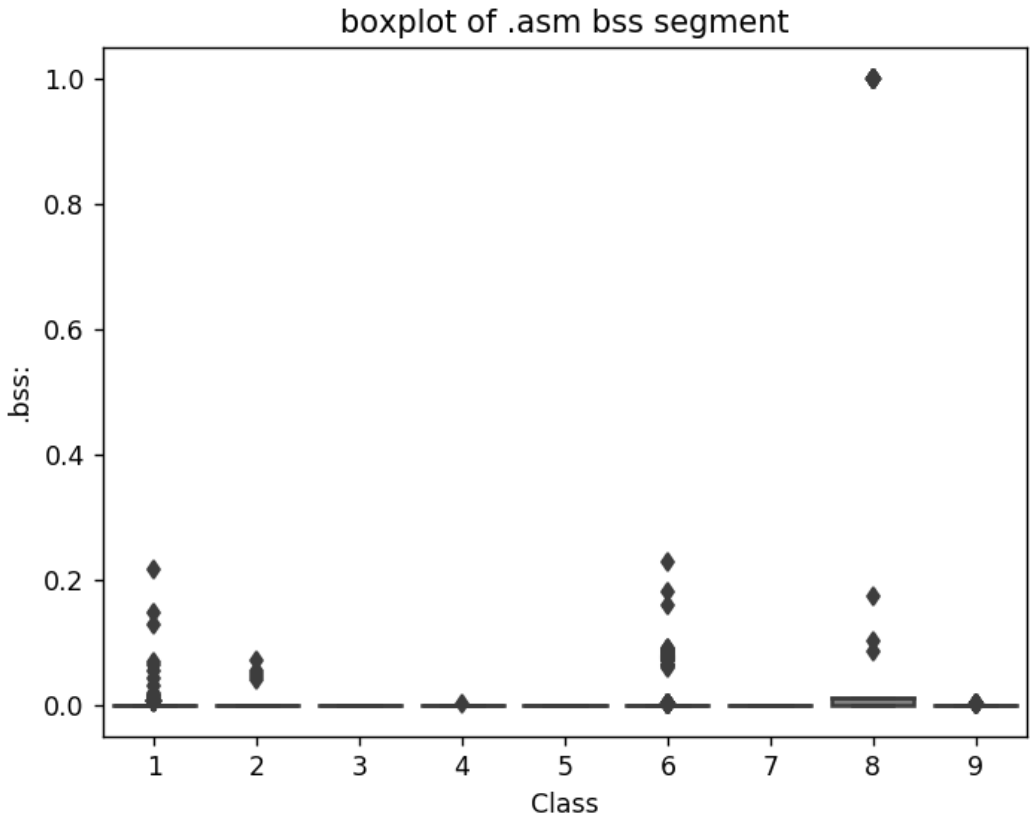


The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [0]:

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

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

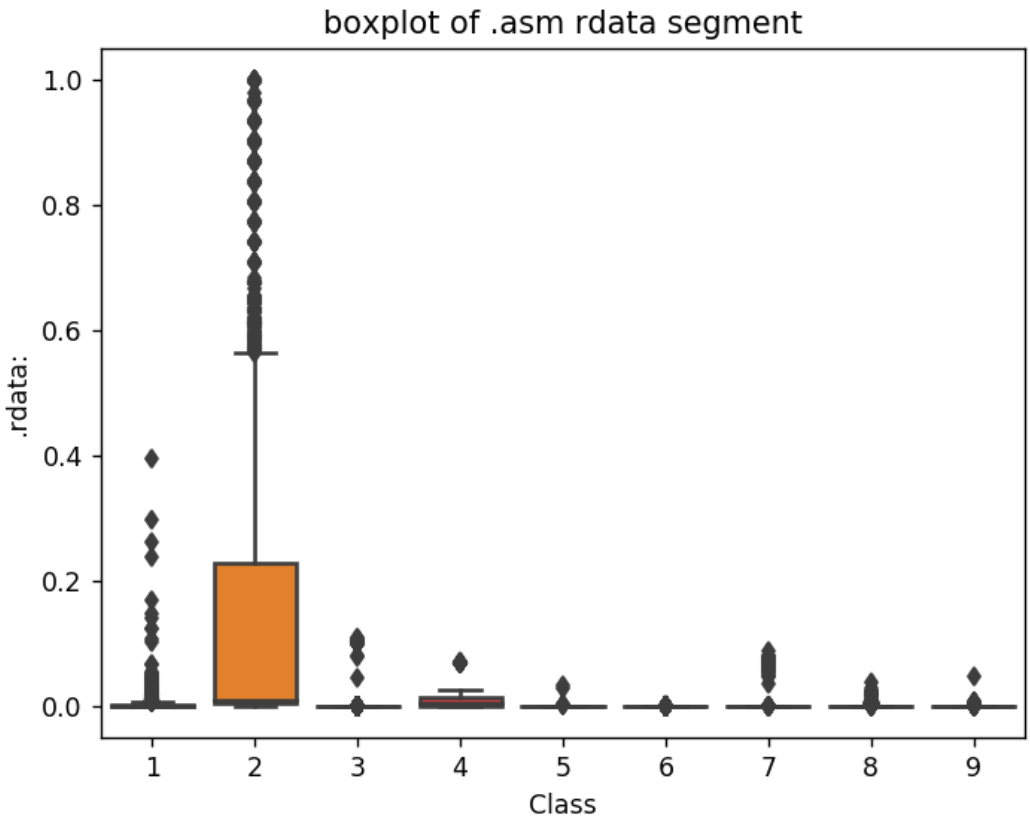


plot between bss segment and class label
very less number of files are having bss segment

In [0]:

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

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

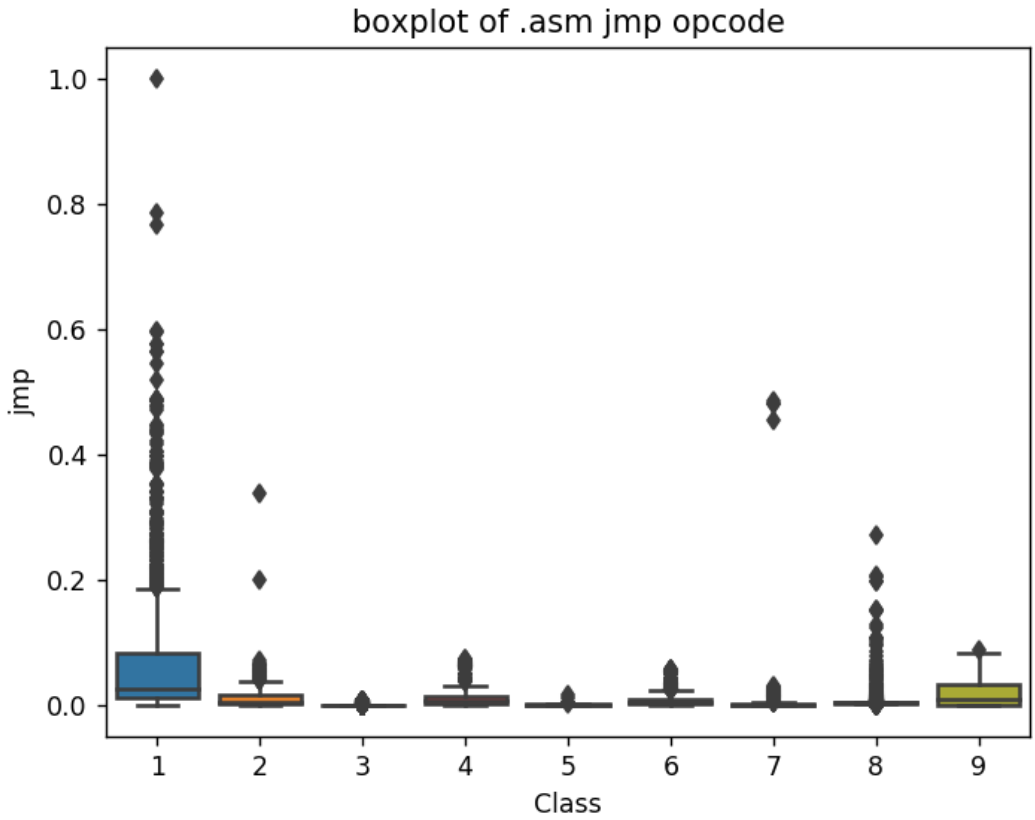


Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [0]:

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

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

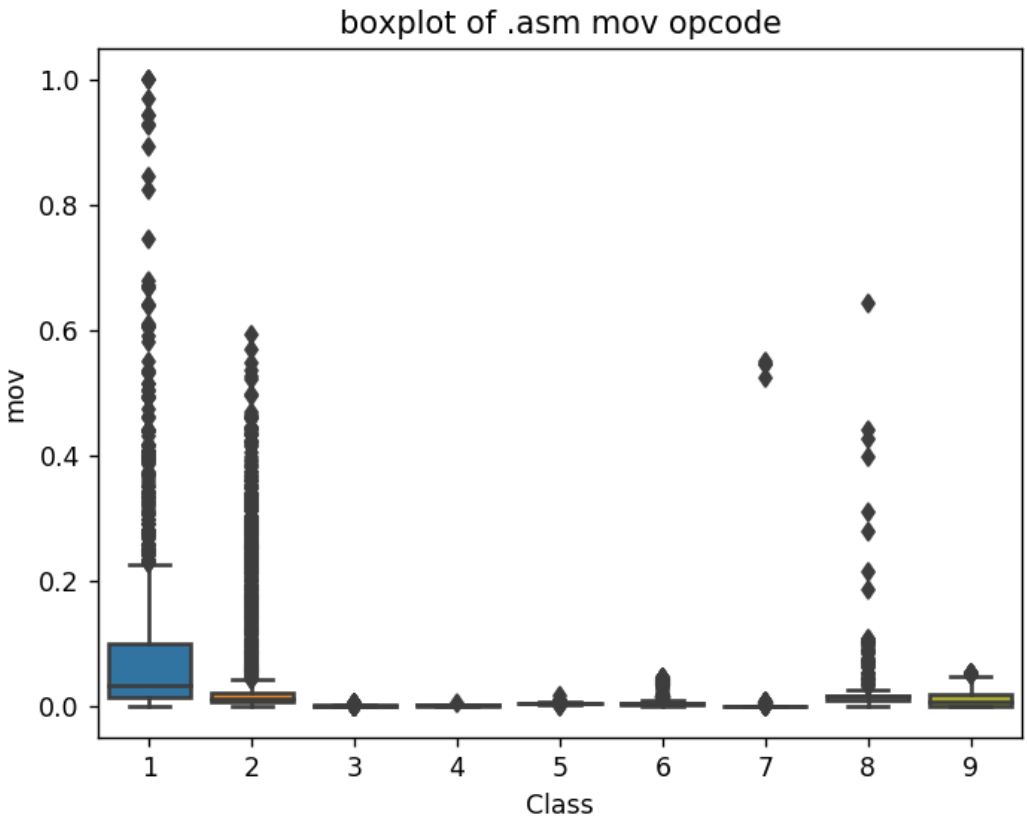


plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [0]:

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

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

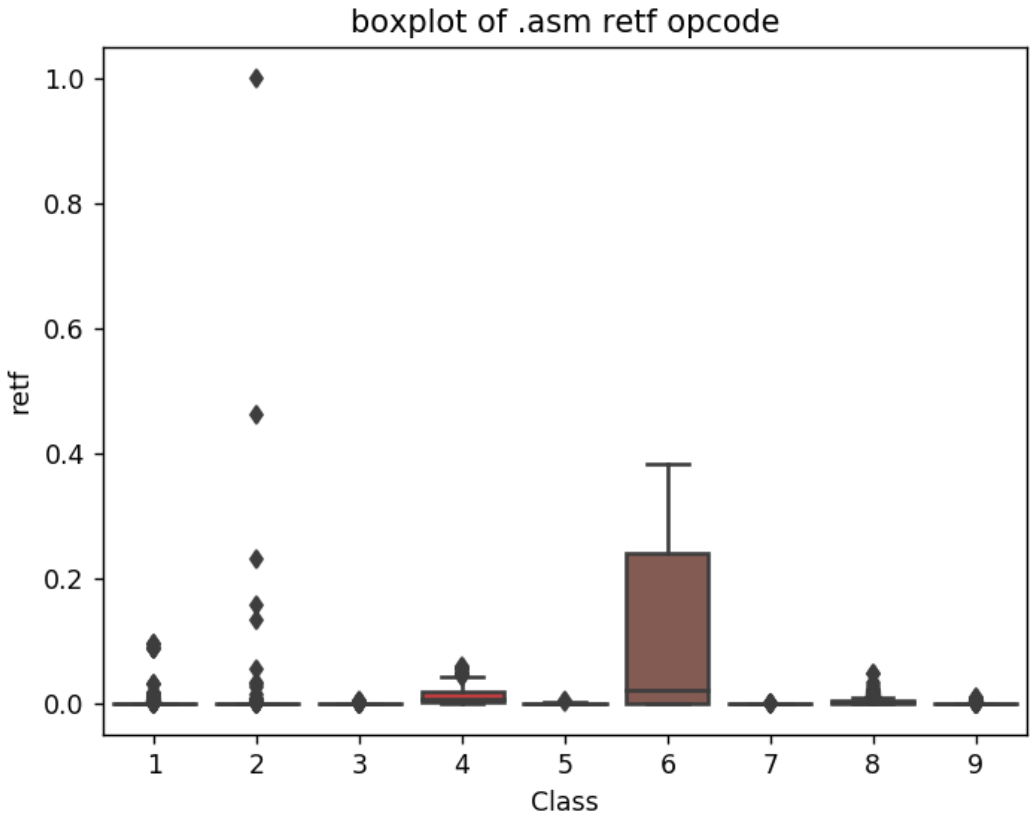


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [0]:

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

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

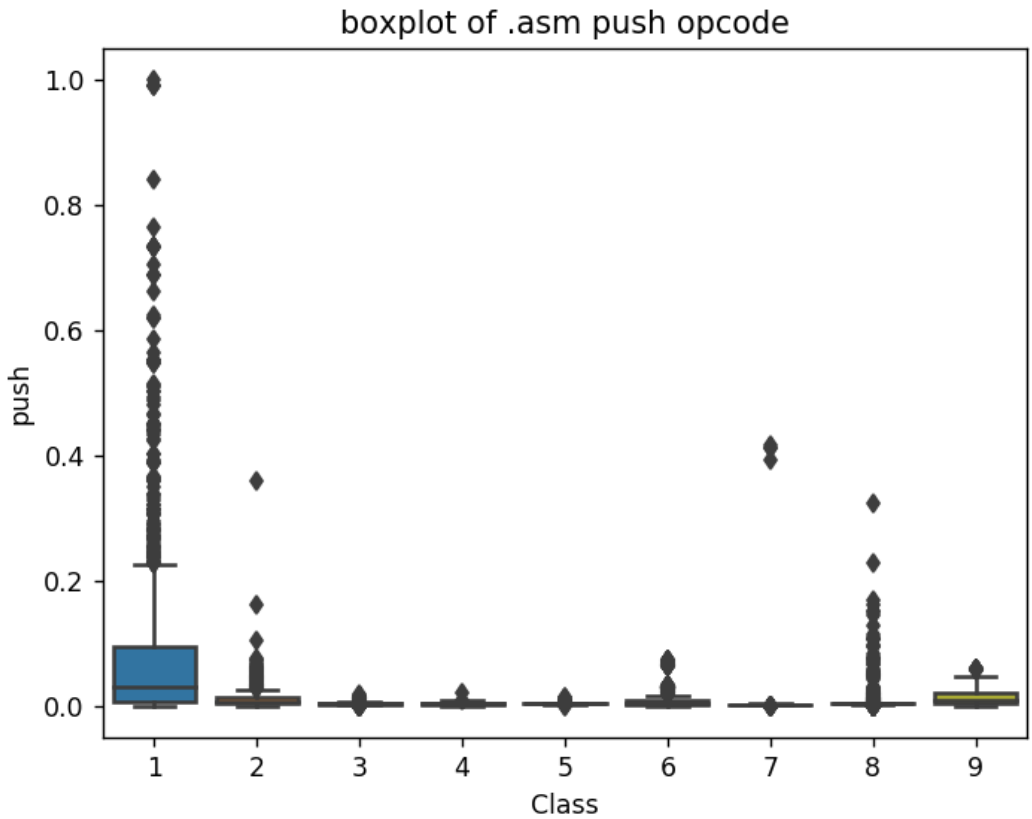


plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [0]:

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

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



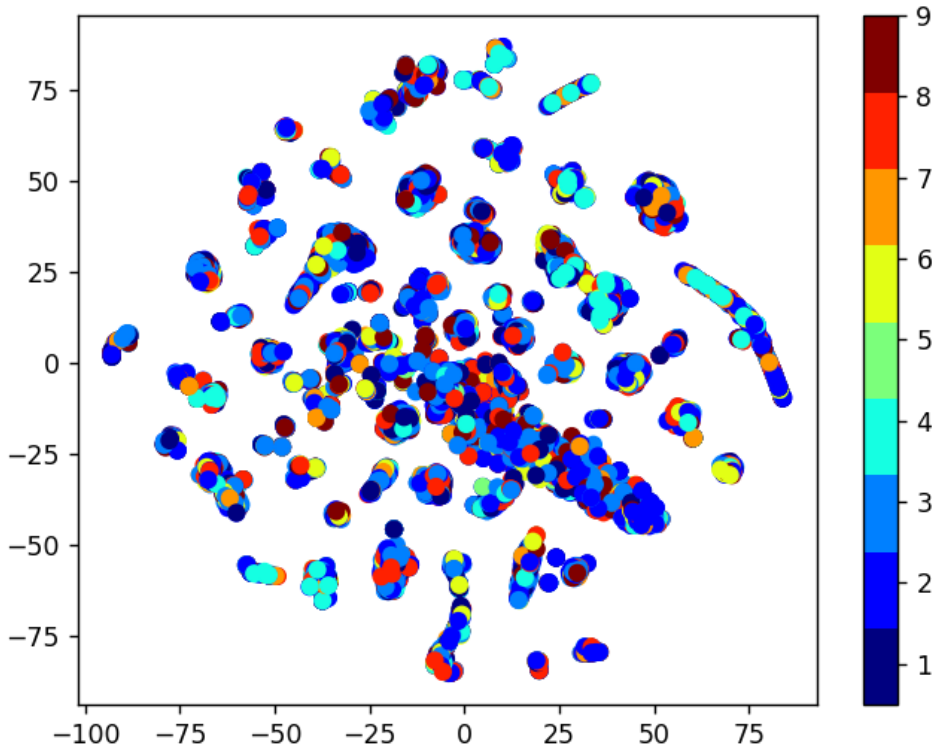
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

In [0]:

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

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

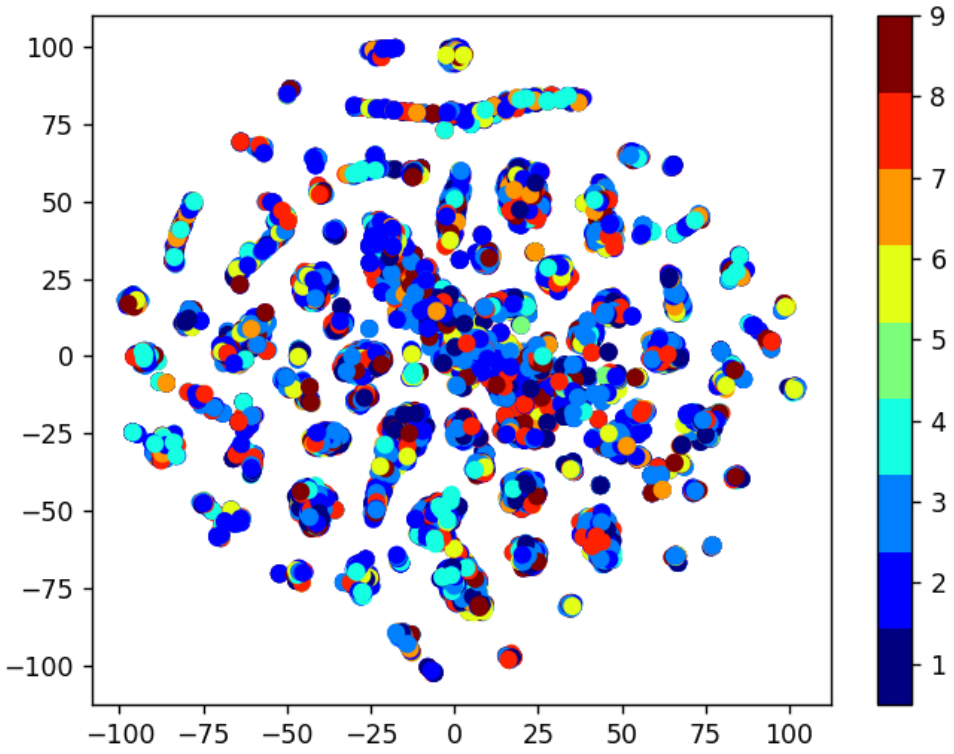
```
<IPython.core.display.Javascript object>
```



In [0]:

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

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

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



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [15]:

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

In [17]:

```
'''
    Getting asm details for train, cv and test
'''
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,str
X_train_asm, X_cv_asm,   y_train_asm, y_cv_asm   = train_test_split(X_train_asm, y_t
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
```

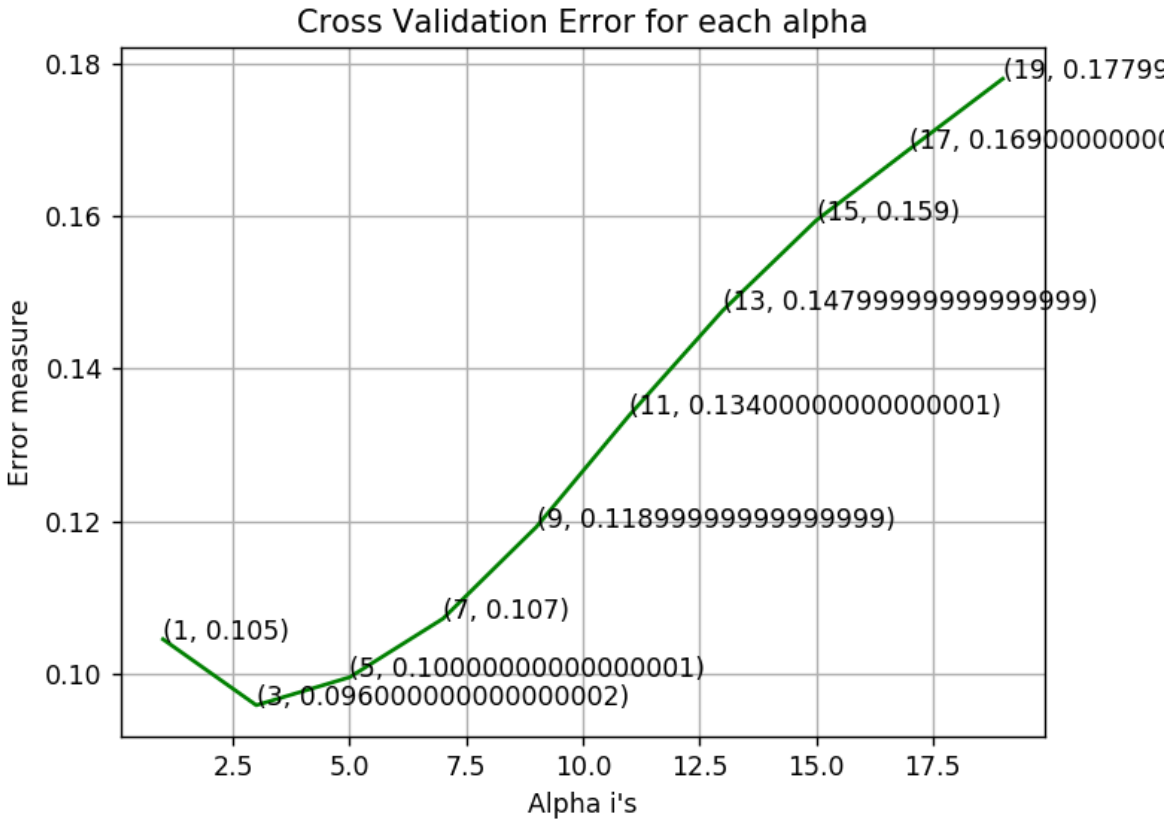
4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

In [0]:

