Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

.asm file

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

.bytes file

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine 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=VLQTRILGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

# File operations:
import shutil
import os
import codecs# this is used for file operations

# Data structures:
import pandas as pd
import pandas_profiling as pp
import numpy as np
import random as r

# Visualizations:
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.manifold import TSNE

# Code speed improvement:
from multiprocessing import Process# this is used for multithreading
import multiprocessing as mp

# Transformations:
from sklearn import preprocessing

# Models:
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# CV:
from sklearn.model_selection import train_test_split, RandomizedSearchCV

# Performance Metrics:
from sklearn.metrics import confusion_matrix, log_loss, accuracy_score

# Object persistance:
import pickle
from sklearn.externals import joblib

from tqdm import tqdm

# date and time:
from datetime import datetime
import time

from prettytable import PrettyTable
```

```
c:\users\byron\applications\pythonmaster\lib\site-packages\sklearn\ensemble
```

```
\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an int
ernal NumPy module and should not be imported. It will be removed in a futur
e NumPy release.
  from numpy.core.umath_tests import inner1d
```

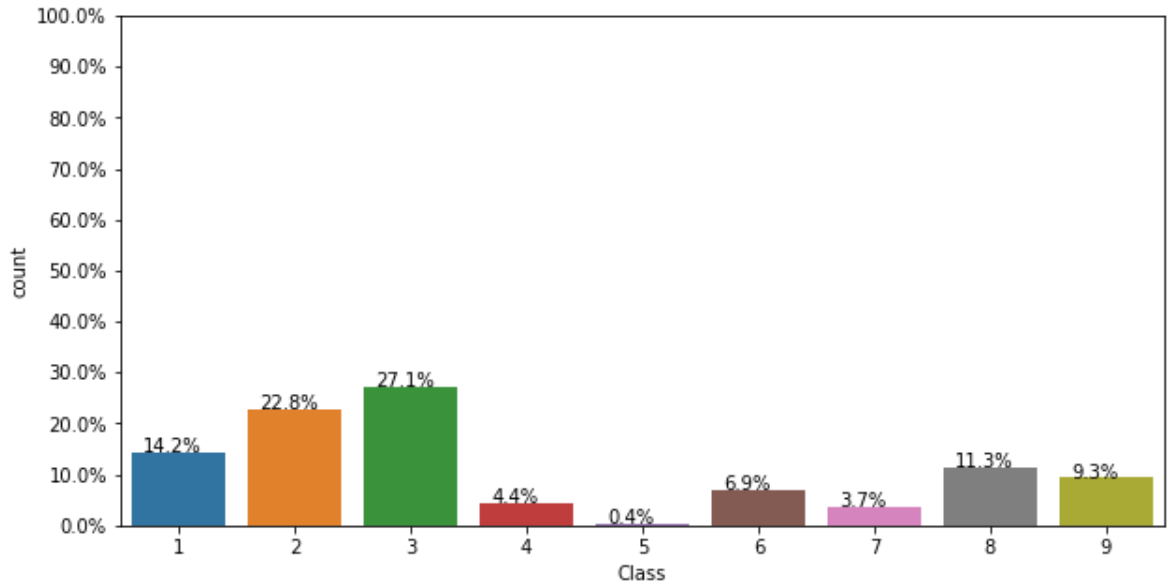
In [2]:

```
#separating byte files and asm files

source = 'train'
destination = 'byteFiles'

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

# 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 with
# 'byteFiles' folder

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

3.1. Distribution of malware classes in whole data set

In [3]:

```
plt.figure(figsize = (10,5))
Y=pd.read_csv("trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_l

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [4]:

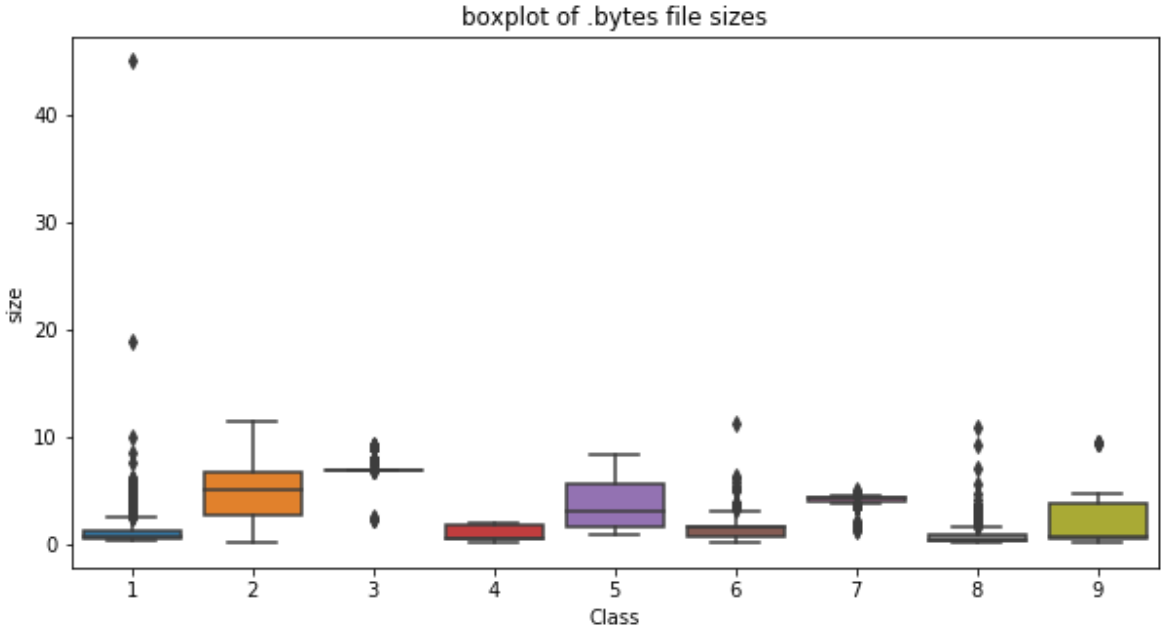
```
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
num_lines=[]
fnames=[]
for file in tqdm(files):
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nli
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('byteFiles/'+file)
    with open('byteFiles/'+file,'r') as content:
        num_of_lines = len(content.readlines())
    # 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))
        num_lines.append(num_of_lines)
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes, 'filelines':num_lines, 'Cla
print (data_size_byte.head())
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 10868/10868 [07:39<00:00, 23.65it/s]

                     ID      size  filelines  Class
0  01azqd4InC7m9JpocGv5  4.234863      90624      9
1  01IsoiSMh5gxyDYTl4CB  5.538818     118528      2
2  01jsnpXSAlgw6aPeDxrU  3.887939      83200      9
3  01kcPWA9K2BOxQeS5Rju  0.574219      12288      1
4  01SuzwMJEIXsK7A8dQbl  0.370850       7936      8
```

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

In [5]:

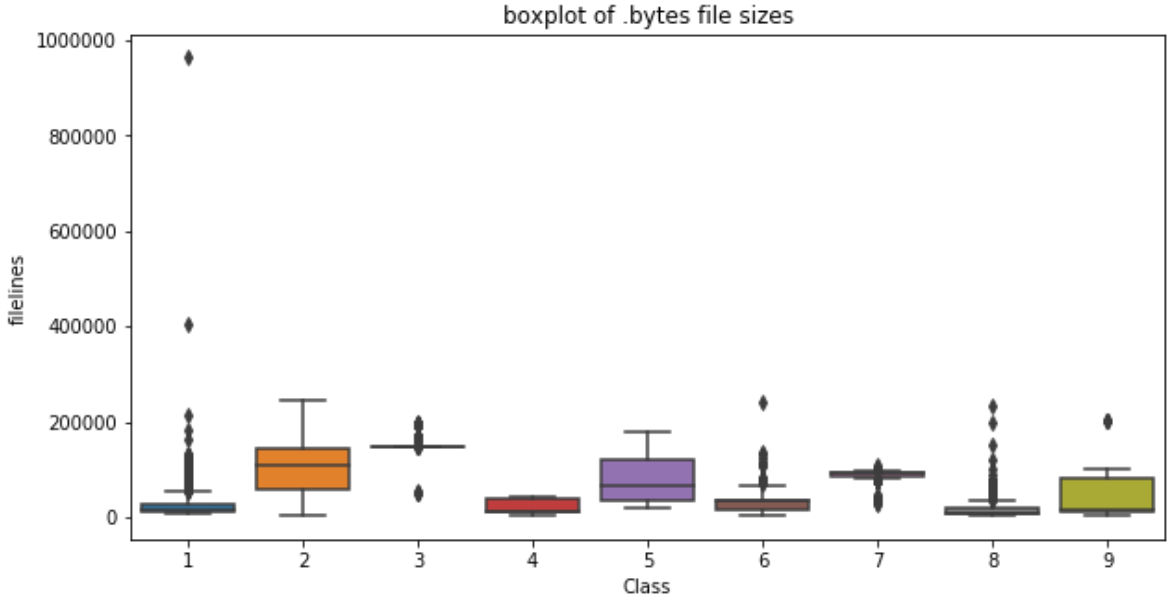
```
#boxplot of byte files
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



3.2.2.2 box plots of file row numbers (.byte files) feature

In [6]:

```
#boxplot of byte files
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="filelines", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



3.2.3 feature extraction from byte files

In [7]:

```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000
files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in tqdm(files):
    if(file.endswith("bytes")):
        filename=file.split('.')[0]
        text_file = open('byteFiles\\'+filename+".txt", 'w+')
        with open('byteFiles\\'+file,"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)
        text_file.close()
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 10868/10868 [00:00<00:00, 1815794.13it/s]
```

In [8]:

```
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
#this runs for about 3 hours so be patient on this section
if not os.path.exists('result.csv'):
    files = os.listdir('byteFiles')
    filenames2=[]
    feature_matrix = np.zeros((len(files),257),dtype=int)
    k=0
    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,10,11,12,13,14,1
    byte_feature_file.write("\n")
    for file in tqdm(files):
        filenames2.append(file.split('.')[0])
        byte_feature_file.write(file.split('.')[0]+",")
        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 in feature_matrix[k]:
            byte_feature_file.write(str(i)+",")
        byte_feature_file.write("\n")

        k += 1
    byte_feature_file.close()
```

In [9]:

```
byte_features=pd.read_csv("result.csv",index_col=False)
print (byte_features.head())
```

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

      8  ...      f7    f8    f9    fa    fb    fc    fd      fe      ff
??
0  2965  ...    2804  3687  3101  3211  3097  2758  3099    2759    5753    182
4
1  9291  ...     451  6536   439   281   302  7639   518   17001   54902    858
8
2  9107  ...    2325  2358  2242  2885  2863  2471  2786    2680   49144     46
8
3  1078  ...     478   873   485   462   516  1133   471     761    7998   1394
0
4   422  ...     847   947   350   209   239   653   221     242    2199    900
8

[5 rows x 258 columns]
```

In [10]:

```
result = pd.merge(byte_features, data_size_byte,on='ID', how='left')
result['bytezeroweighted'] = result['0']*2
result.head()
```

Out[10]:

	ID	0	1	2	3	4	5	6	7	8	...	fb
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3097
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	302
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2863
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	516
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	239

5 rows × 262 columns

In [11]:

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

In [12]:

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

Out[12]:

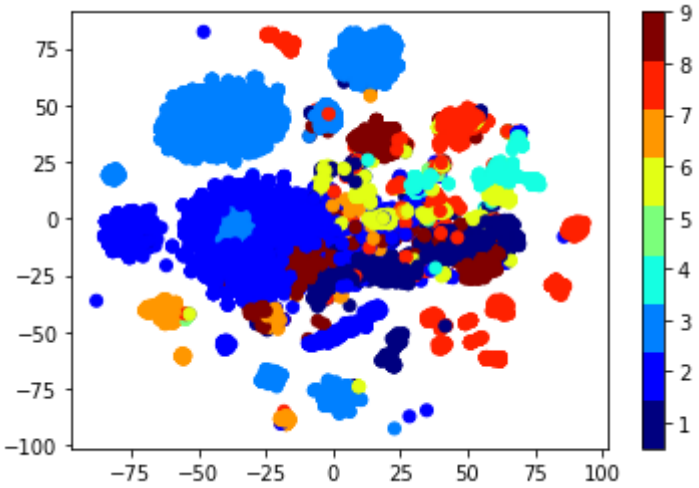
	ID	0	1	2	3	4	5	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.0020
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0047
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0050
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708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