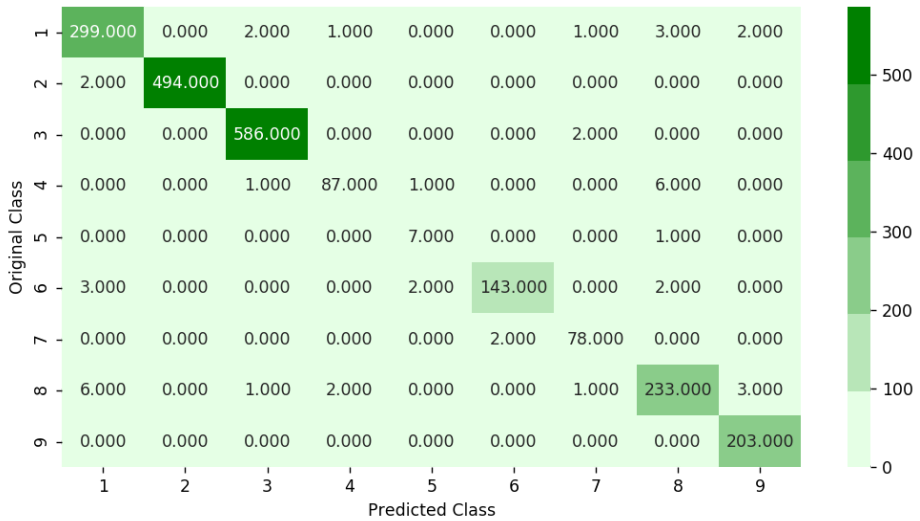
```
alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)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

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

```
log_loss for k =  1 is 0.104531321344
log_loss for k =  3 is 0.0958800580948
log_loss for k =  5 is 0.0995466557335
log_loss for k =  7 is 0.107227274345
log_loss for k =  9 is 0.119239543547
log_loss for k =  11 is 0.133926642781
log_loss for k =  13 is 0.147643793967
log_loss for k =  15 is 0.159439699615
log_loss for k =  17 is 0.16878376444
log_loss for k =  19 is 0.178020728839

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



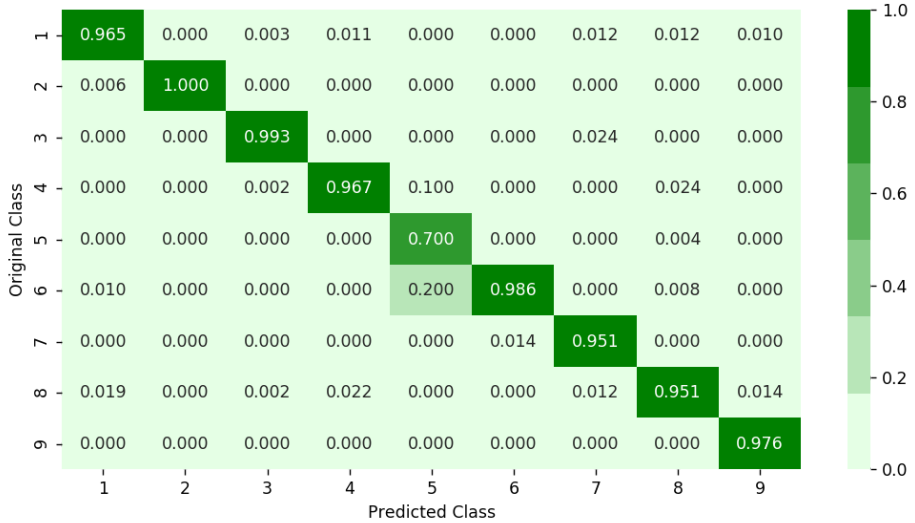

```
log loss for train data 0.0476773462198
log loss for cv data 0.0958800580948
log loss for test data 0.0894810720832
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix --
--------------------------------------------------
```

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

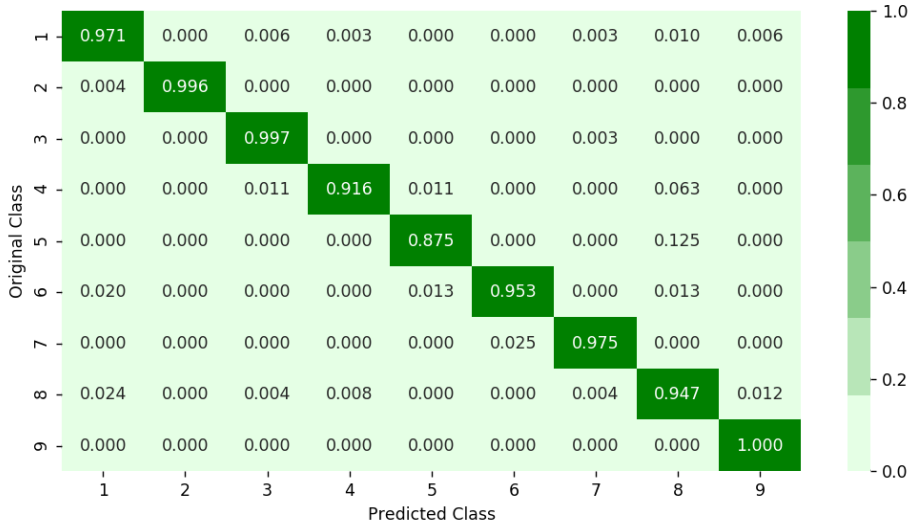



```
---------------------------------------------------- Precision matrix --
--------------------------------------------------
```

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




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

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




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

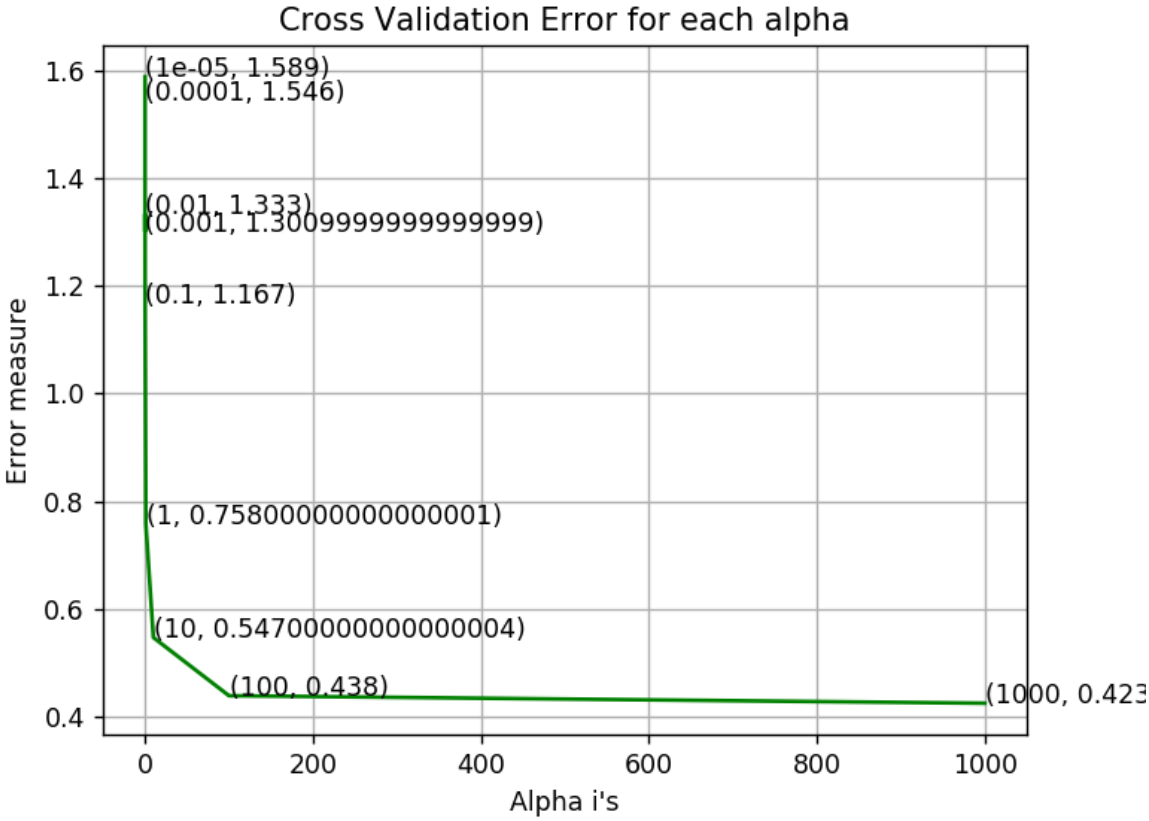
4.4.2 Logistic Regression

In [0]:

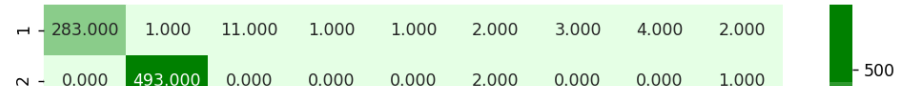
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_interce
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='
# 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 Gradien
# predict(X)    Predict class labels for samples in X.

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


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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  1e-05 is 1.58867274165
log_loss for c =  0.0001 is 1.54560797884
log_loss for c =  0.001 is 1.30137786807
log_loss for c =  0.01 is 1.33317456931
log_loss for c =  0.1 is 1.16705751378
```

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

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

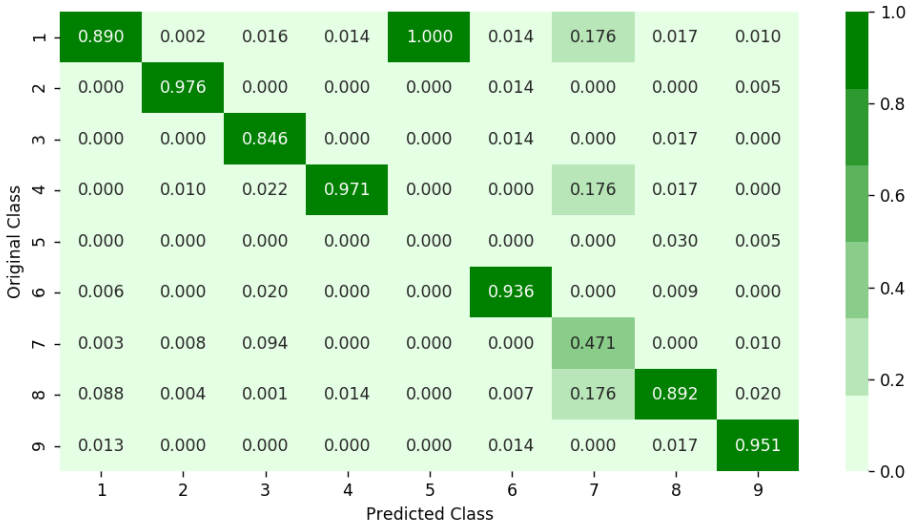



```
log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points  9.61361545538
---------------------------------------------------- Confusion matrix --
-----------------------------------------------

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



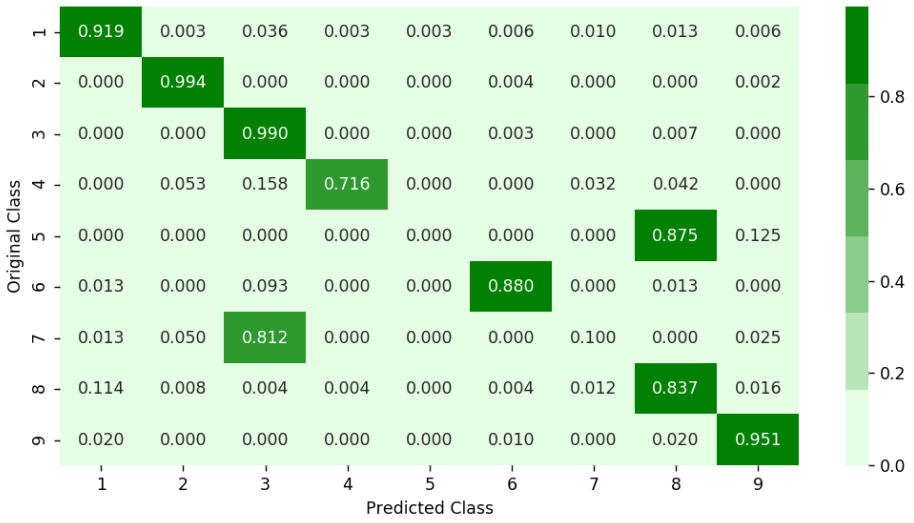

```
---------------------------------------------------- Precision matrix --
-----------------------------------------------
```

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




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

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




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

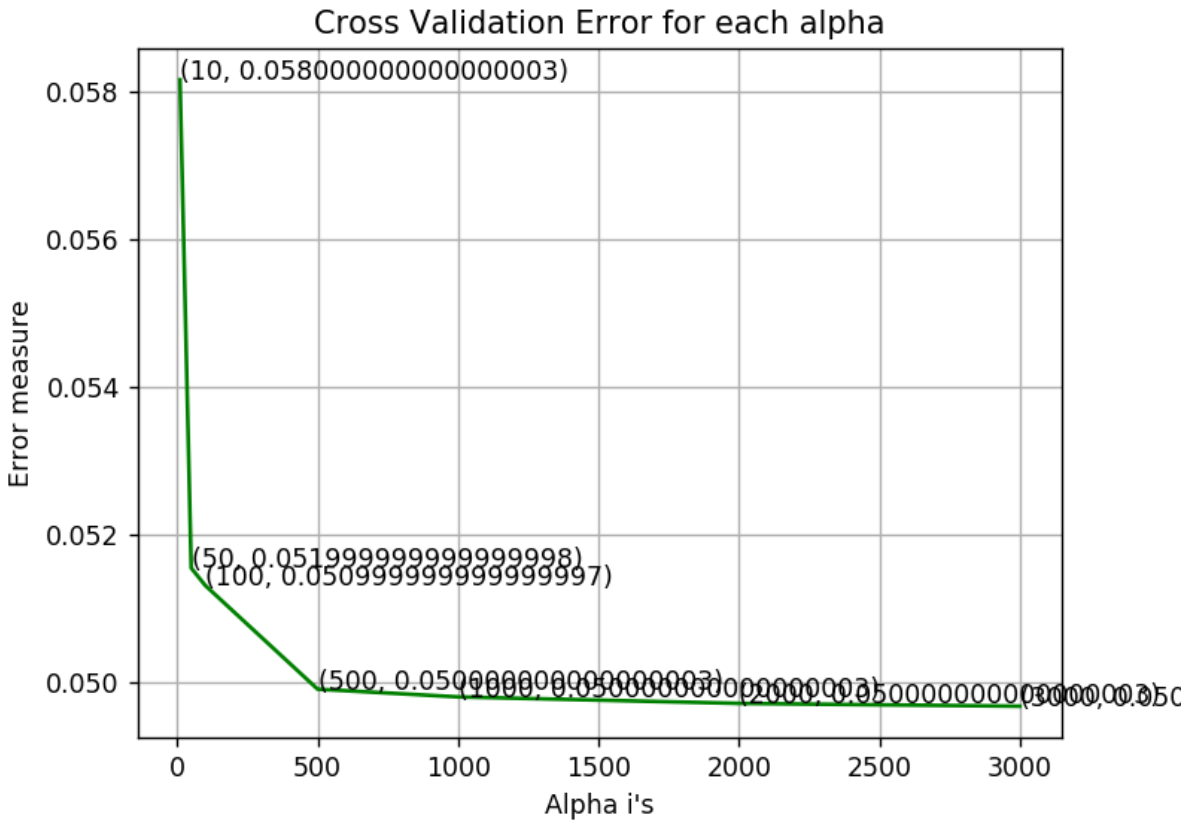
4.4.3 Random Forest Classifier

In [0]:

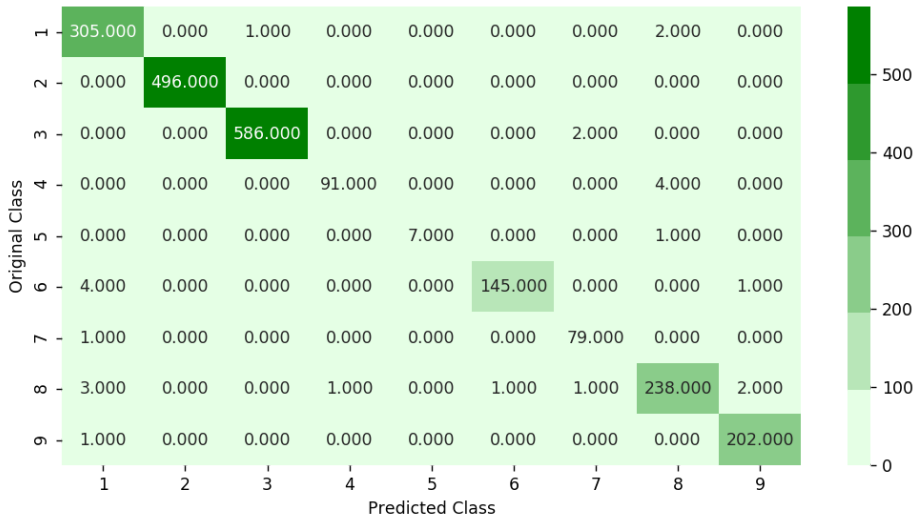
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_dep
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_no
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N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

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training da
# 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/lesson
# --------------------------------

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_, e

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

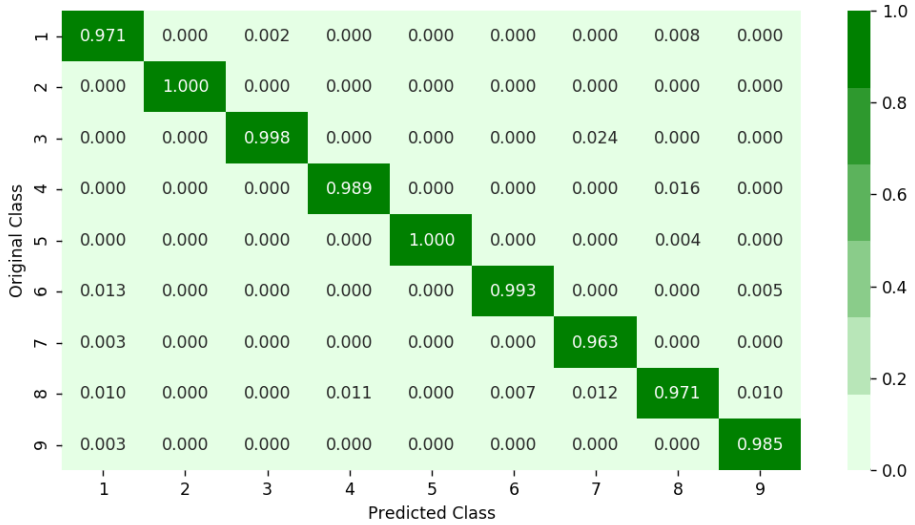
```
log_loss for c =  50 is 0.0515443148419
log_loss for c =  100 is 0.0513084973231
log_loss for c =  500 is 0.0499021761479
log_loss for c =  1000 is 0.0497972474298
log_loss for c =  2000 is 0.0497091690815
log_loss for c =  3000 is 0.0496706817633

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



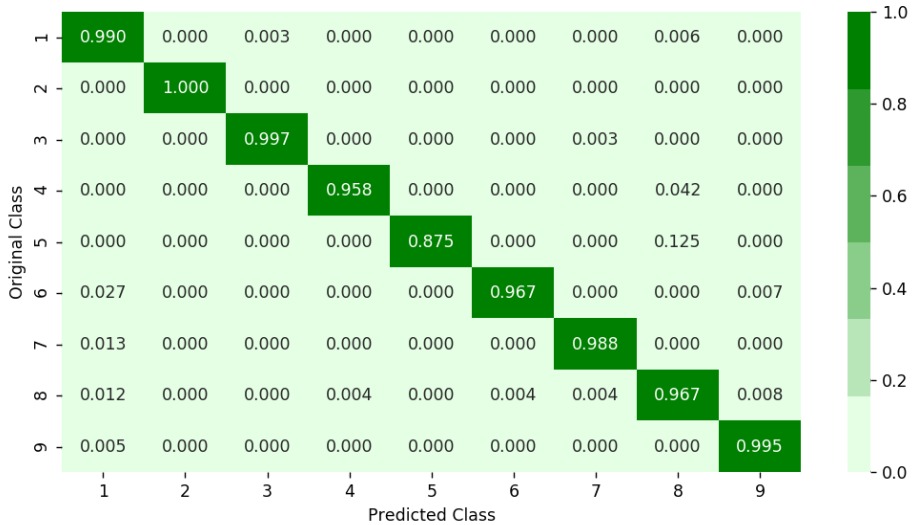

```
log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points  1.14995400184
---------------------------------------------------- Confusion matrix --
--------------------------------------------------

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



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

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




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

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



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

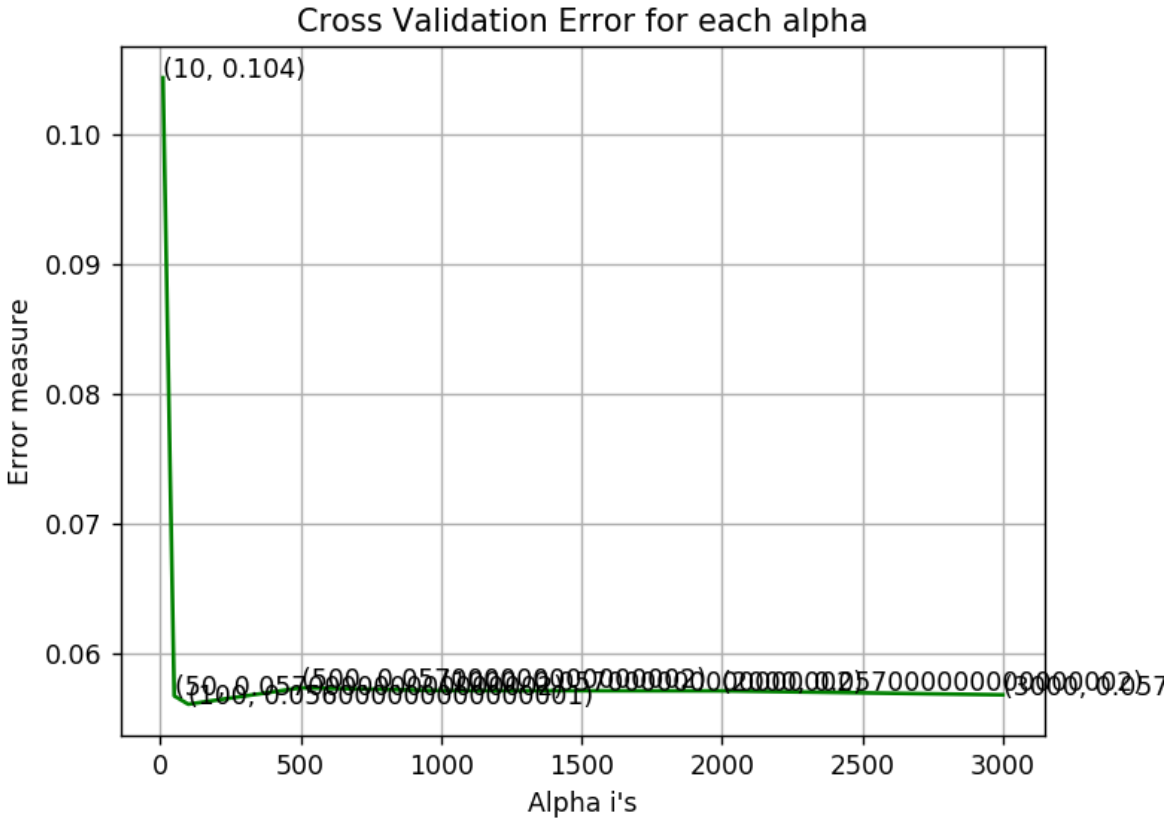
4.4.4 XgBoost Classifier

In [0]:

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

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

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

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_, e

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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

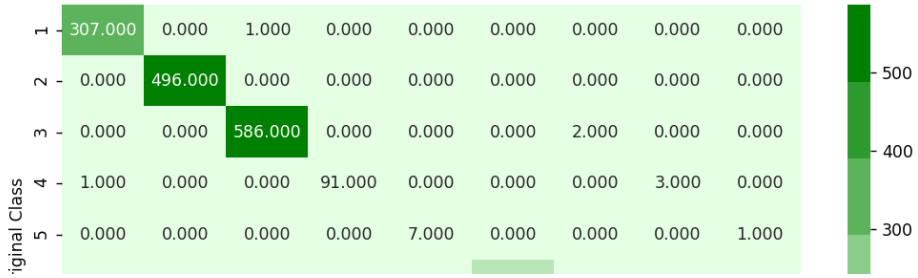
```
log_loss for c =  10 is 0.104344888454
log_loss for c =  50 is 0.0567190635611
log_loss for c =  100 is 0.056075038646
log_loss for c =  500 is 0.057336051683
log_loss for c =  1000 is 0.0571265109903
log_loss for c =  2000 is 0.057103406781
log_loss for c =  3000 is 0.0567993215778
```