5 rows × 262 columns

3.2.4 Multivariate Analysis

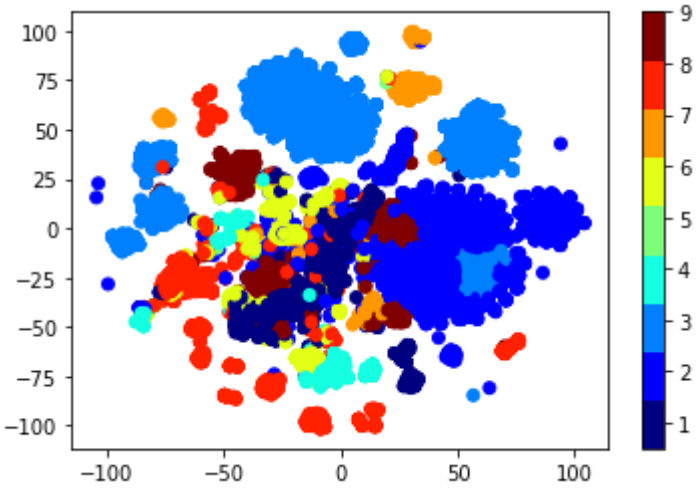
In [13]:

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



In [14]:

```
#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()
```



dChad Features

In [15]:

```
dchad = pd.DataFrame()
for file in os.listdir('dchad'):
    if file[-3:] == 'csv':
        filelocation = 'dchad/'+file
        dchad = pd.concat(objs=[dchad,pd.read_csv(filelocation)],axis=0)
dchad.rename(columns={'filename':'ID'},inplace=True)
dchad.head()
```

Out[15]:

	ID	vertex_count	edge_count	delta_max	density
0	4jKA1GUDv6TMNpPuIxER	3243	5440	85	0.005500
1	4ZBJzEqnW52fFUw0PG3v	40	39	36	13.000000
2	6m8NxLfg2MR0nwXFuEq5	95	94	61	0.029012
3	28U1hRkQ6YI57493ZdXD	120	119	10	0.200000
4	45Wy3TxE98HfiXreOCSu	339	337	50	0.047999

In [16]:

```
dchad1 = normalize(dchad)
```

In [17]:

```
dchad1.head()
```

Out[17]:

	ID	vertex_count	edge_count	delta_max	density
0	4jKA1GUDv6TMNpPuIxER	0.113854	0.139936	0.010612	0.000015
1	4ZBJzEqnW52fFUw0PG3v	0.001370	0.001003	0.004494	0.034685
2	6m8NxLfg2MR0nwXFuEq5	0.003301	0.002418	0.007615	0.000077
3	28U1hRkQ6YI57493ZdXD	0.004179	0.003061	0.001248	0.000534
4	45Wy3TxE98HfiXreOCSu	0.011870	0.008669	0.006242	0.000128

In [18]:

```
result = pd.merge(result,dchad1, on = 'ID', how='inner')
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	0.0020
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.0047
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.0050
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.0003
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.0001

5 rows × 266 columns

Train Test split

In [19]:

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaib
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1
# split the train data into train and cross validation by maintaining same distributio
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_
```

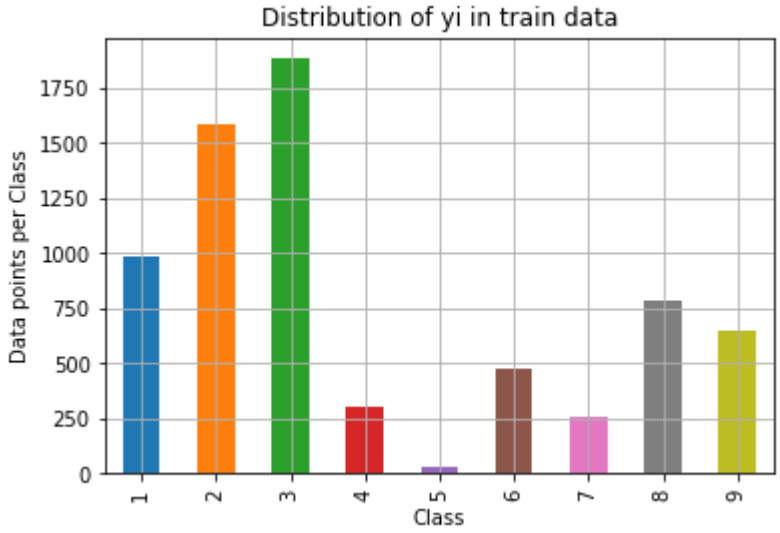
In [20]:

```
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])
```

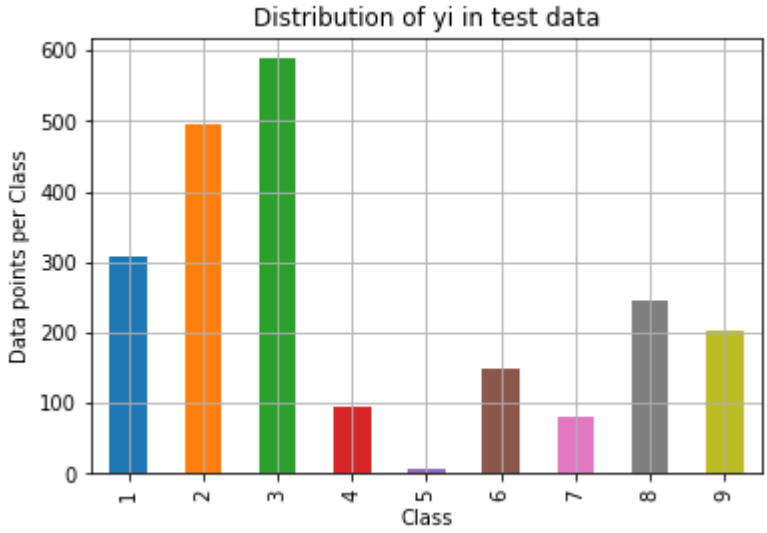
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [21]:

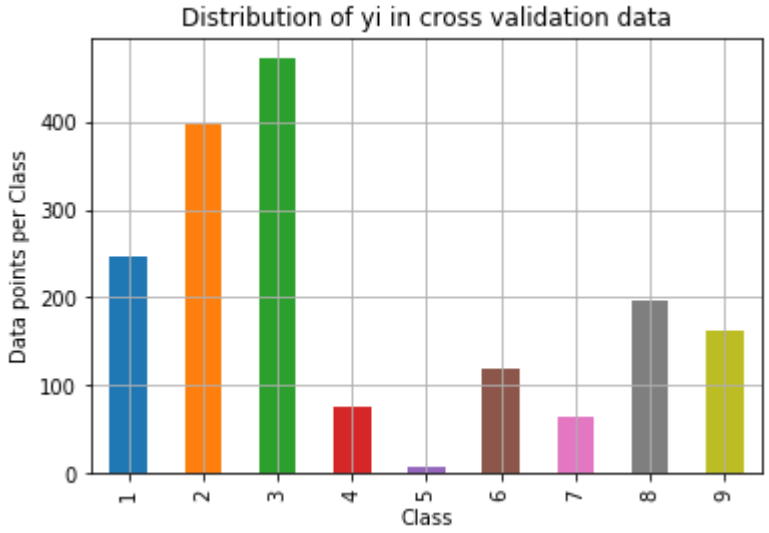
```
# it returns a dict, keys as class labels and values as the number of data points in tl
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

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

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


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

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

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

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

```




```
Number of data points in class 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 %)
--------------------------------------------------------------------------------
----
```




```
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 %)
--------------------------------------------------------------------------------
----
```




```
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 [22]:

```
def plot_confusion_matrix(test_y, predict_y, model):

    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predi

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

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

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

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
    # 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=la
    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=la
    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=la
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
#     print("Sum of rows in precision matrix",A.sum(axis=1))

```

```
    miss_classified_rate = round((1 - accuracy_score(test_y, predict_y))*100,2)
    print("Percentage of misclassified points using {} Model: ".format(model),str(miss
```

4. Machine Learning Models

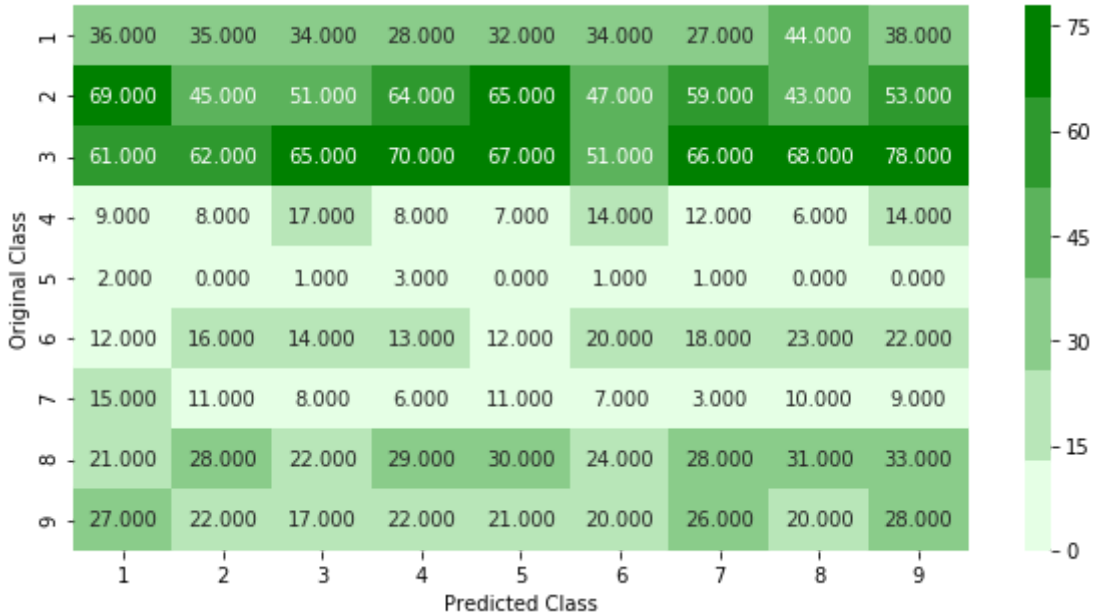
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

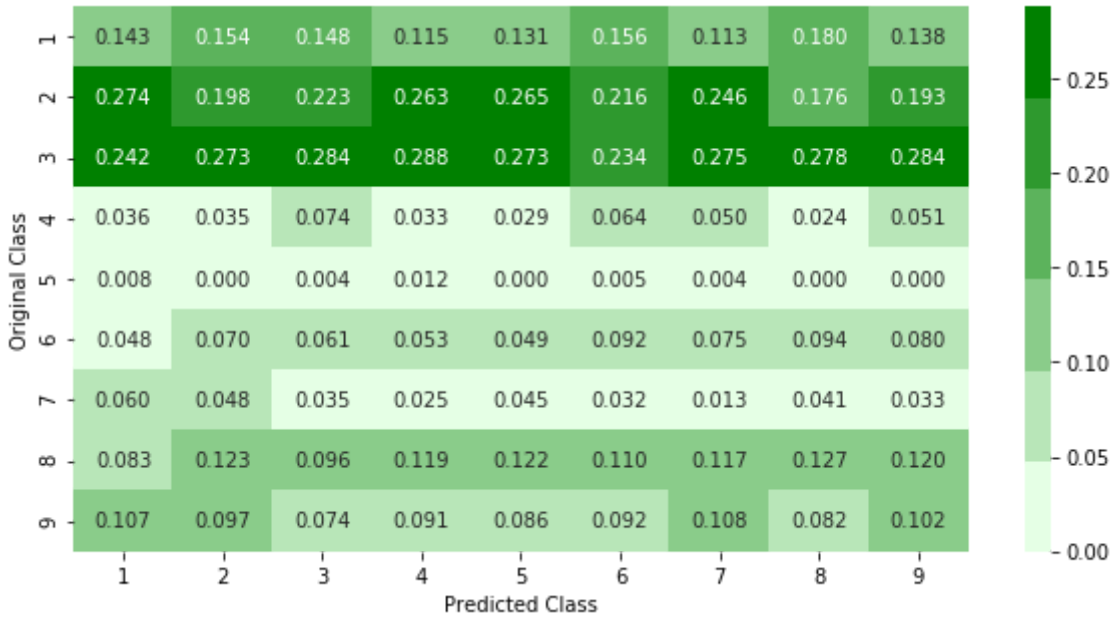
In [23]:

```
# 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

train_data_len = X_train.shape[0]
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 train data
random_pred_train = np.random.rand(train_data_len,9)
for key, value in dict(zip(np.sum(random_pred_train,axis=1),random_pred_train)).items(
    dict(zip(np.sum(random_pred_train,axis=1),random_pred_train))[key] = value/key
print("Log loss on Training Data using Random Model: ",log_loss(y_train,random_pred_tr

# we create a output array that has exactly same size as the CV data
random_pred_cv = np.random.rand(cv_data_len,9)
for key, value in dict(zip(np.sum(random_pred_cv,axis=1),random_pred_cv)).items():
    dict(zip(np.sum(random_pred_cv,axis=1),random_pred_cv))[key] = value/key
print("Log loss on Cross Validation Data using Random Model: ",log_loss(y_cv,random_pr