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



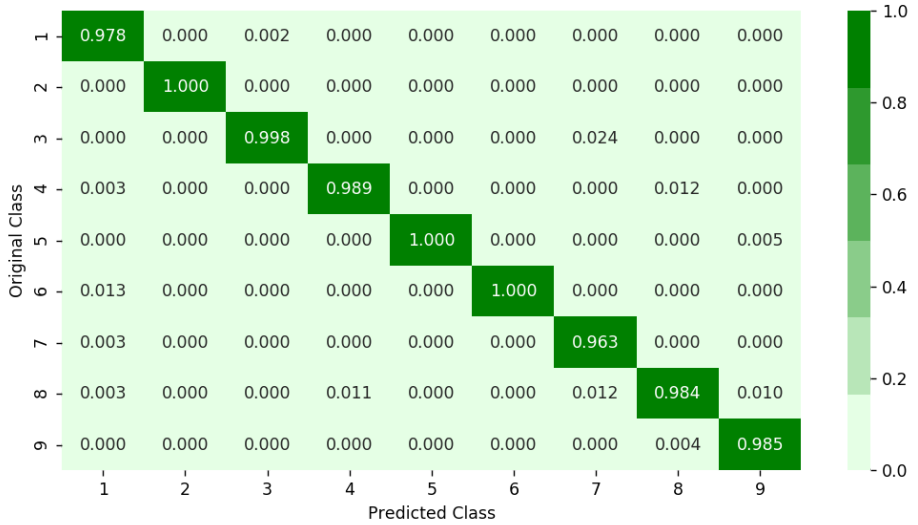

```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.05
6075038646
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
---------------------------------------------------- Confusion matrix --
--------------------------------------------------
```

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



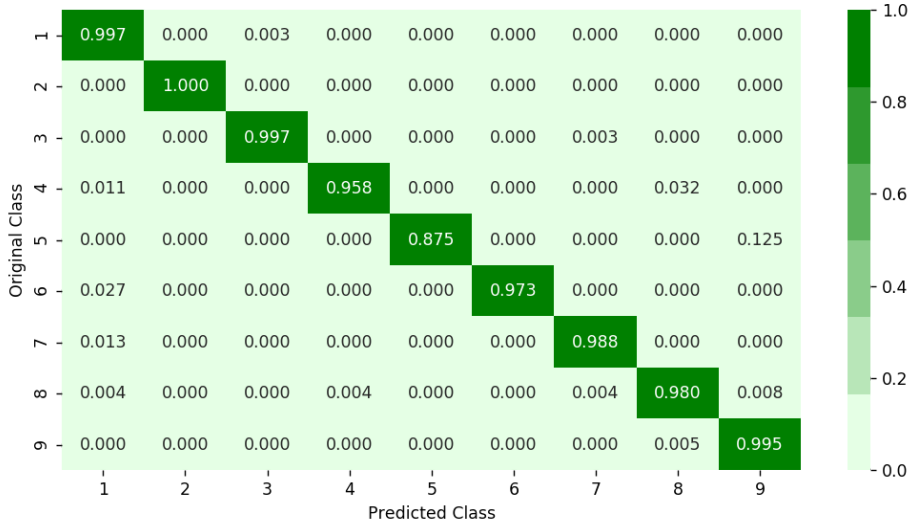

```
---------------------------------------------------- Precision matrix --
-------------------------------------------------
```

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




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

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




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

4.4.5 Xgboost Classifier with best hyperparameters

In [0]:

```
x_cfl=XGBClassifier()

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

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  1.1min remainin
g:   39.3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.3min remainin
g:   23.0s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  1.4min remainin
g:    9.2s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.3min finished
```

Out[163]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1,
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, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.
1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_dept
h': [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=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

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

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

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

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

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

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

4.5.1. Merging both asm and byte file features

In [18]:

```
result.head()
```

Out[18]:

	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.00205
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.00474
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.00507
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.00031
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.00014

5 rows × 260 columns

In [19]:

```
result_asm.head()
```

Out[19]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.eda
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	

5 rows × 54 columns

In [20]:

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

```
(10869, 260)
(10868, 54)
```

In [21]:

```
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[21]:

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

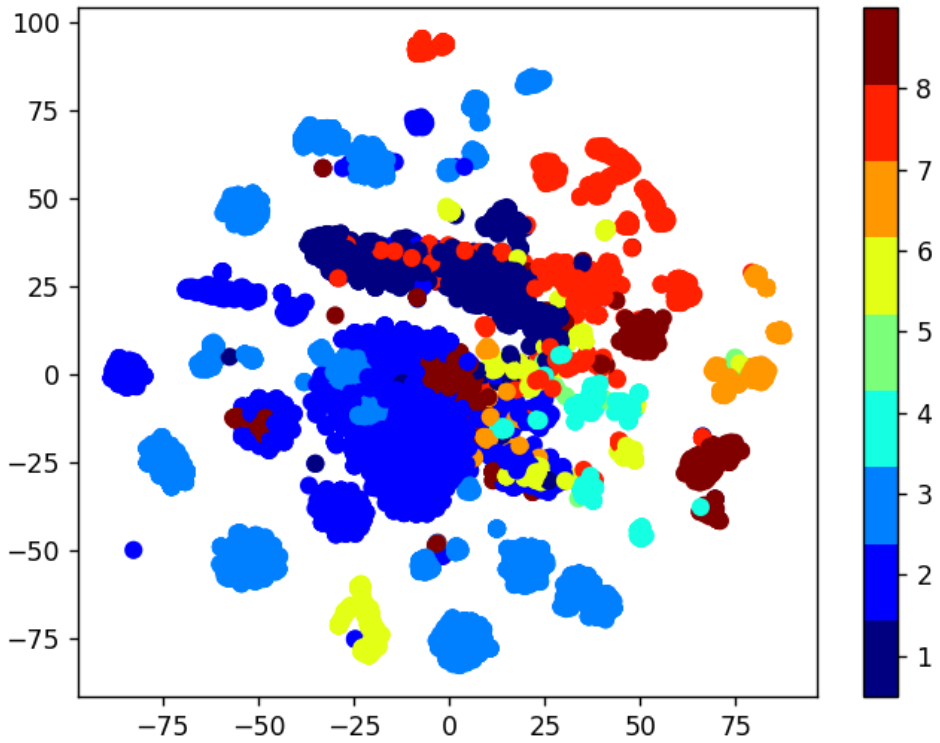
5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

In [0]:

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

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



4.5.3. Train and Test split

In [0]:

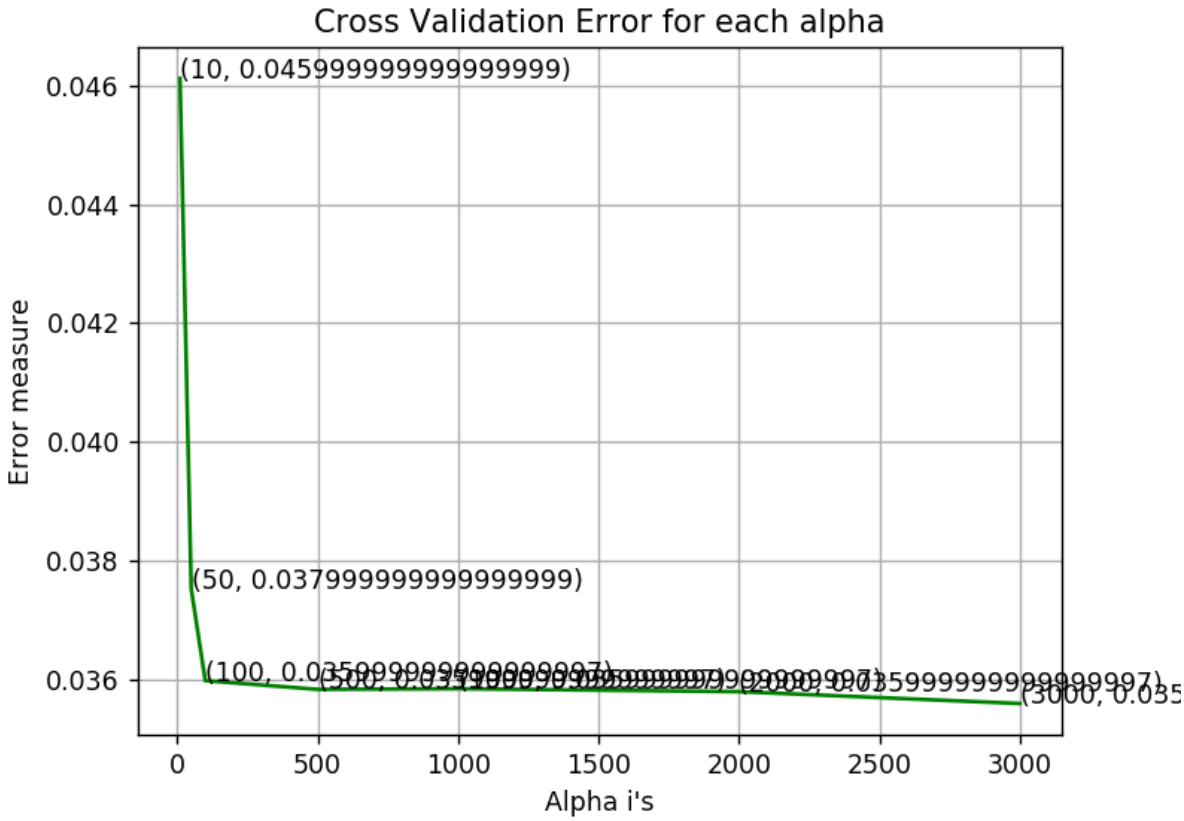
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,s
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_t
```

4.5.4. Random Forest Classifier on final features

In [0]:

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

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training da
# 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/lesson
# --------------------------------

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_,

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-
r_cfl.fit(X_train_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:",lo
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
```

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

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




```
For values of best alpha =  3000 The train log loss is: 0.016626761475
3
For values of best alpha =  3000 The cross validation log loss is: 0.0
355909487962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

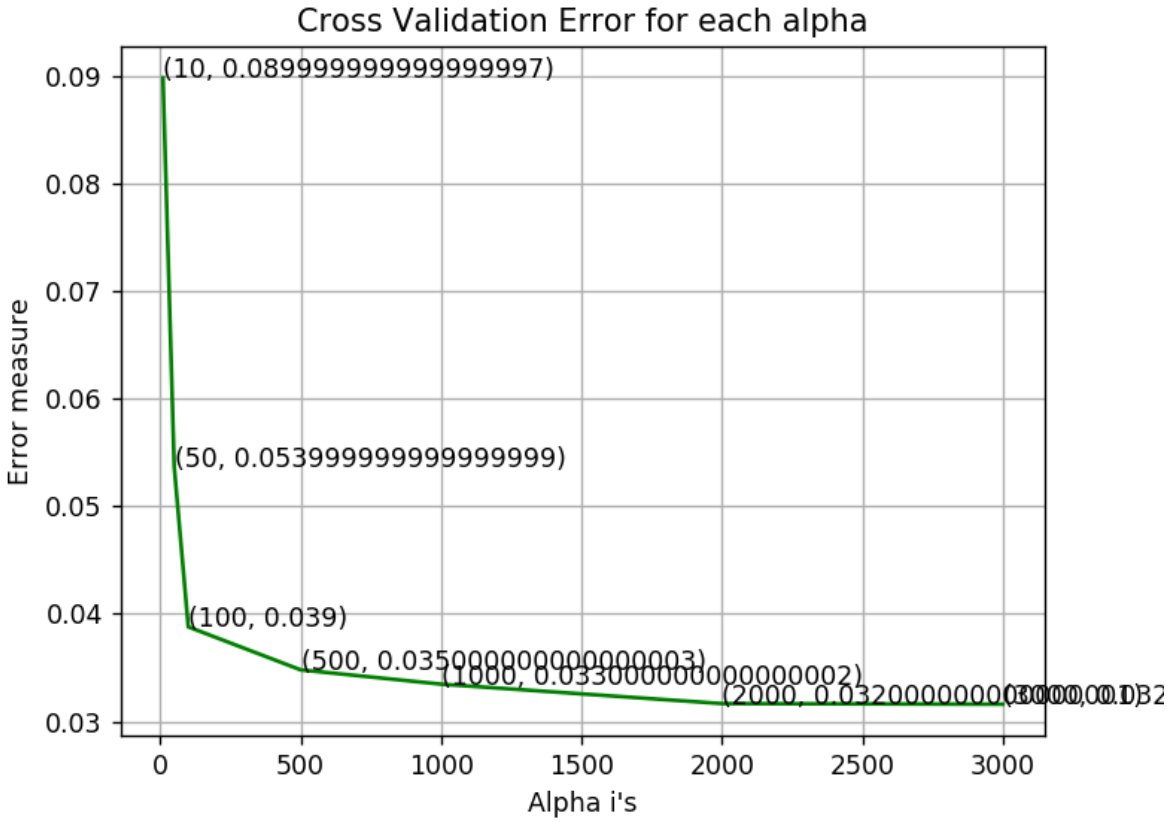
4.5.5. XgBoost Classifier on final features

In [0]:

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

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

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

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_,

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

```
log_loss for c =  10 is 0.0898979446265
log_loss for c =  50 is 0.0536946658041
```

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

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




```
For values of best alpha =  3000 The train log loss is: 0.011191880934
2
For values of best alpha =  3000 The cross validation log loss is: 0.0
315972694477
For values of best alpha =  3000 The test log loss is: 0.0323978515915
```

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

In [0]:

```
x_cfl=XGBClassifier()

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

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  4.5min remainin
g:  2.6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remainin
g:  1.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remainin
g:   44.5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

Out[187]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1,
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, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.
1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_dept
h': [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=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

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

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytr
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:",lo
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
For values of best alpha =  3000 The train log loss is: 0.012192283229
7
For values of best alpha =  3000 The cross validation log loss is: 0.0
344955487471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

5. Assignments

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

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

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

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

   https://askubuntu.com/a/341637

In [102]:

```
# list byteFiles directory
files = os.listdir('byteFiles')
len(files)
```

Out[102]:

```
10870
```

In [101]:

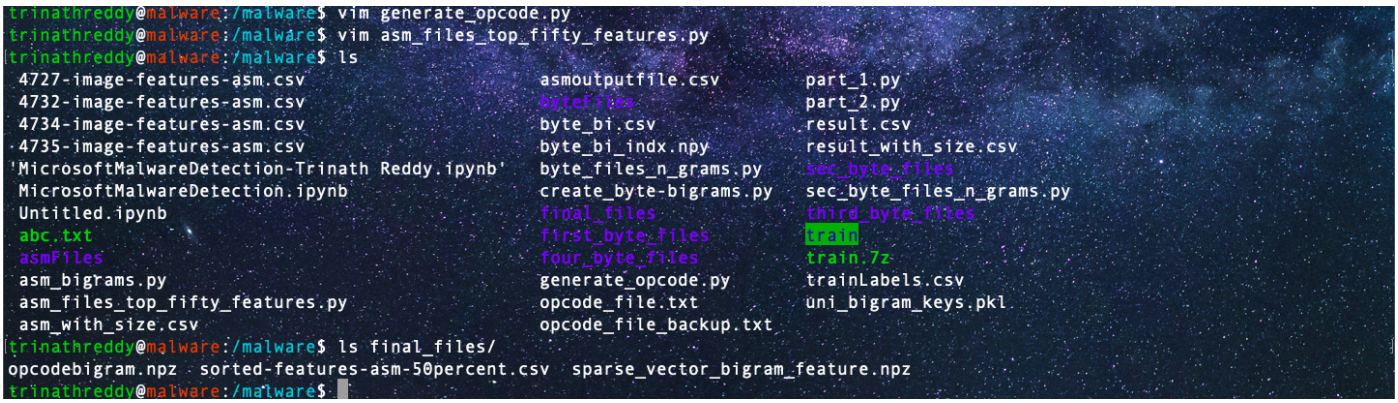
```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10868 entries, 0 to 10868
Columns: 260 entries, ID to Class
dtypes: float64(258), int64(1), object(1)
memory usage: 21.6+ MB
```

```
import os

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


byte_bigram_vocab = byte_bigram()

files = os.listdir('byteFiles/')

import scipy
import scipy.sparse

from tqdm import tqdm

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(lowercase=False, ngram_range=(2,2), vocabulary = byte_bigram_vocab)

feature_matrix = scipy.sparse.csr_matrix((len(files), len(byte_bigram_vocab)))

for index, file in tqdm(enumerate(files)):
    print(index)
    byte_file = open('byteFiles/'+file)
    feature_matrix[index,:] += scipy.sparse.csr_matrix(vectorizer.fit_transform([byte_file.read().replace('\n', ' ').lower()]))
    byte_file.close()

scipy.sparse.save_npz('bytebigram.npz', feature_matrix)
"create byte-bigrams.py" 34L, 1728C                                    34,0-1        Bot
```

```
import scipy
import os,codecs
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer


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

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

asmopcodebigram = asmopcodebigram()

def opcode_collect():
    op_file = open("opcode_file.txt", "w+")
    cnt =0
    total = len(os.listdir('asmFiles'))
    for asmfile in os.listdir('asmFiles'):
        cnt +=1
        print("Running file:", cnt, " of ",total)
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
```

```
#merging all generated csv files

labels = pd.read_csv('trainLabels.csv')
d1 = pd.read_csv('4727-image-features-asm.csv')
d2 = pd.read_csv('4732-image-features-asm.csv')
d3 = pd.read_csv('4734-image-features-asm.csv')
d4 = pd.read_csv('4735-image-features-asm.csv')
d4.shape

print(labels.head())
data = pd.concat([d1, d2, d3, d4])
data.shape


data.reset_index(drop=True, inplace=True)

sorted_train_data = data.sort_values(by='filename', axis=0, ascending=True, inplace=False)
sorted_train_labels = labels.sort_values(by='Id', axis=0, ascending=True, inplace=False)
X = sorted_train_data.iloc[:,1:]
y = np.array(sorted_train_labels.iloc[:,1])



# find the top 50 percent variance features, from 1000 -> 500 features
fsp = SelectPercentile(chi2, percentile=50)
X_new_50 = fsp.fit_transform(X,y)
X_new_50.shape

selected_names = fsp.get_support(indices=True)
selected_names = selected_names + 1
data_trimmed = sorted_train_data.iloc[:,selected_names]
```



In [103]:

```
! ls final_files/
```

```
opcodebigram.npz                    sparse_vector_bigram_feature.npz
sorted-features-asm-50percent.csv
```

In [104]:

```
import scipy
import scipy.sparse
from sklearn.preprocessing import normalize
byte_bigram_vect = normalize(scipy.sparse.load_npz('final_files/sparse_vector_bigram
```

In [105]:

```
byte_bigram_vect.shape
```

Out[105]:

```
(10868, 66049)
```

In [106]:

```
result_y = result['Class']
```

In [107]:

```
'''
    Function to get image features from asm files
'''
def get_image_features_from_asm_files(data, features, keep):
    random_forest_model = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    random_forest_model.fit(data, result_y)
    imp_feature_indx = np.argsort(random_forest_model.feature_importances_)[::-1]
    imp_value = np.take(random_forest_model.feature_importances_, imp_feature_indx[:
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    return imp_feature_indx[:keep]
```

In [37]:

```
'''
    Get bi-grams from byte files
'''

byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,1
def return_byte_bigram():
    byte_bigram_vocab = []
    for i, v in enumerate(byte_vocab.split(',')):
        byte_bigram_vocab[ v + ' ' +byte_vocab.split(',')[j]  for j in range(0, len(
    return byte_bigram_vocab

get_byte_bi_gram_keys = return_byte_bigram()
```

In [108]:

```
len(get_byte_bi_gram_keys)
```

Out[108]:

```
66049
```

In [109]:

```
# collect top 300 features from bigrams
getting_bi_grams_byte_files_indexs = get_image_features_from_asm_files(byte_bigram_v
```

In [110]:

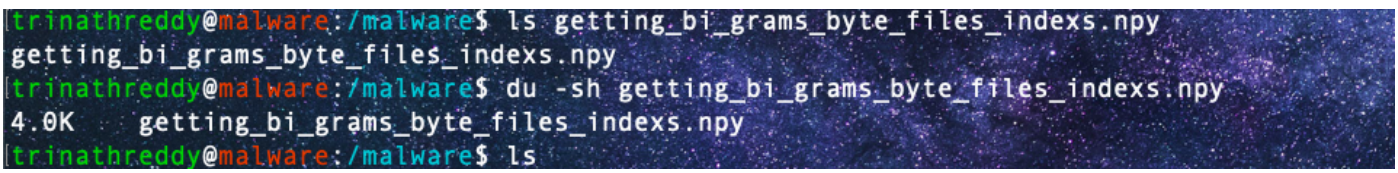
```
getting_bi_grams_byte_files_indexs
```

Out[110]:

```
array([    0,  1028,   255,    80, 65617, 65533, 65790,    48,   238,
           2,   106, 65019,   257,   771,   232,    83, 49087,   188,
       45232, 22591, 60716,    60,   240, 22334, 57568,   128,    28,
       22446, 23212, 65621, 22445, 49461,    72, 22183,  9252, 20815,
       20897,    82,    86,    51, 63220,    81, 15420, 65654, 65775,
       21411, 21049, 12850,     4, 20792, 21329,    92, 14649, 64762,
           3, 65719,    35, 65718, 21155,  1542,   131, 20641,   114,
       25186,   252, 12593,   180,   246,    22,   224, 50115, 37147,
          11, 36373, 21072, 13878,    87,   514,     8,   148,     7,
       59753, 14135, 19625, 59879, 51912, 65666, 59787,   248, 20923,
       65016, 33924, 50380, 12336,   156, 24415, 65581,    76, 13364,
       35932, 59810, 38291,     6, 55512,  4112,   199,    32,   108,
       52428,   216,    62, 20900,   129, 65582,  1285,  2056, 59658,
         160, 65664, 65535, 59759, 52683, 33408,    88, 21588, 28011,
       47288,   110,    50, 27242,   146,   228, 48830,   162,   186,
       65620, 22954,  2313, 28370, 61920, 62963,   116,   185, 65744,
       19530, 57823, 48059, 59757, 21418,  1064, 47545, 33024, 20646,
       63991, 40090,  7967, 65688, 63734,   192,   140,   205, 57825,
       65615, 53454, 33665, 61938, 65711,    90, 48761, 25038, 22442,
          89, 34950, 65676,    18, 60393, 21154, 62706, 65574,    33,
       22871,    15, 65713,   236,   204, 41120, 53711, 50254, 12848,
          12, 21845,   152, 42403, 49536, 13621, 65673,   194,  4206,
       13697, 65671, 30581,   213, 59737, 65622, 20647, 28124, 35980,
       32896, 65683, 65573, 39321, 65276,   184, 16563,  8355, 20903,
       27854,    94, 59839, 35721,   141, 16705,  2054, 22100, 59800,
       62195,   206, 29419, 65674, 65787,    14, 47029, 56026, 49085,
       21586, 39064, 63737, 21563,  1026,   244,  2570, 21259, 37006,
          36,    19, 65589,    44, 65560, 17544, 29929,    16, 24602,
        5481, 46260, 50372, 59699, 65536, 61421, 20904,   178, 54998,
       22102, 33407, 21157, 29812,    84, 21175,    93,    20, 59807,
         202, 65678,  5521, 13669, 25039, 23128, 49866, 65743, 43431,
       33866,    26,  5477, 59625, 22182, 22184, 23736,    10, 65669,
          25, 65626,    96, 26471, 28895, 36235,    23, 35588, 51398,
         247,     5, 35937])
```

In [111]:

```
# Save generated index file
np.save('getting_bi_grams_byte_files_indexs', getting_bi_grams_byte_files_indexs)
byte_bi_indxes = np.load('getting_bi_grams_byte_files_indexs.npy')
```



In [112]:

```
# Pick top 300 features
top_bi_grams_byte_files_indexs = np.zeros((10868, 0))
for i in byte_bi_indxes:
    sliced = byte_bigram_vect[:, i].todense()
    top_bi_grams_byte_files_indexs = np.hstack([top_byte_bi, sliced])
```

In [113]:

```
#Getting the top reqired details
top_bi_grams_byte_files_indexs_df = pd.DataFrame(top_bi_grams_byte_files_indexs, col
top_bi_grams_byte_files_indexs_df.to_csv('getting_bi_grams_byte_files_indexs.csv')
top_bi_grams_byte_files_indexs_df = pd.read_csv('getting_bi_grams_byte_files_indexs.
top_bi_grams_byte_files_indexs_df
```

Out[113]:

	00 00	04 00	00 ff	00 50	ff 52	fe ff	ff ff	00 30	00 e
0	0.003009	0.003541	0.009485	0.000890	0.006422	0.010339	0.006278	0.000186	0.00002
1	0.004008	0.004298	0.013528	0.001667	0.004281	0.003686	0.010547	0.000049	0.00004
2	0.001784	0.002208	0.003417	0.002023	0.000459	0.002616	0.001346	0.000382	0.00011
3	0.016892	0.007786	0.008815	0.012994	0.006677	0.007741	0.009912	0.002930	0.00321
4	0.004956	0.004512	0.007933	0.003447	0.001325	0.012779	0.004553	0.001577	0.00053
...	...	...	...	...	...	...	...	...	...
10863	0.002045	0.000256	0.000342	0.000939	0.000102	0.000334	0.000074	0.000147	0.00007
10864	0.003482	0.000693	0.000554	0.001715	0.000000	0.002282	0.000763	0.000049	0.00003
10865	0.002727	0.000395	0.000246	0.001553	0.001784	0.000176	0.000067	0.000044	0.00002
10866	0.003372	0.003968	0.003595	0.000987	0.007543	0.003669	0.003894	0.000109	0.00013
10867	0.001036	0.000448	0.000130	0.000550	0.002294	0.000474	0.000131	0.000120	0.00005

10868 rows × 300 columns

In [114]:

```
# checking the top bi-gram details
result_x = top_bi_grams_byte_files_indexs_df
result_y = data_y
result_x.tail()
```

Out[114]:

	00 00	04 00	00 ff	00 50	ff 52	fe ff	ff ff	00 30	00 e
10863	0.002045	0.000256	0.000342	0.000939	0.000102	0.000334	0.000074	0.000147	0.00007
10864	0.003482	0.000693	0.000554	0.001715	0.000000	0.002282	0.000763	0.000049	0.00003
10865	0.002727	0.000395	0.000246	0.001553	0.001784	0.000176	0.000067	0.000044	0.00002
10866	0.003372	0.003968	0.003595	0.000987	0.007543	0.003669	0.003894	0.000109	0.00013
10867	0.001036	0.000448	0.000130	0.000550	0.002294	0.000474	0.000131	0.000120	0.00005

5 rows × 300 columns

In [117]:

```
#splitting the data into train and test details
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_t
X_train.shape, y_train.shape
```

Out[117]:

```
((8694, 300), (8694,))
```

In [118]:

```
result_x.shape
```

Out[118]:

```
(10868, 300)
```

Bi-GRAMS BYTE FILES RandomForestClassifier

In [121]:

```
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('Bi-grams for byte files log_loss with c = ',alpha[i],'is',cv_log_error_a


# 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()
```

```
Bi-grams for byte files log_loss with c =  10 is 0.08267341126639815
Bi-grams for byte files log_loss with c =  50 is 0.06331342796062295
Bi-grams for byte files log_loss with c =  100 is 0.06205143455650585
Bi-grams for byte files log_loss with c =  500 is 0.062388746329948244
Bi-grams for byte files log_loss with c =  1000 is 0.06318184279924256
Bi-grams for byte files log_loss with c =  2000 is 0.06283061462565234
Bi-grams for byte files log_loss with c =  3000 is 0.06270454305789515
```

In [122]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=500,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 = ', 500, "The train log loss is:",log_loss(y_train
print ('For values of best alpha = ', 500, "The train log loss is:",log_loss(y_train
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', 500, "The cross validation log loss is:",log_lo
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', 500, "The test log loss is:",log_loss(y_test_me
```

```
Bi-grams for byte files of best alpha =  500 & train log loss is: 1.91
59980033560353
Bi-grams for byte files of best alpha =  500 & cross validation log lo
ss is: 1.9010450922430568
Bi-grams for byte files of best alpha =  500 & test log loss is: 1.900
0374593165934
CPU times: user 6min 29s, sys: 3.2 s, total: 6min 32s
Wall time: 56.7 s
```

In [125]:

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

```
Number of misclassified points  72.95308187672494
---------------------------------------------------- Confusion matrix --
----------------------------------------------------

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


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

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

In [29]:

```
from prettytable import PrettyTable
prty_tble = PrettyTable(["MODEL-TYPE", "HYPRE-PARAMETRE", "TRAIN LOSS", "VAL-LOSS","
prty_tble.add_row(["RandomForestClassifier (best param)","500","1.9159980033560353",
print(prty_tble)
```

```
+-----------------------------------+-----------------+-------------
-------+--------------------+--------------------+
|              MODEL-TYPE             | HYPRE-PARAMETRE |     TRAIN LO
SS     |      VAL-LOSS      |     TEST-LOSS      |
+-----------------------------------+-----------------+-------------
-------+--------------------+--------------------+
| RandomForestClassifier (best param) |       500       | 1.9159980033
560353 | 1.9010450922430568 | 1.9000374593165934 |
+-----------------------------------+-----------------+-------------
-------+--------------------+--------------------+
```

Bi-GRAMS BYTE FILES XGBClassifier

In [126]:

```
cv_log_error_array=[]
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('Bi-grams for byte files log_loss with c = ',alpha[i],'is',cv_log_error_a


# 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()
```

```
Bi-grams for byte files log_loss with c =  10 is 0.1240123654208798
Bi-grams for byte files log_loss with c =  50 is 0.06702316457133047
Bi-grams for byte files log_loss with c =  100 is 0.047847134504250573
Bi-grams for byte files log_loss with c =  500 is 0.04310767064887841
Bi-grams for byte files log_loss with c =  1000 is 0.04168655613461821
8
Bi-grams for byte files log_loss with c =  2000 is 0.04095815656011010
6
Bi-grams for byte files log_loss with c =  3000 is 0.04096654702317071
34
```

In [127]:

```
%%time
# plt.close()
x_cfl=XGBClassifier(n_estimators=2000,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 ('Bi-grams for byte filesof best alpha = ', 2000, "& train log loss is:",log_l
predict_y = sig_clf.predict_proba(X_cv_merge)
print('Bi-grams for byte filesof best alpha = ', 2000, "& cross validation log loss
predict_y = sig_clf.predict_proba(X_test_merge)
print('Bi-grams for byte filesbest alpha = ', 2000, "& test log loss is:",log_loss(y

# plt.close()
# plot_confusion_matrix(y_test_merge,sig_clf.predict(X_test_merge))
```

```
Bi-grams for byte files of best alpha =  2000 & train log loss is: 0.0
141584358235132147
Bi-grams for byte files of best alpha =  2000 & cross validation log l
oss is: 0.040945431586010106
Bi-grams for byte files of best alpha =  2000 & test log loss is: 0.05
8356345248527767
CPU times: user 28min 17s, sys: 5.85 s, total: 28min 23s
Wall time: 28min 23s
```

In [127]:

```
data_reduced = pd.read_csv('final_files/ASM-TOP-IMAGE-50-PERCENT.csv')
data_reduced.shape
```

Out[127]:

```
(10868, 501)
```

In [128]:

```
data_reduced.rename(columns={'filename': 'ID'}, inplace=True)
```

In [129]:

```
data_reduced.head()
```

Out[129]:

	ID	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23	ASM
0	01IsoiSMh5gxyDYTl4CB	116	120	116	9	32	32	32	
1	01SuzwMJEIXsK7A8dQbl	69	68	69	48	9	9	13	
2	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13	
3	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13	
4	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13	

5 rows × 501 columns

In [130]:

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

Out[130]:

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

5 rows × 757 columns

In [131]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,s
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_t
```

In [30]:

```
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('IMAGE FEATURES for asm files log_loss c = ',alpha[i],'is',cv_log_error_a


# 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()
```

```
IMAGE FEATURES for asm files log_loss with c =  10 is 0.03198343094752
871
IMAGE FEATURES for asm files log_loss with c =  50 is 0.03185567636919
456
IMAGE FEATURES for asm files log_loss with c =  100 is 0.0300077459186
41725
IMAGE FEATURES for asm files log_loss with c =  500 is 0.0299672099312
8468
IMAGE FEATURES for asm files log_loss with c =  1000 is 0.029971575881
27103
IMAGE FEATURES for asm files log_loss with c =  2000 is 0.030281356953
265455
IMAGE FEATURES for asm files log_loss with c =  3000 is 0.030274656167
71141
```

In [31]:

```
%%time
plt.close()

r_cfl=RandomForestClassifier(n_estimators=500,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 ('IMAGE FEATURES for asm files log_loss with best alpha = ', 500, "& train log
predict_y = sig_clf.predict_proba(X_cv_merge)

print('IMAGE FEATURES for asm files log_loss best alpha = ', 500, " & cross validati
predict_y = sig_clf.predict_proba(X_test_merge)

print('IMAGE FEATURES for asm files log_loss best alpha = ', 500, "& test log loss i
```

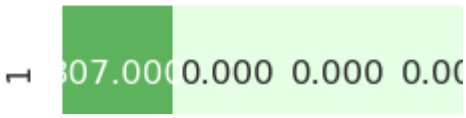
```
IMAGE FEATURES for asm files log_loss with best alpha =  500 & train l
og loss is: 0.01339184484689122
IMAGE FEATURES for asm files log_loss with best alpha =  500 & cross v
alidation log loss is: 0.026546743573283283
IMAGE FEATURES for asm files log_loss with best alpha =  500 & test lo
g loss is: 0.03407080810191573
CPU times: user 2min 28s, sys: 4.08 s, total: 2min 32s
Wall time: 28.2 s
```

In [133]:

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

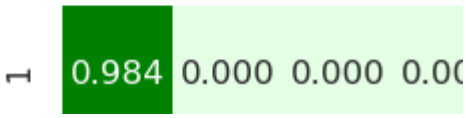
```
Number of misclassified points  0.5059797608095675
---------------------------------------------------- Confusion matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>



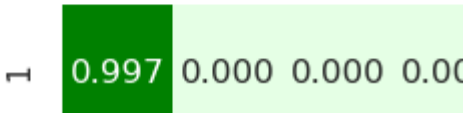

```
------------------------------------------------------ Precision matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

In [32]:

```
cv_log_error_array=[]
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('ASM IMAGE + BYTE UNI-GRAM with 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()
```

```
ASM IMAGE + BYTE UNI-GRAM with c =  10 is 0.07824971546661065
ASM IMAGE + BYTE UNI-GRAM with c =  50 is 0.0329859879132399546
ASM IMAGE + BYTE UNI-GRAM with c =  100 is 0.026713759879898451427
ASM IMAGE + BYTE UNI-GRAM with c =  500 is 0.0253230456938319807
ASM IMAGE + BYTE UNI-GRAM with c =  1000 is 0.025764903786663419
ASM IMAGE + BYTE UNI-GRAM with c =  2000 is 0.02616496536273984
ASM IMAGE + BYTE UNI-GRAM with c =  3000 is 0.026065226522220864
```

In [33]:

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=500,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 ('ASM IMAGE + BYTE UNI-GRAM with best alpha = ', 500, "& train log loss is:",lo
predict_y = sig_clf.predict_proba(X_cv_merge)
print('ASM IMAGE + BYTE UNI-GRAM with best alpha = ', 500, "& cross validation log lo
predict_y = sig_clf.predict_proba(X_test_merge)
print('ASM IMAGE + BYTE UNI-GRAM with best alpha = ', 500, "& test log loss is:",log_
```

```
ASM IMAGE + BYTE UNI-GRAM with best alpha =  500 & train log loss is:
0.011985457054981705
ASM IMAGE + BYTE UNI-GRAM with best alpha =  500 & cross validation lo
g loss is: 0.026065224622220864
ASM IMAGE + BYTE UNI-GRAM with best alpha =  500 & test log loss is:
0.0341661065666613246
CPU times: user 1h 26min 35s, sys: 0 ns, total: 1h 26min 35s
Wall time: 1h 26min 35s
```

In [134]:

```
%%time
#model with XGBClassifier
x_cfl=XGBClassifier(n_estimators=1000,max_depth=3,learning_rate=0.15,colsample_bytre
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)
```

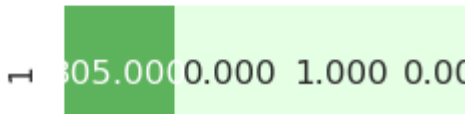
```
CPU times: user 21min 43s, sys: 357 ms, total: 21min 43s
Wall time: 21min 44s
```

In [135]:

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

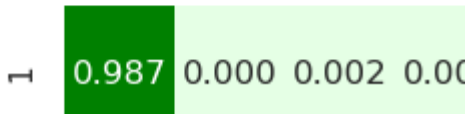
```
Number of misclassified points  0.41398344066237347
---------------------------------------------------- Confusion matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>



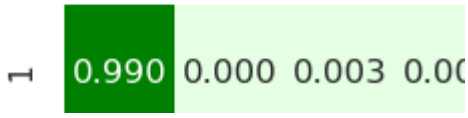

```
---------------------------------------------------- Precision matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

Implement asm unigram + asm extracted image features

In [136]:

```
#get the required shapes
print(data_reduced.shape)
print(result_asm.shape)
```

```
(10868, 501)
(10868, 54)
```

In [137]:

```
result_x = pd.merge(result_asm, data_reduced,on='ID', how='left')
#only storing required details
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
#echo resluts
result_x.head()
```

Out[137]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	A
0	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	0.0	...	
1	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	0.0	...	
2	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	0.0	...	
3	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	0.0	...	
4	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	0.0	...	

5 rows × 549 columns

In [141]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,s
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_t
```