# we create a output array that has exactly same as the test data
random_pred_test = np.random.rand(test_data_len,9)
for key, value in dict(zip(np.sum(random_pred_test,axis=1),random_pred_test)).items():
    dict(zip(np.sum(random_pred_test,axis=1),random_pred_test))[key] = value/key
print("Log loss on Test Data using Random Model: ",log_loss(y_test,random_pred_test, e

predicted_y = np.argmax(random_pred_test, axis=1) + 1 # here we return the index(class
plot_confusion_matrix(y_test, predicted_y,'Random')
```

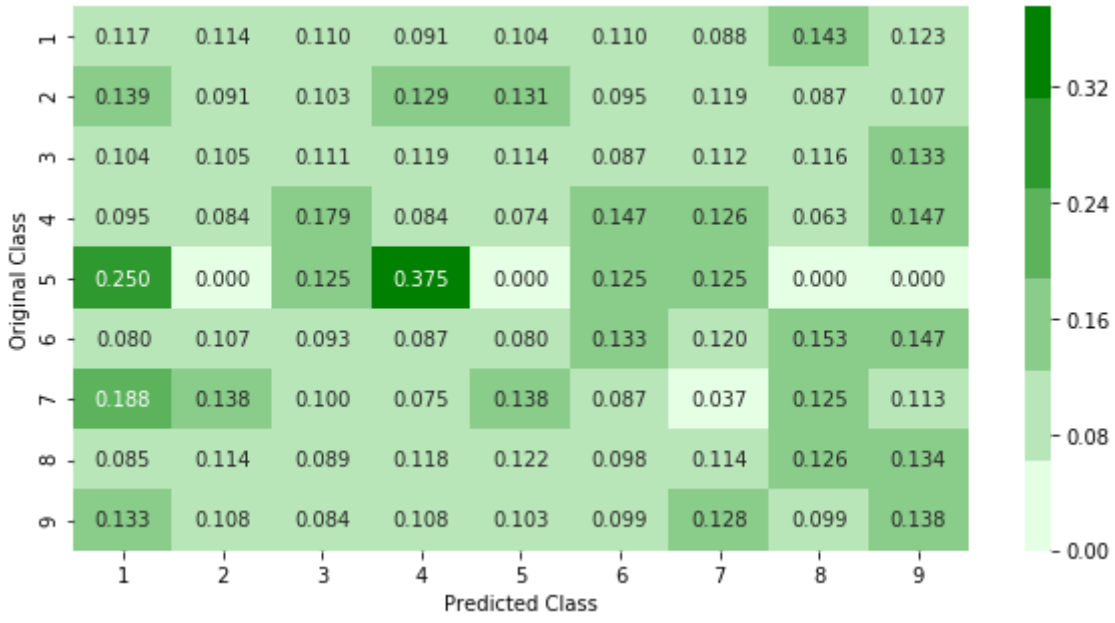
```
Log loss on Training Data using Random Model:  2.485348708199024
Log loss on Cross Validation Data using Random Model:  2.4860901717051105
Log loss on Test Data using Random Model:  2.4631112739428467
---------------------------------------------------- Confusion matrix --------
-------------------------------------------
```



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




```
------------------------------------------------- Recall matrix -----------
--------------------------------------
```



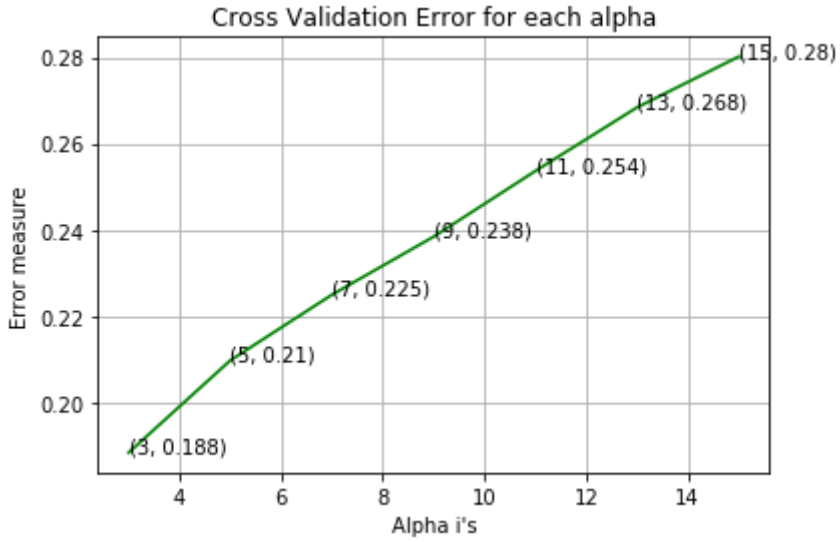

```
Percentage of misclassified points using Random Model:  89.14%
```

4.1.2. K Nearest Neighbour Classification

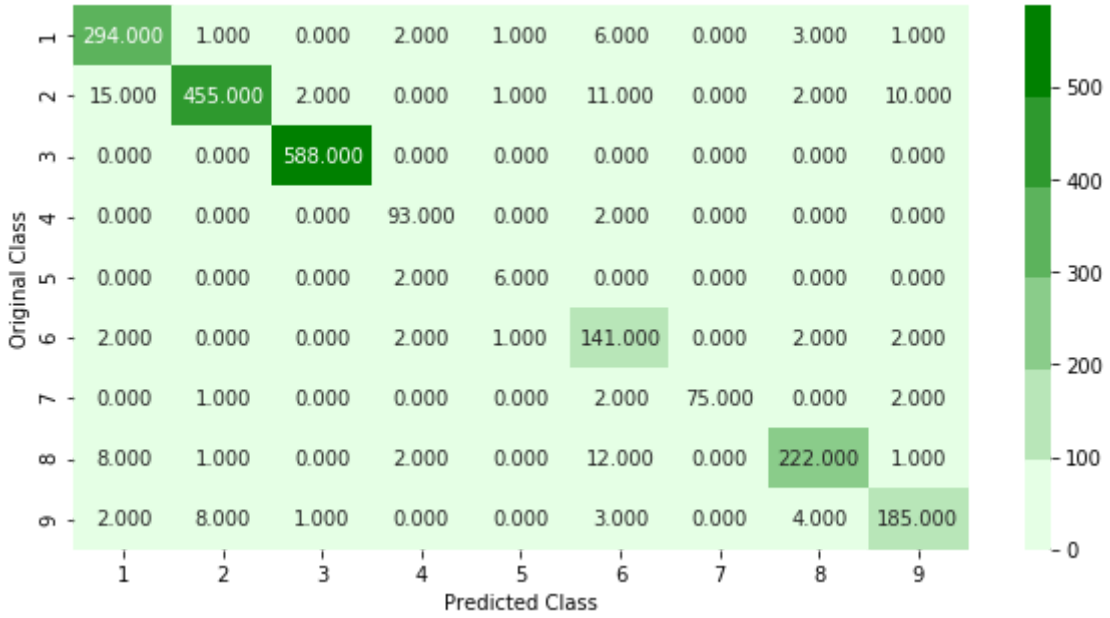
In [24]:

```
alpha = [x for x in [3,5,7,9,11,13,15]]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-

for i in zip(alpha,cv_log_error_array):
    print ('log_loss for k = ',i[0],'is',i[1])

best_alpha = np.argmin(cv_log_error_array)

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

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

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

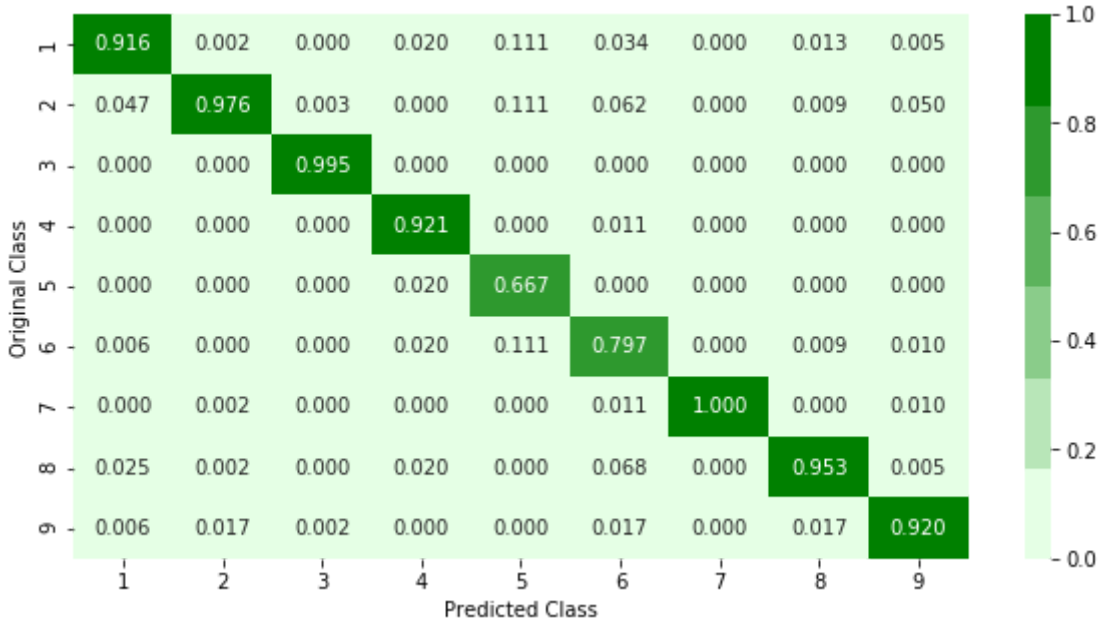
```
log_loss for k =  3 is 0.18837038397077538
log_loss for k =  5 is 0.20974175439876952
log_loss for k =  7 is 0.2249754515332201
log_loss for k =  9 is 0.23842533912526098
log_loss for k =  11 is 0.2536408767785393
log_loss for k =  13 is 0.2683846685296005
log_loss for k =  15 is 0.2801513482426867
```



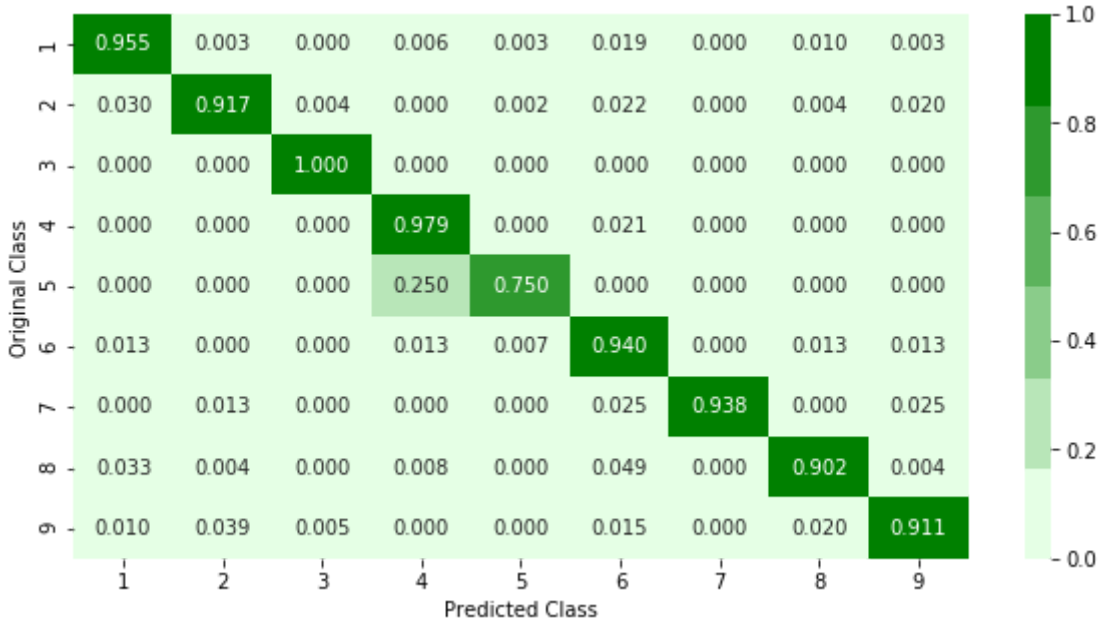

```
For values of best alpha =  3 The train log loss is: 0.11842650208618187
For values of best alpha =  3 The cross validation log loss is: 0.1883703839
7077538
For values of best alpha =  3 The test log loss is: 0.20877787985735408
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```




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