In [34]:

```
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('ASM UNI-GRAMS + IMAGE FEATURES WITH c = ',alpha[i],'is',cv_log_error_arr


# 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()
```

```
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  10 is 0.021275775701326152
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  50 is 0.0198955899264623286
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  100 is 0.019639514435224526
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  500 is 0.019132586421543085
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  1000 is 0.01893897948023988
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  2000 is 0.018992036016380148
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  3000 is 0.01906281256167374
```

In [87]:

```
%%time
plt.close()
# model for RandomForestClassifier
r_cfl=RandomForestClassifier(n_estimators=1000,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 ('ASM UNI-GRAMS + IMAGE FEATURES with best alpha = ', 1000, "& train log loss
predict_y = sig_clf.predict_proba(X_cv_merge)
print('ASM UNI-GRAMS + IMAGE FEATURES with best alpha = ', 1000, "& cross validatior
predict_y = sig_clf.predict_proba(X_test_merge)
print('ASM UNI-GRAMS + IMAGE FEATURES with best alpha = ', 1000, "& test log loss is
```

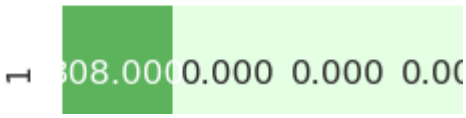
```
ASM UNI-GRAMS + IMAGE FEATURES with best alpha =  1000 & train log los
s is: 0.010999728489891
ASM UNI-GRAMS + IMAGE FEATURES with best alpha =  1000 & cross validat
ion log loss is: 0.01680704293117681
ASM UNI-GRAMS + IMAGE FEATURES with best alpha =  1000 & test log loss
is: 0.02945939131323688
CPU times: user 2min 58s, sys: 9.76 s, total: 3min 7s
Wall time: 3min 7s
```

In [143]:

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

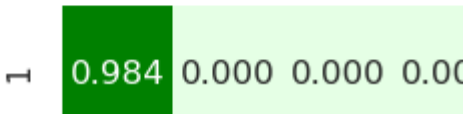
```
Number of misclassified points  0.5519779208831647
---------------------------------------------------- Confusion matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>



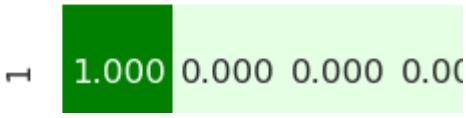

```
---------------------------------------------------- Precision matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

In [36]:

```
cv_log_error_array=[]
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('ASM UNI-GRAMS + IMAGE FEATURES WITH c = ',alpha[i],'is',cv_log_error_arr

# 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()
```

```
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  10 is 0.06268899176536516
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  50 is 0.035508785513269654
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  100 is 0.02861795575938164
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  500 is 0.027898979315273183
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  1000 is 0.027231393234936435
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  2000 is 0.02715772002665156
ASM UNI-GRAMS + IMAGE FEATURES WITH c =  3000 is 0.02701750007288518
```

In [37]:

```
%time
lt.close()
 Model for XGBClassifier
_cfl=XGBClassifier(n_estimators=3000,nthread=-1)
_cfl.fit(X_train_merge,y_train_merge,verbose=True)
ig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
ig_clf.fit(X_train_merge, y_train_merge)

redict_y = sig_clf.predict_proba(X_train_merge)
rint ('ASM UNI-GRAMS + IMAGE FEATURES WITH of best alpha = ', 3000, "& train log loss

redict_y = sig_clf.predict_proba(X_cv_merge)
rint('ASM UNI-GRAMS + IMAGE FEATURES WITH of best alpha = ', 3000, "& cross validati

redict_y = sig_clf.predict_proba(X_test_merge)
rint('ASM UNI-GRAMS + IMAGE FEATURES WITH of best alpha = ', 3000, "& test log loss
```

```
ASM UNI-GRAMS + IMAGE FEATURES WITH of best alpha =  3000 & train log
loss is: 0.008320609361993012
ASM UNI-GRAMS + IMAGE FEATURES WITH of best alpha =  3000 & cross vali
dation log loss is: 0.02725750007288518
ASM UNI-GRAMS + IMAGE FEATURES WITH of best alpha =  3000 & test log l
oss is: 0.02159222942854402
CPU times: user 51min, sys: 7.8 ms, total: 58.8min
Wall time: 58.8min
```

ASM UNI + BYTE BI + ASM IMAGE FEATURES

In [157]:

```
#Displaying shapes
print(data_reduced.shape)
print(result_asm.shape)
print(result.shape)
```

```
(10868, 501)
(10868, 54)
(10868, 260)
```

In [158]:

```
#checking required details
data_reduced.columns, result_asm.columns
```

Out[158]:

```
(Index(['ID', 'ASM_1', 'ASM_3', 'ASM_4', 'ASM_14', 'ASM_20', 'ASM_21',
'ASM_23',
        'ASM_24', 'ASM_25',
        ...
        'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_99
4',
        'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
       dtype='object', length=501),
 Index(['ID', 'HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bs
s:',
        '.rdata:', '.edata:', '.rsrc:', '.tls:', '.reloc:', '.BSS:',
'.CODE',
        'jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'su
b', 'inc',
        'dec', 'add', 'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'sh
l', 'ror',
        'rol', 'jnb', 'jz', 'rtn', 'lea', 'movzx', '.dll', 'std::', ':
dword',
        'edx', 'esi', 'eax', 'ebx', 'ecx', 'edi', 'ebp', 'esp', 'eip',
'Class',
        'size'],
       dtype='object'))
```

In [159]:

```
#Required details
result_x = pd.merge(result_asm, data_reduced, on='ID', how='left')

result_x = pd.merge(result_x, result.drop('size', axis=1), on='ID', how='left')

result_x = result_x.drop(['ID','rtn','.BSS:','.CODE'], axis=1)

result_x.head()
```

Out[159]:

	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	
0	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	0.0	...	0.
1	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	0.0	...	0.
2	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	0.0	...	0.
3	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	0.0	...	0.
4	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	0.0	...	0.

5 rows × 808 columns

In [160]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,s
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_t
```

ASM UNI + BYTE BI + ASM IMAGE FEATURES

In [39]:

```
%%time
plt.close()

cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in [10,50,100,500,1000,2000,3000]:

    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_,

for i in range(len(cv_log_error_array)):
    print ('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE for c = ',alpha[i],'is',cv_log_error_a


# 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()
```

```
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE for c =  10 is 0.01123057551993269
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE  for c =  50 is 0.011319907320875825
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE  for c =  100 is 0.01145085038053396
2
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE  for c =  500 is 0.01170124656230979
8
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE  for c =  1000 is 0.0116612853655793
24
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE  for c =  2000 is 0.0116648935172601
47
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE  for c =  3000 is 0.0115187359574614
27
```

In [40]:

```
%%time
plt.close()
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-

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 ('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', alpha[best_alpha], "& t

predict_y = sig_clf.predict_proba(X_cv_merge)
print('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', alpha[best_alpha], "& cr

predict_y = sig_clf.predict_proba(X_test_merge)
print('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', alpha[best_alpha], "& te

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

```
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  10 & train log loss
is: 0.009189865009960088
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  10 & cross validati
on log loss is: 0.01123057551993269
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  10 & test log loss
is: 0.012817156034718172
CPU times: user 2.18 s, sys: 105 ms, total: 2.29 s
Wall time: 2.29 s
```

In [82]:

```
%%time
plt.close()

cv_log_error_array=[]
for i in [10,50,100,500,1000,2000,3000]:
    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_,

for i in range(len(cv_log_error_array)):
    print ('XGBClassifier 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()
```

```
XGBClassifier for c =  10 is 0.01124057551993269
XGBClassifier for c =  50 is 0.011219907320875825
XGBClassifier for c =  100 is 0.011450850380533962
XGBClassifier for c =  500 is 0.011781246562309798
XGBClassifier for c =  1000 is 0.011861285365579324
XGBClassifier for c =  2000 is 0.011764893517260147
XGBClassifier for c =  3000 is 0.011518735957461427
```

In [ ]:

```
# x_cfl=XGBClassifier(n_estimators=1000,max_depth=3,learning_rate=0.03,colsample_byt
# 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)
```

In [41]:

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=100,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 ('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', 100, "& train log loss

predict_y = sig_clf.predict_proba(X_cv_merge)
print('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', 100, "& cross validation

predict_y = sig_clf.predict_proba(X_test_merge)
print('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', 100, "& test log loss is
```

```
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  100 & train log los
s is: 0.007329945358658675
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  100 & cross validat
ion log loss is: 0.008010114827994947
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  100 & test log loss
is: 0.012309688611571683
CPU times: user 6min 10s, sys: 424 ms, total: 6min 10s
Wall time: 6min 10s
```

In [43]:

```
%%time
x_cfl=XGBClassifier( n_estimators=100, max_depth=3, learning_rate=0.03, colsample_by

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)
```

```
CPU times: user 19min 13s, sys: 260 ms, total: 19min 13s
Wall time: 19min 13s
```

In [42]:

```
alpha=[10,50,100,500,1000,2000,3000]
best_alpha = 2
predict_y = sig_clf.predict_proba(X_train_merge)
print ('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', 100, "& train log loss

predict_y = sig_clf.predict_proba(X_cv_merge)
print('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', 100, "& cross validatior

predict_y = sig_clf.predict_proba(X_test_merge)
print('(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha = ', 100, "& test log loss is
```

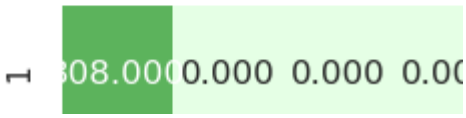
```
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  100 & train log los
s is: 0.0068621379955847535
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  100 & cross validat
ion log loss is: 0.009061602545277658
(ASM +BYTE - UNIGRAMS)+ ASM IMAGE of best alpha =  100 & test log loss
is: 0.011034226867838054
```

In [164]:

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

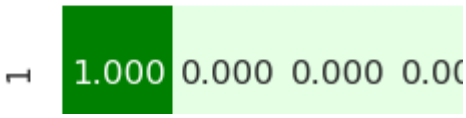
```
Number of misclassified points  0.09199632014719411
---------------------------------------------------- Confusion matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>



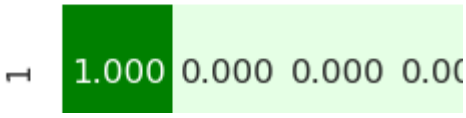

```
---------------------------------------------------- Precision matrix --
-----------------------------------------------
```

<IPython.core.display.Javascript object>




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

<IPython.core.display.Javascript object>




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

In [93]:

```
from prettytable import PrettyTable
prty_tble = PrettyTable(["MODEL-TYPE", "HYPRE-PARAMETRE", "TRAIN LOSS", "VAL-LOSS","
prty_tble.add_row(["Random Forest","10","0.009189865009960088","0.01123057551993269"
prty_tble.add_row(["XGBClassifier","100","0.007329945358658675","0.00801011482799494
prty_tble.add_row(["XGBClassifier(best param)","100","0.0068621379955847535","0.0090
print(prty_tble)
```

```
+--------------------------+-----------------+-----------------------
+----------------------+----------------------+
|          MODEL-TYPE         | HYPRE-PARAMETRE |       TRAIN LOSS
|       VAL-LOSS       |      TEST-LOSS       |
+--------------------------+-----------------+-----------------------
+----------------------+----------------------+
|       Random Forest       |        10       |  0.009189865009960088
| 0.01123057551993269  | 0.012817156034718172 |
|       XGBClassifier       |       100       |  0.007329945358658675
| 0.008010114827994947 | 0.012309688611571683 |
| XGBClassifier(best param) |       100       | 0.0068621379955847535
| 0.009061602545277658 | 0.011034226867838054 |
+--------------------------+-----------------+-----------------------
+----------------------+----------------------+
```

Code Reference's:

1. https://medium.com/kaggle-blog/microsoft-malware-winners-interview-1st-place-no-to-overfitting-ee0b664bfb4c
2. https://towardsdatascience.com/microsoft-malware-prediction-and-its-9-million-machines-22e0fe8c80c8
3. (Optional) https://www.researchgate.net/profile/Mfevzi_Esen/publication/340535548_Multiclass_Classification_with_Decision_Trees_Naive_Bayes_and_Logistic_Regression_An_Application_with_R/links/5e8f2767299bf130798a19fb/Multiclass-Classification-with-Decision-Trees-Naive-Bayes-and-Logistic-Regression-An-Application-with-R.pdf#page=101
4. https://www.kaggle.com/c/malware-classification/discussion/12389
5. https://www.kaggle.com/c/malware-classification/discussion/12818
6. Some other references from online resources code references
7. Code reference also from aaic notes