```
------------------------------------------------- Recall matrix -----------
---------------------------------------
```



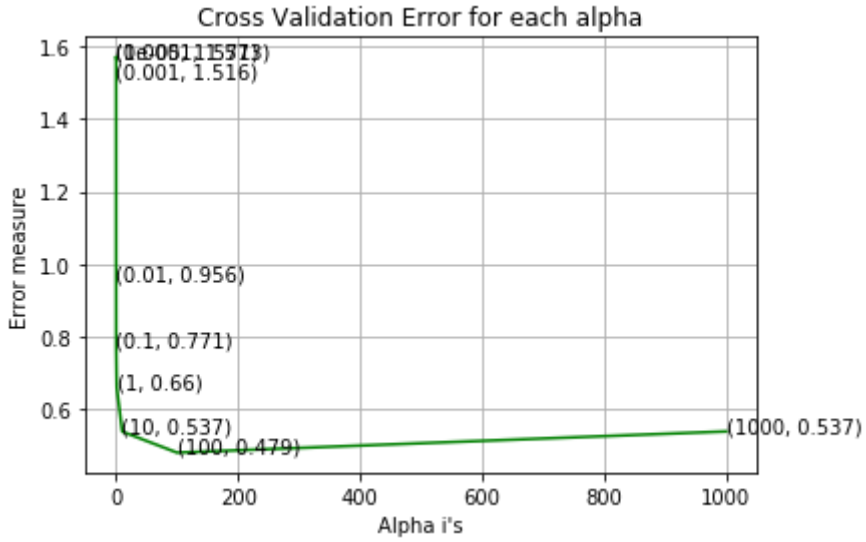

```
Percentage of misclassified points using KNN(n = 3) Model:  5.29%
```

4.1.3. Logistic Regression

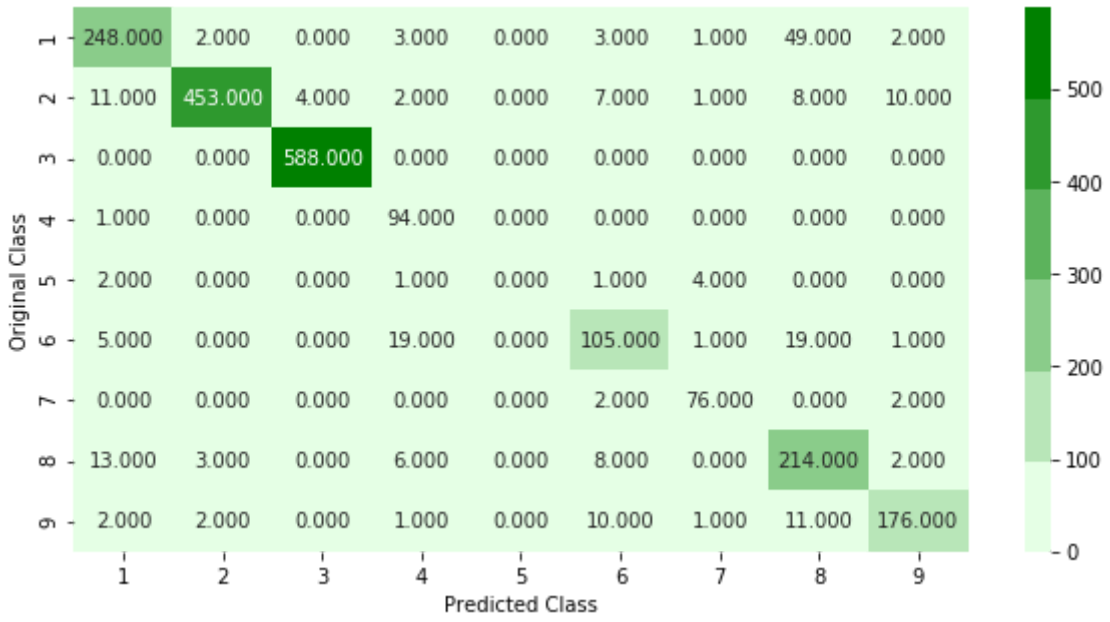
In [25]:

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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


predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps
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),'Logistic Regression (C = {})'.f
```

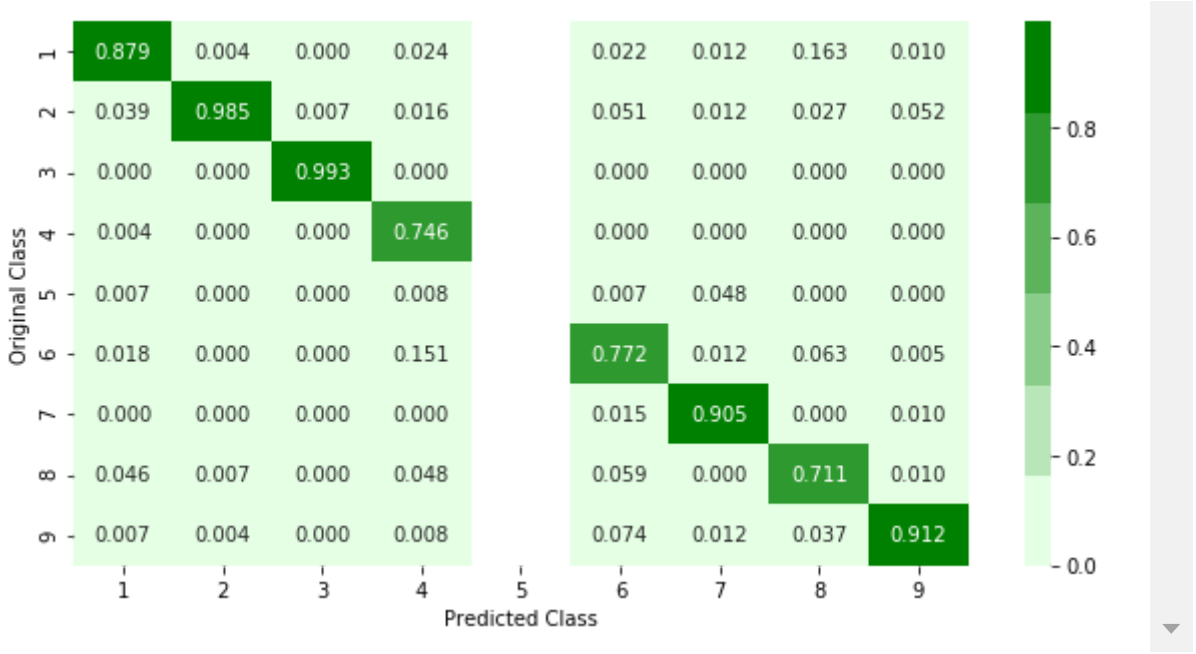
```
log_loss for c =  1e-05 is 1.5707695803563106
log_loss for c =  0.0001 is 1.5730050621309748
log_loss for c =  0.001 is 1.5162064471788645
log_loss for c =  0.01 is 0.9555554238450514
log_loss for c =  0.1 is 0.7711637984102886
log_loss for c =  1 is 0.6595735959566625
log_loss for c =  10 is 0.5368871361250238
log_loss for c =  100 is 0.47860793818742364
log_loss for c =  1000 is 0.5372097559827
```



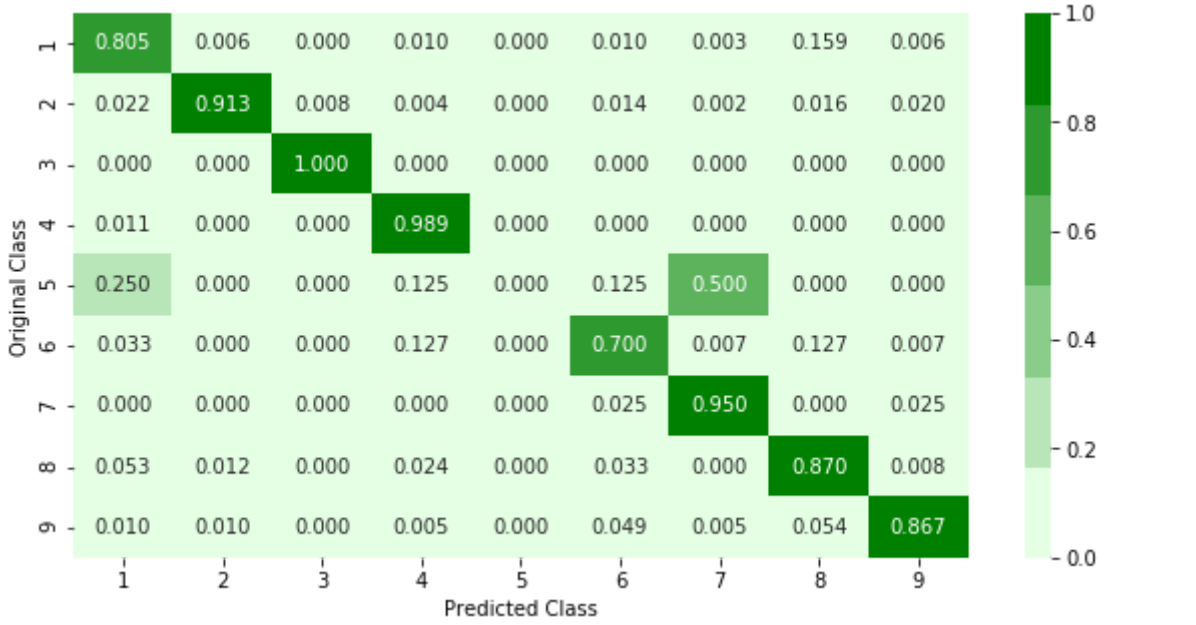

```
log loss for train data 0.4347852779128462
log loss for cv data 0.47860793818742364
log loss for test data 0.45486238918841615
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```




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



---------- Recall matrix ----------



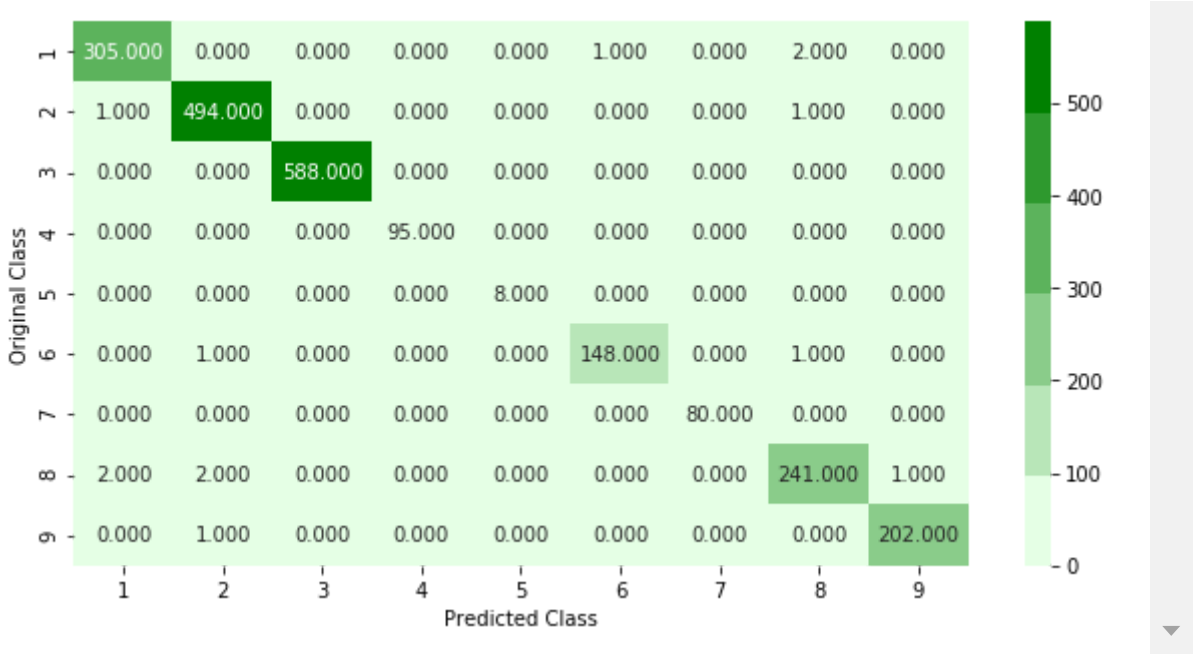
Percentage of misclassified points using Logistic Regression (C = 100) Model: 10.12%

4.1.4. Random Forest Classifier

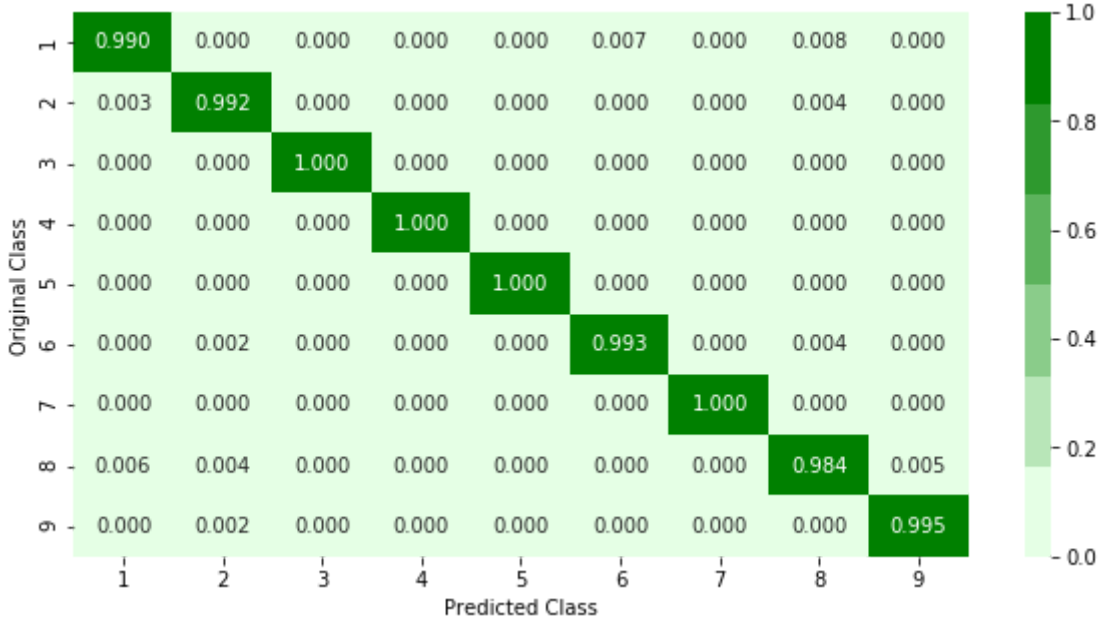
In [26]:

```
if not os.path.exists('RF_one_param_tuning.joblib'):
    alpha=[10,50,100,500,1000,2000,3000]
    cv_log_error_array=[]
    train_log_error_array=[]
    for i in alpha:
        r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
        r_cfl.fit(X_train,y_train)
        sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
        sig_clf.fit(X_train, y_train)
        predict_y = sig_clf.predict_proba(X_cv)
        cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps

    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)

    joblib.dump(sig_clf,'RF_one_param_tuning.joblib')
else:
    sig_clf = joblib.load('RF_one_param_tuning.joblib')

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

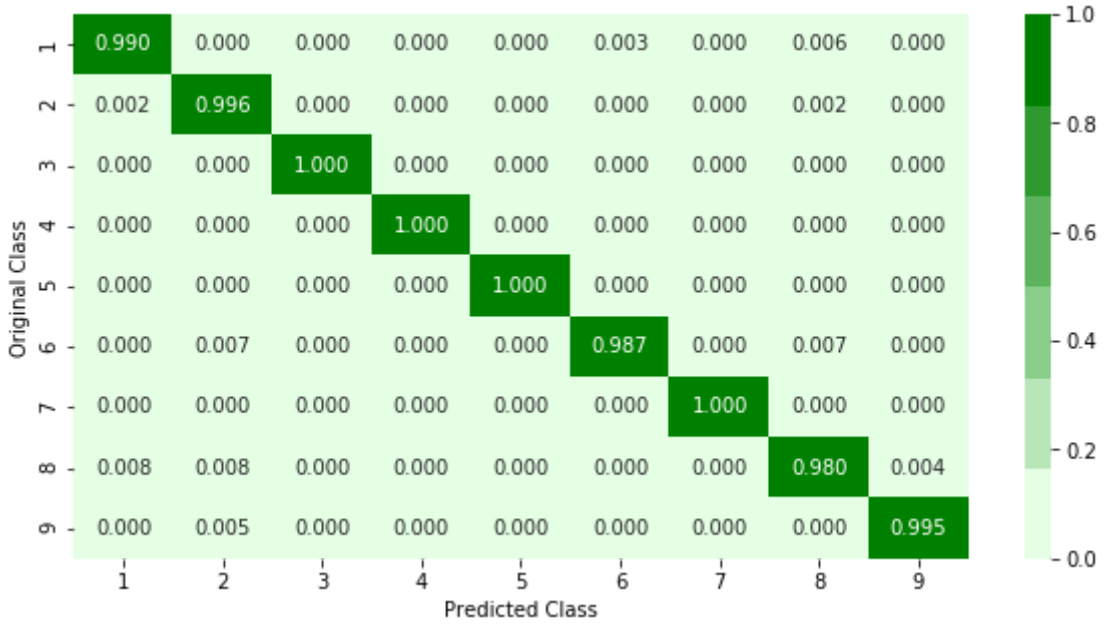
```
For values of best alpha =  100 The train log loss is: 0.044166783323594104
For values of best alpha =  100 The cross validation log loss is: 0.03739752
427035803
For values of best alpha =  100 The test log loss is: 0.04016687451345051
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```



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



---------- Recall matrix ----------



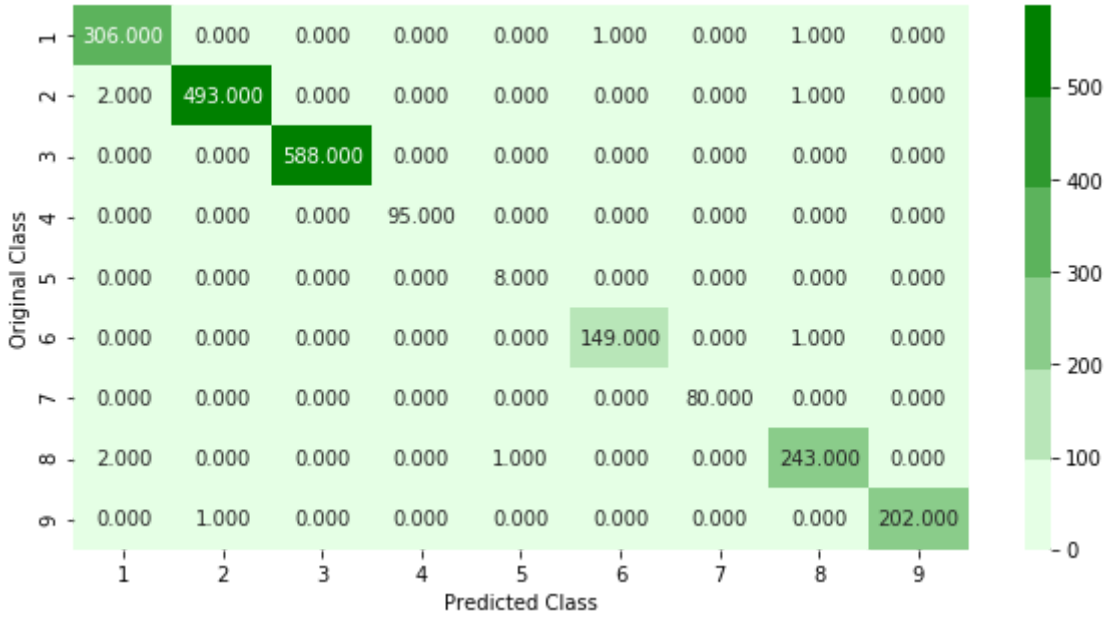

```
Percentage of misclassified points using Random Forest (n_estimators = 100)
Model:  0.6%
```

4.1.5. XgBoost Classification

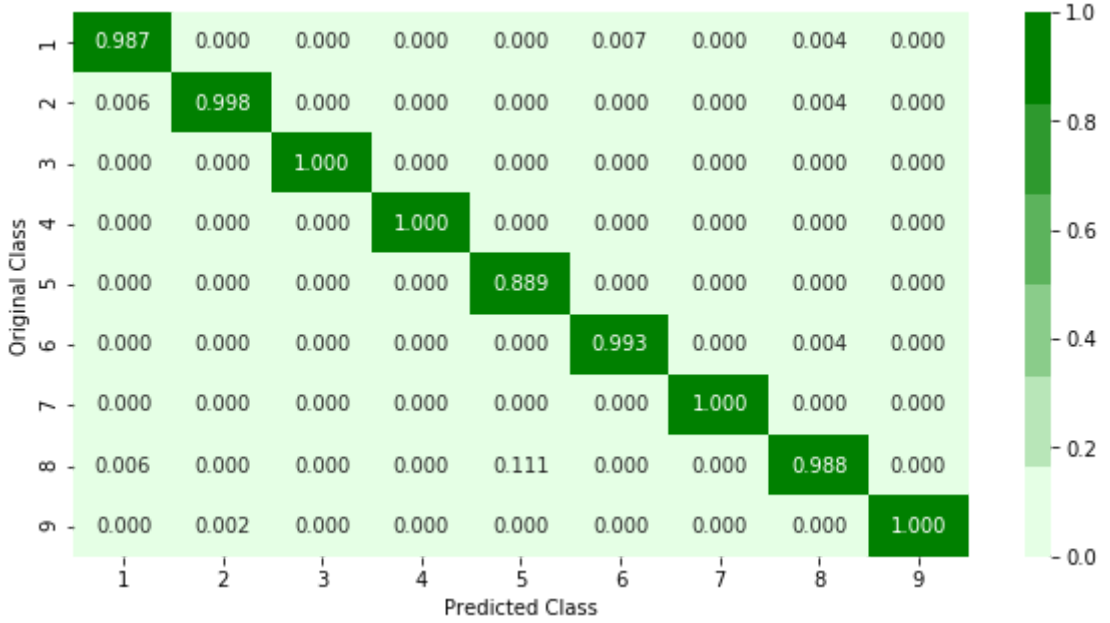
In [27]:

```
if not os.path.exists('XGB_one_param_tuning.joblib'):
    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

    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)

    joblib.dump(sig_clf,'XGB_one_param_tuning.joblib')
else:
    sig_clf = joblib.load('XGB_one_param_tuning.joblib')

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 loss
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
plot_confusion_matrix(y_test, sig_clf.predict(X_test),'XGB Model (n_estimators = {})'.
```

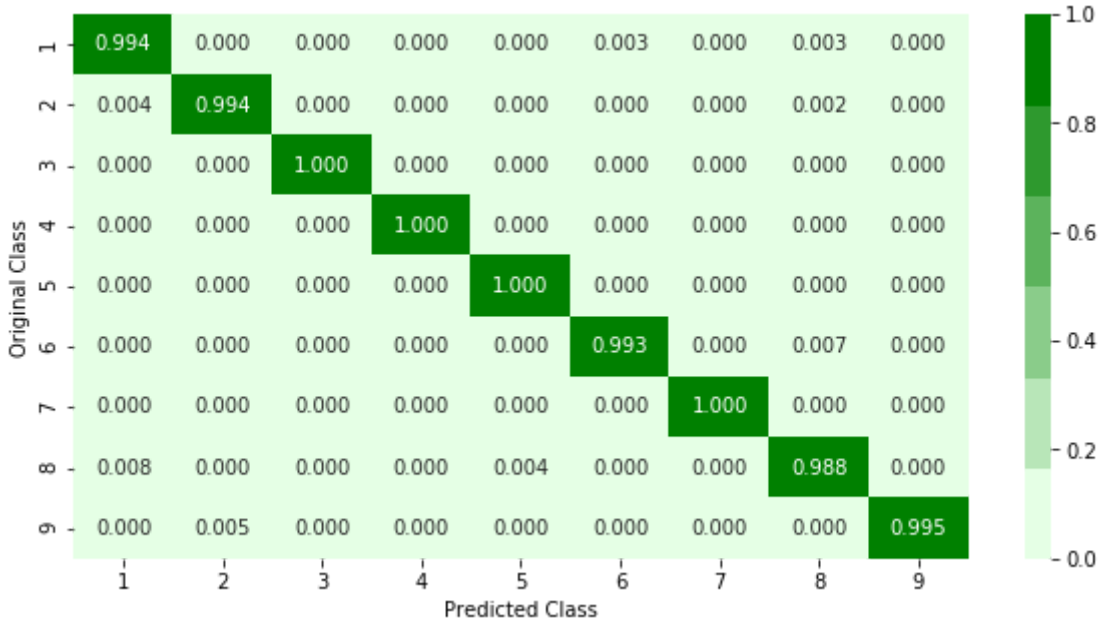
```
For values of best alpha =  100 The train log loss is: 0.03208534964008829
For values of best alpha =  100 The cross validation log loss is: 0.02210324
001473583
For values of best alpha =  100 The test log loss is: 0.03295365564516063
-------------------------------------------------- Confusion matrix --------
----------------------------------------
```



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



---------- Recall matrix -----------




```
Percentage of misclassified points using XGB Model (n_estimators = 100) Mode
l:  0.46%
```

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [28]:

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost
if not os.path.exists('XGB_multiple_param_tuning.joblib'):
    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,n_iter=10,verbose=1
    random_cfl1.fit(X_train,y_train)
    joblib.dump(random_cfl1, 'XGB_multiple_param_tuning.joblib')
else:
    random_cfl1 = joblib.load('XGB_multiple_param_tuning.joblib')
```

In [29]:

```
1 random_cfl1.best_params_
```

Out[29]:

```
{'subsample': 1,
 'n_estimators': 500,
 'max_depth': 5,
 'learning_rate': 0.2,
 'colsample_bytree': 1}
```

In [30]:

```
1 random_cfl1.best_estimator_
```

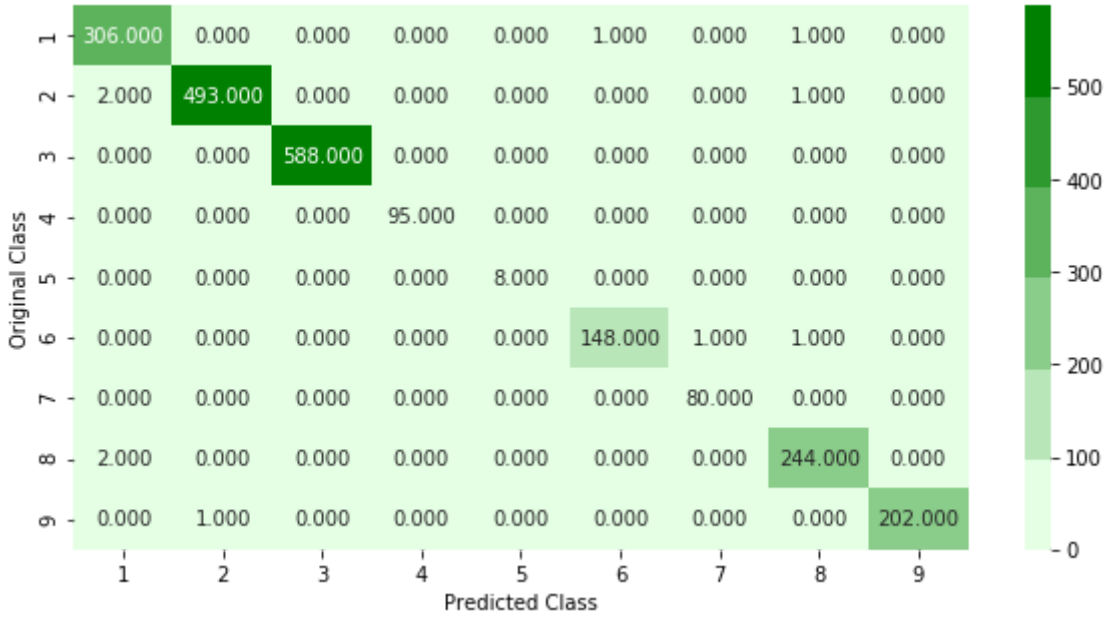
Out[30]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.2, max_delta_step=0,
       max_depth=5, min_child_weight=1, missing=nan, n_estimators=500,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

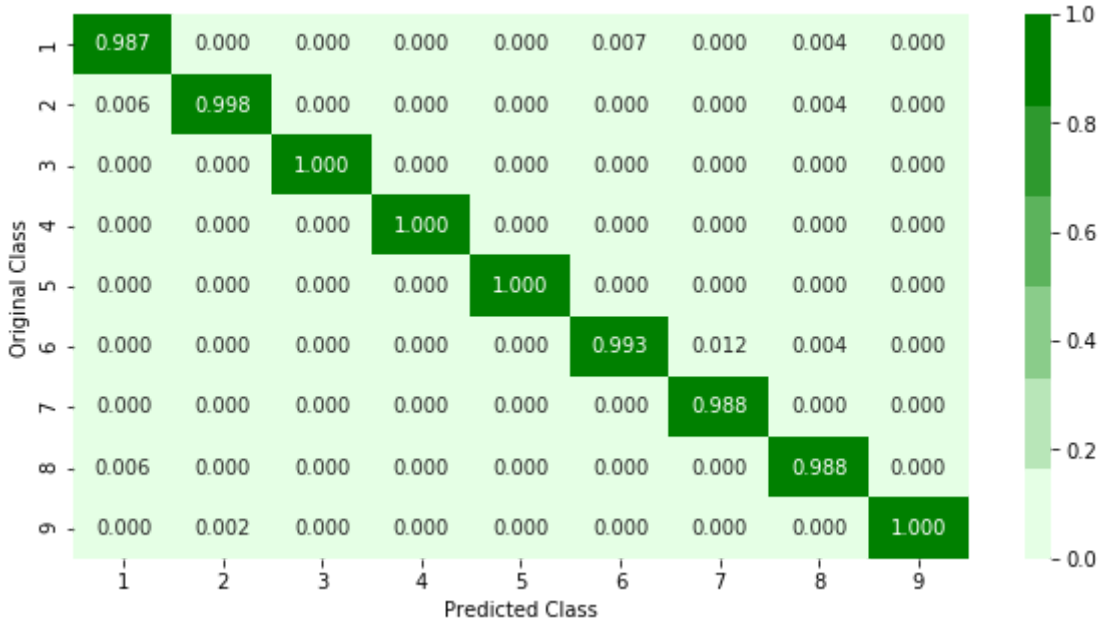
In [31]:

```
if not os.path.exists('XGB_multiple_param_tuning_Calibrated.joblib'):
    c_cfl = CalibratedClassifierCV(random_cfl1.best_estimator_ ,method='sigmoid')
    c_cfl.fit(X_train,y_train)
    joblib.dump(c_cfl, 'XGB_multiple_param_tuning_Calibrated.joblib')
else:
    c_cfl = joblib.load('XGB_multiple_param_tuning_Calibrated.joblib')

predict_y = c_cfl.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print("The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, c_cfl.predict(X_test),'XGB Model (with more parameters t
```

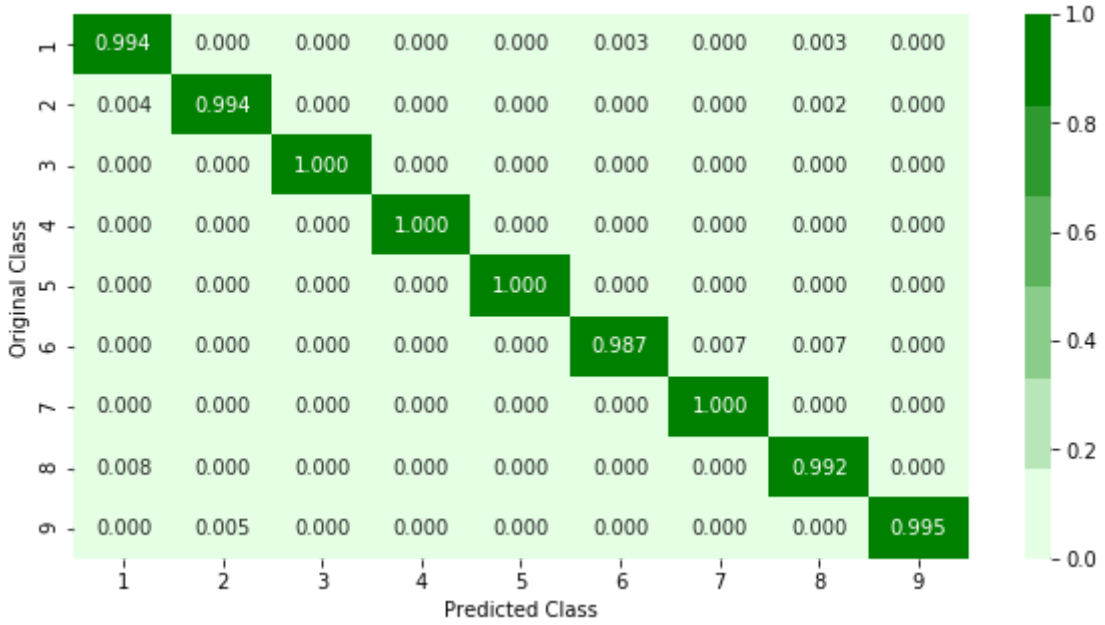
```
The train log loss is: 0.032408503225918525
The cross validation log loss is: 0.02121710172377088
The test log loss is: 0.03332660642651036
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```




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




```
---------------------------------------------------- Recall matrix -----------
---------------------------------------
```




```
Percentage of misclassified points using XGB Model (with more parameters tun
ed) Model:  0.46%
```

4.2 Modeling with .asm files

```
There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
```

6. APIs
With the help of parallel processing we extracted all the features.In parallel we can use all the cores that are present in our computer.

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

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

4.2.1 Feature extraction from asm files

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

In [32]:

```
# Create the directories for parallel processing:
folder_1 = 'amsfiles_batch1'
folder_2 = 'amsfiles_batch2'
folder_3 = 'amsfiles_batch3'
folder_4 = 'amsfiles_batch4'
folder_5 = 'output'
for i in [folder_1,folder_2,folder_3,folder_4,folder_5]:
    if not os.path.isdir(i):
        os.makedirs(i)
# ================================================================================
source_dir ='asmFiles' # change to the asmfiles directory (puth this back to testing i
files = os.listdir(source_dir)
total_files_per_batch = len(files)//4 # since my computer has 4 cores I create 4 equal
total_files_per_batch
for destination_dir in [folder_1,folder_2,folder_3,folder_4]:
    if len(os.listdir(destination_dir)) == 0:
        print('{} transfer started....'.format(destination_dir))
        for i in range(total_files_per_batch):
            file = files.pop(files.index(r.choice(files))) #pick a random file with equ
            src = os.path.join(source_dir,file)
            dst = os.path.join(destination_dir,file)
            shutil.copy2(src=src,dst=dst)
        print('{} transfer complete'.format(destination_dir))
    else:
        print('Files already moved to the destination directory')
```

```
Files already moved to the destination directory
Files already moved to the destination directory
Files already moved to the destination directory
Files already moved to the destination directory
```

In [33]:

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

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

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

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

    #symbols
    symbols=['[','-','+','*','?','@',']']


    asmfile1 = open(os.path.join(folder_5,'asmfile1.txt'),"w+") # open the file stream

    files = os.listdir(folder_1)

    for file 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)
        symbolscount = np.zeros(len(symbols),dtype=int)

        features = list()

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

        asmfile1.write(filename + ",")

        # 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(os.path.join(folder_1,file),encoding='cp1252',errors ='replac
            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

                for i in range(len(symbols)):
                    for item in line:
                        if symbols[i] in item:
                            symbolscount[i]+=1


        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            asmfile1.write(str(prefix)+",")
        for opcode in opcodescount:
            asmfile1.write(str(opcode)+",")
        for register in registerscount:
            asmfile1.write(str(register)+",")
        for key in keywordcount:
            asmfile1.write(str(key)+",")
        for symbol in symbolscount:
            asmfile1.write(str(symbol)+",")
        asmfile1.write("\n")
    asmfile1.close()

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

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

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

    #symbols
    symbols=['[','-','+','*','?','@',']']

    asmfile2 = open(os.path.join(folder_5,'asmfile2.txt'),"w+") # open the file stream

    files = os.listdir(folder_2)

    for file 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)
        symbolscount = np.zeros(len(symbols),dtype=int)

        features = list()

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

        asmfile2.write(filename + ",")

        # 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(os.path.join(folder_2,file),encoding='cp1252',errors ='replac
            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

                for i in range(len(symbols)):
                    for item in line:
                        if symbols[i] in item:
```

```
                            symbolscount[i]+=1

        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            asmfile2.write(str(prefix)+",")
        for opcode in opcodescount:
            asmfile2.write(str(opcode)+",")
        for register in registerscount:
            asmfile2.write(str(register)+",")
        for key in keywordcount:
            asmfile2.write(str(key)+",")
        for symbol in symbolscount:
            asmfile2.write(str(symbol)+",")
        asmfile2.write("\n")
    asmfile2.close()

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

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

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

    #symbols
    symbols=['[','-','+','*','?','@',']']

    asmfile3 = open(os.path.join(folder_5,'asmfile3.txt'),"w+") # open the file stream

    files = os.listdir(folder_3)

    for file 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)
        symbolscount = np.zeros(len(symbols),dtype=int)

        features = list()

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

        asmfile3.write(filename + ",")

        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
```

```
241
242         with codecs.open(os.path.join(folder_3,file),encoding='cp1252',errors ='replac
243             for lines in fli:
244                 # https://www.tutorialspoint.com/python3/string_rstrip.htm
245
246                 line = lines.rstrip().split()
247                 l = line[0]
248                 #counting the prefixs in each and every line
249                 for i in range(len(prefixes)):
250                     if prefixes[i] in line[0]:
251                         prefixescount[i]+=1
252
253                 line = line[1:]
254                 #counting the opcodes in each and every line
255                 for i in range(len(opcodes)):
256                     if any(opcodes[i] == li for li in line):
257                         features.append(opcodes[i])
258                         opcodescount[i]+=1
259
260                 #counting registers in the line
261                 for i in range(len(registers)):
262                     for li in line:
263                         # we will use registers only in 'text' and 'CODE' segments
264                         if registers[i] in li and ('text' in l or 'CODE' in l):
265                             registerscount[i]+=1
266
267                 #counting keywords in the line
268                 for i in range(len(keywords)):
269                     for li in line:
270                         if keywords[i] in li:
271                             keywordcount[i]+=1
272
273                 for i in range(len(symbols)):
274                     for item in line:
275                         if symbols[i] in item:
276                             symbolscount[i]+=1
277
278         #pushing the values into the file after reading whole file
279         for prefix in prefixescount:
280             asmfile3.write(str(prefix)+",")
281         for opcode in opcodescount:
282             asmfile3.write(str(opcode)+",")
283         for register in registerscount:
284             asmfile3.write(str(register)+",")
285         for key in keywordcount:
286             asmfile3.write(str(key)+",")
287         for symbol in symbolscount:
288             asmfile3.write(str(symbol)+",")
289         asmfile3.write("\n")
290     asmfile3.close()
291
292 # same as above
293 def fourthprocess():
294     #The prefixes tells about the segments that are present in the asm files
295     #There are 450 segments(approx) present in all asm files.
296     #this prefixes are best segments that gives us best values.
297     #https://en.wikipedia.org/wiki/Data_segment
298
299     prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:',
300                 '.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
301     #this are opcodes that are used to get best results
```

```
    #https://en.wikipedia.org/wiki/X86_instruction_listings

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

    #symbols
    symbols=['[','-','+','*','?','@',']']

    asmfile4 = open(os.path.join(folder_5,'asmfile4.txt'),"w+") # open the file stream

    files = os.listdir(folder_4)

    for file 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)
        symbolscount = np.zeros(len(symbols),dtype=int)

        features = list()

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

        asmfile4.write(filename + ",")

        # 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(os.path.join(folder_4,file),encoding='cp1252',errors ='replac
            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

                for i in range(len(symbols)):
                    for item in line:
                        if symbols[i] in item:
                            symbolscount[i]+=1

        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            asmfile4.write(str(prefix)+",")
        for opcode in opcodescount:
            asmfile4.write(str(opcode)+",")
        for register in registerscount:
            asmfile4.write(str(register)+",")
        for key in keywordcount:
            asmfile4.write(str(key)+",")
        for symbol in symbolscount:
            asmfile4.write(str(symbol)+",")
        asmfile4.write("\n")
    asmfile4.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=mp.Process(target=firstprocess)
#     p2=mp.Process(target=secondprocess)
#     p3=mp.Process(target=thirdprocess)
#     p4=mp.Process(target=fourthprocess)

#     #p1.start() is used to start the thread execution
#     p1.start()
#     p2.start()
#     p3.start()
#     p4.start()
#     #After completion all the threads are joined
#     p1.join()
#     p2.join()
#     p3.join()
#     p4.join()

def main():
    try:
        print('feature extraction starting...')
        firstprocess()
        print('batch 1 processing done')
        secondprocess()
        print('batch 2 processing done')
        thirdprocess()
        print('batch 3 processing done')
        fourthprocess()
        print('process complete')
    except:
        print('Error')
```

In [34]:

```
if len(os.listdir(folder_5)) == 0:
    start = datetime.now()
    main()
    print(datetime.now() - start)
else:
    print('Output files already created')
```

```
Output files already created
```

In [35]:

```
if not os.path.exists('asmoutputfile.csv'):
    files = os.listdir('output')
    asmoutputfile = pd.DataFrame()
    for file in files:
        asmoutputfile = pd.concat(objs=[asmoutputfile,pd.read_csv(os.path.join('output
    asmoutputfile.columns = ['ID','HEADER:','.text:','.Pav:','.idata:','.data:','.bss:
                             '.CODE', 'jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retu
                             'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','j
                             'esi','eax','ebx','ecx','edi','ebp','esp','eip','extra','
    asmoutputfile.drop(columns='extra',inplace=True)
    asmoutputfile.to_csv('asmoutputfile.csv',header=True, index=False)
```

In [36]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the ext
# 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.drop(columns=']',inplace=True)
result_asm.head()
```

Out[36]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	
1	02IOCvYEy8mjiuAQHax3	17	838	0	41	77840	0	0	0	
2	04EjIdbPV5e1XroFOpiN	19	60476	0	349	3760	0	0	0	
3	05EeG39MTRrI6VY21DPd	17	11119	0	323	1047	0	3385	0	
4	05LHG8fR3iPn6agIo9z7	17	0	0	178	0	0	511	0	

5 rows × 59 columns

4.2.1.1 Files sizes of each .asm file

In [37]:

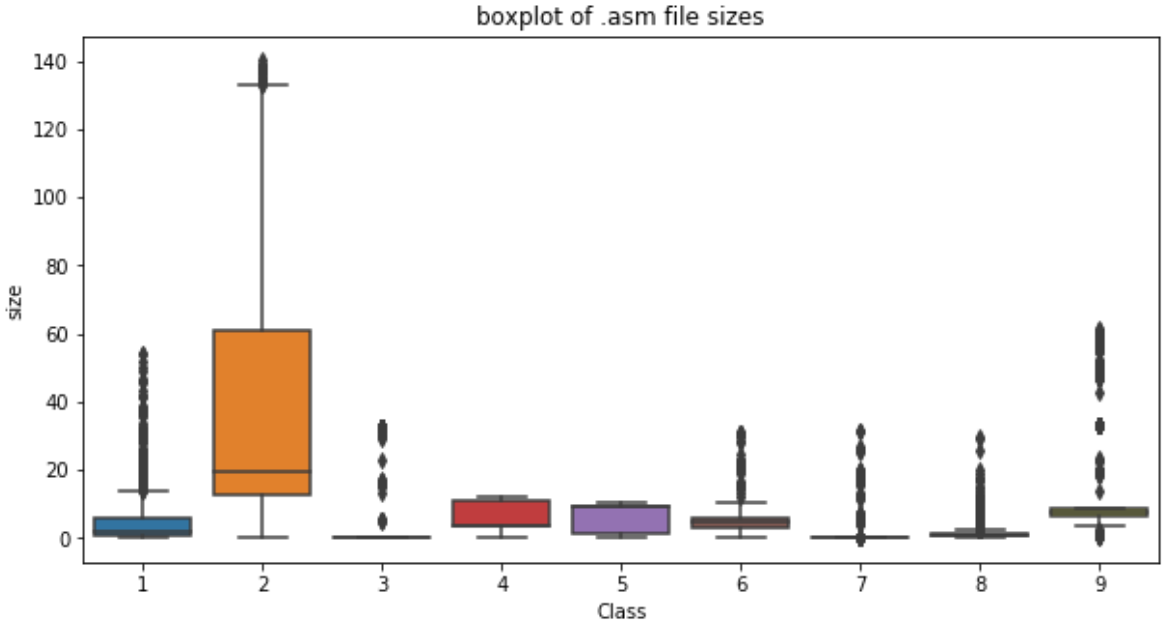
```
#file sizes of byte files
files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
num_lines = []
fnames=[]
for file in tqdm(files):
    statinfo=os.stat('asmFiles\\'+file)
    with codecs.open('asmFiles\\'+file,'r',encoding='cp1252',errors ='replace') as con
        num_of_lines = len(content.readlines())
    # 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)
        num_lines.append(num_of_lines)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes, 'filelines':num_lines, 'Clas
print (asm_size_byte.head())
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 10868/10868 [34:15<00:00, 14.27it/s]

                     ID       size  filelines  Class
0  01azqd4InC7m9JpocGv5  56.229886    1392154      9
1  01IsoiSMh5gxyDYTl4CB  13.999378     161528      2
2  01jsnpXSAlgw6aPeDxrU   8.507785      70960      9
3  01kcPWA9K2BOxQeS5Rju   0.078190       1276      1
4  01SuzwMJEIXsK7A8dQbl   0.996723      15282      8
```

4.2.1.2.1 Distribution of .asm file sizes

In [38]:

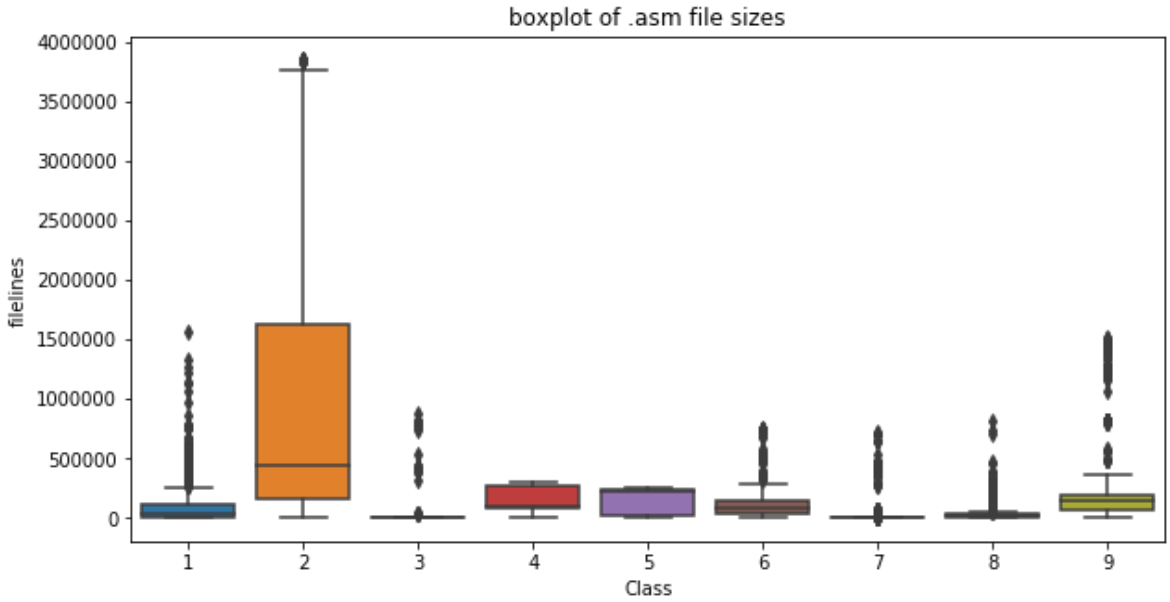
```
#boxplot of asm files
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="size", data=asm_size_byte)
plt.title("boxplot of .asm file sizes")
plt.show()
```



4.2.1.2.2 Distribution of file row numbers of .asm files

In [39]:

```
#boxplot of asm files
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="filelines", data=asm_size_byte)
plt.title("boxplot of .asm file sizes")
plt.show()
```



In [40]:

```
# 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['.dataweighted'] = result_asm['.data:']*4
result_asm.head()
```

```
(10868, 59)
(10868, 4)
```

Out[40]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	
1	02IOCvYEy8mjiuAQHax3	17	838	0	41	77840	0	0	0	
2	04EjIdbPV5e1XroFOpiN	19	60476	0	349	3760	0	0	0	
3	05EeG39MTRrI6VY21DPd	17	11119	0	323	1047	0	3385	0	
4	05LHG8fR3iPn6agIo9z7	17	0	0	178	0	0	511	0	

5 rows × 62 columns

In [41]:

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

Out[41]:

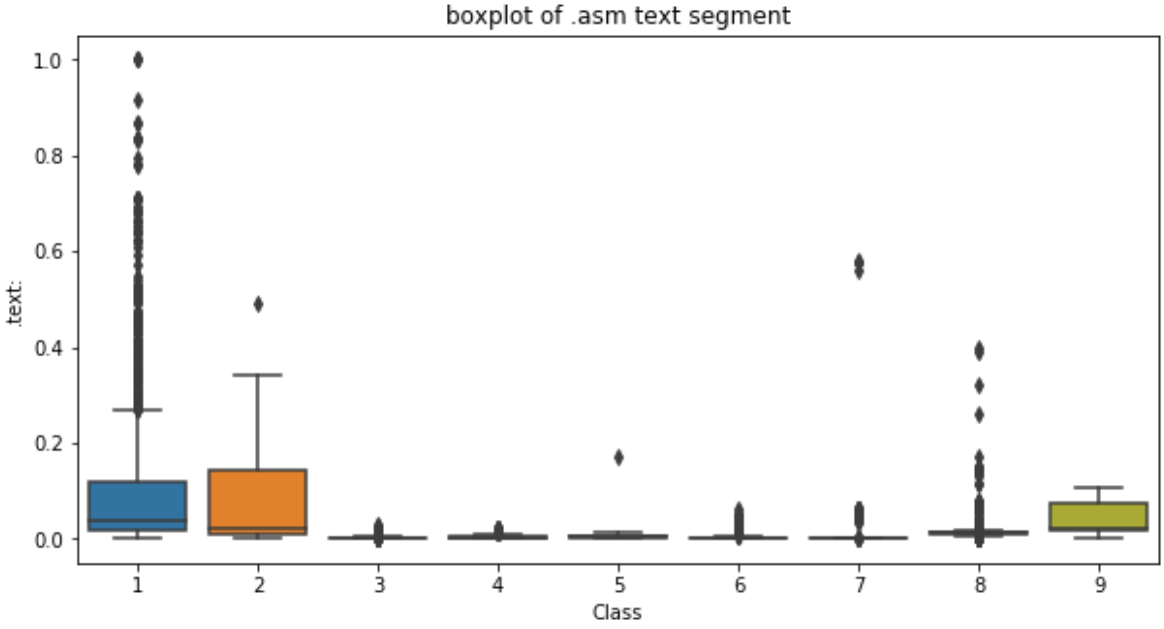
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.ec
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	02IOCvYEy8mjiuAQHax3	0.096045	0.001230	0.0	0.000246	0.030916	0.0	0.000000	
2	04EjIdbPV5e1XroFOpiN	0.107345	0.088779	0.0	0.002091	0.001493	0.0	0.000000	
3	05EeG39MTRrI6VY21DPd	0.096045	0.016323	0.0	0.001935	0.000416	0.0	0.000882	
4	05LHG8fR3iPn6agIo9z7	0.096045	0.000000	0.0	0.001066	0.000000	0.0	0.000133	

5 rows × 62 columns

4.2.2 Univariate analysis on asm file features

In [42]:

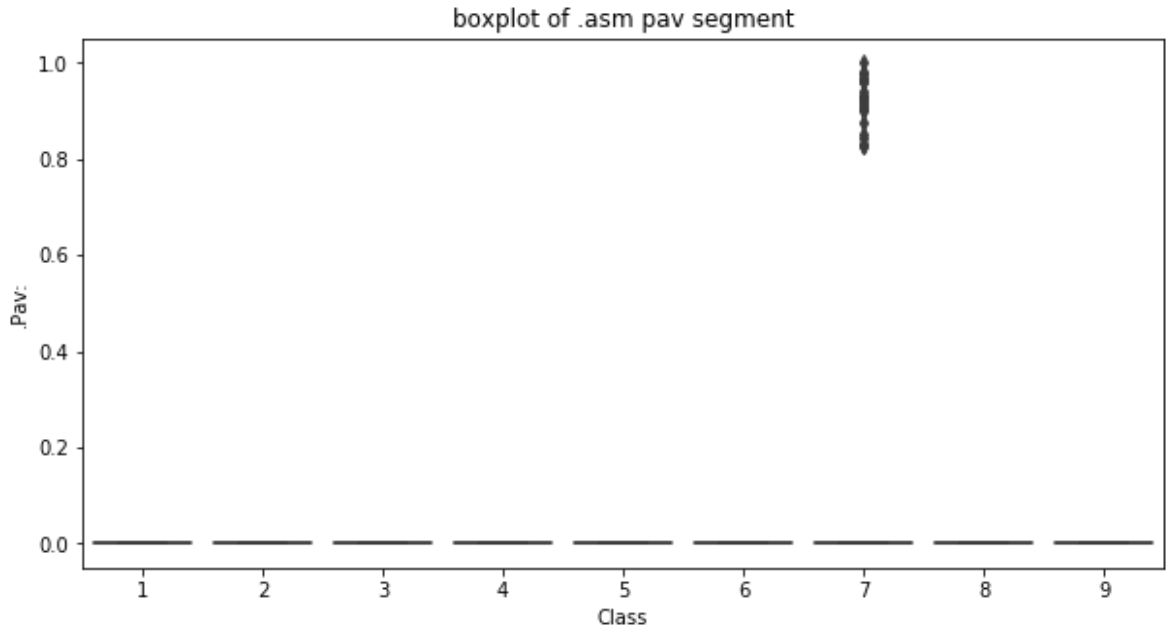
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```



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

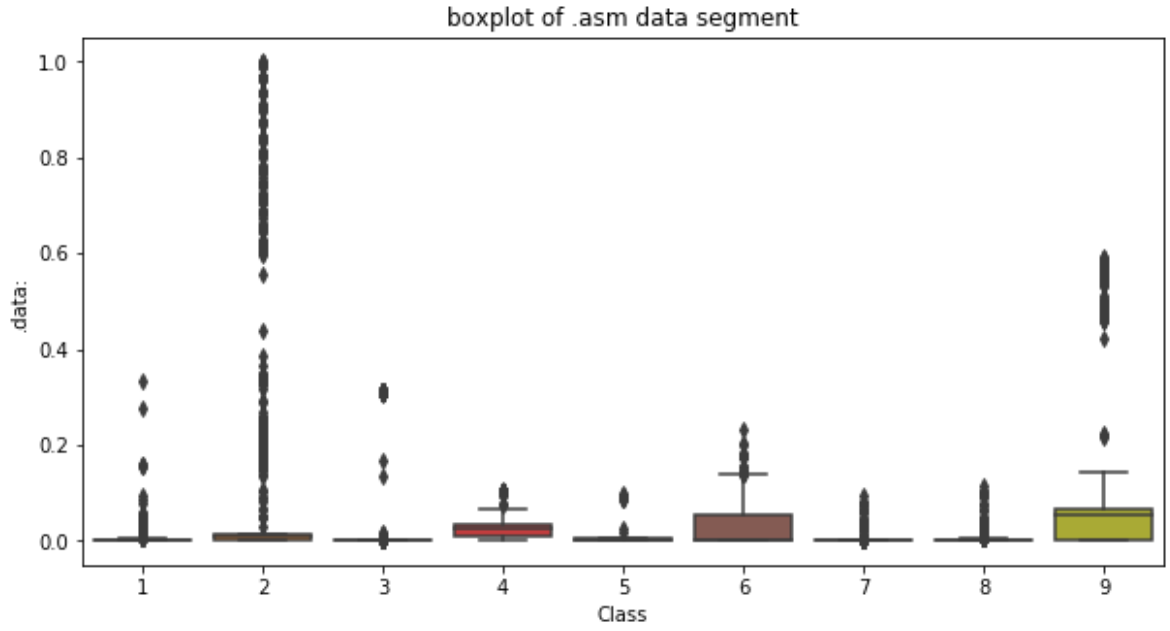
In [43]:

```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```



In [44]:

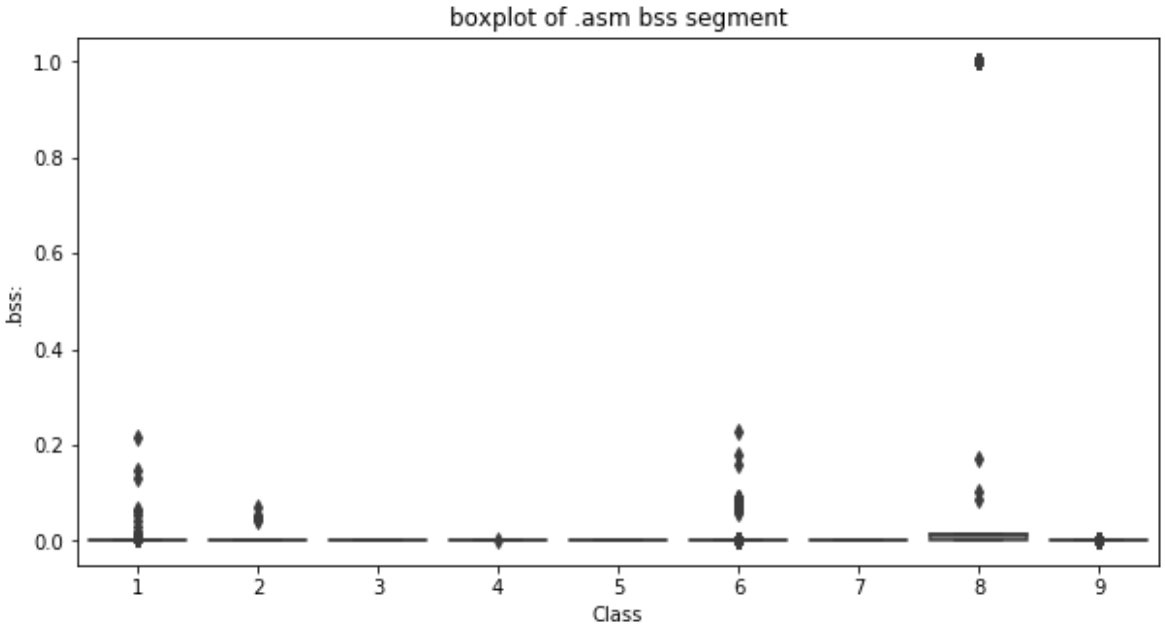
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



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

In [45]:

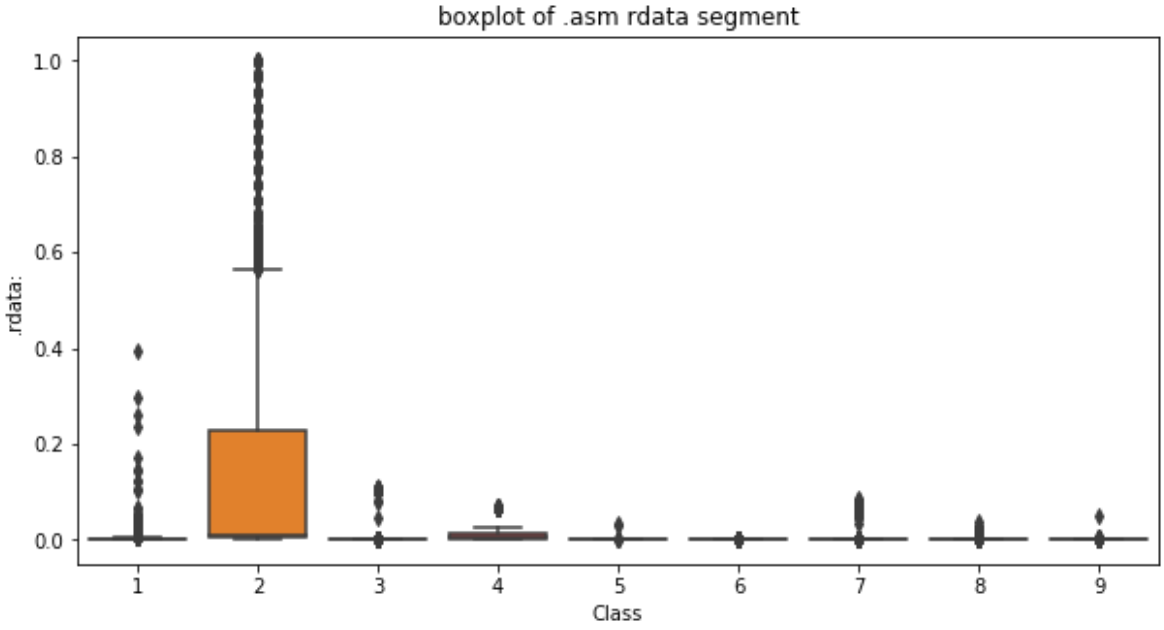
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [46]:

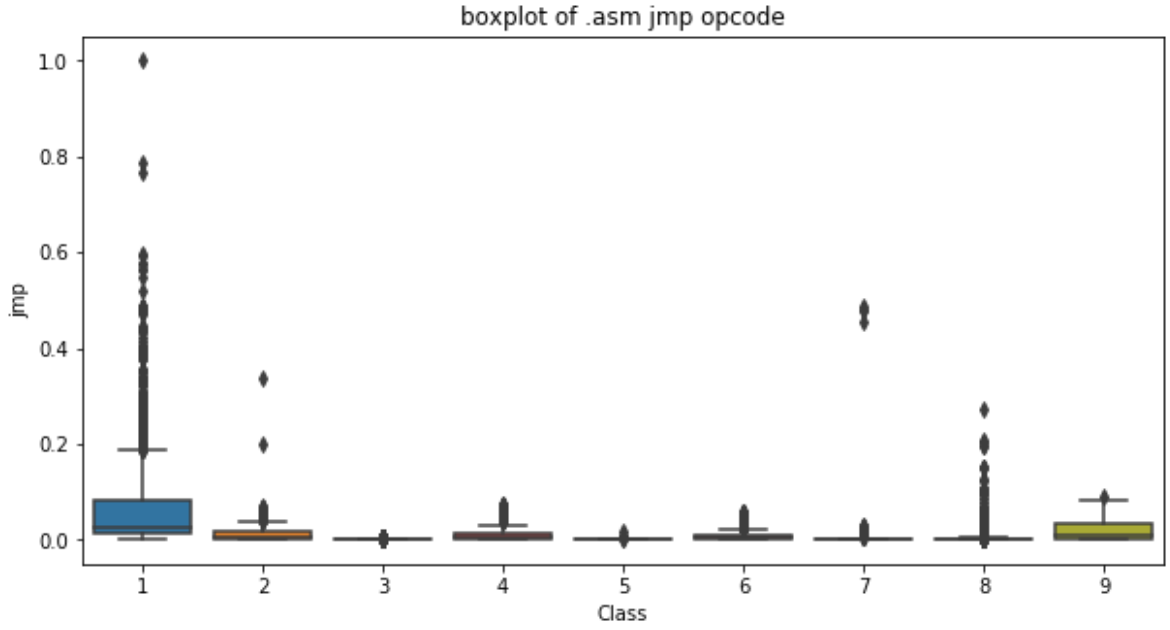
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

In [47]:

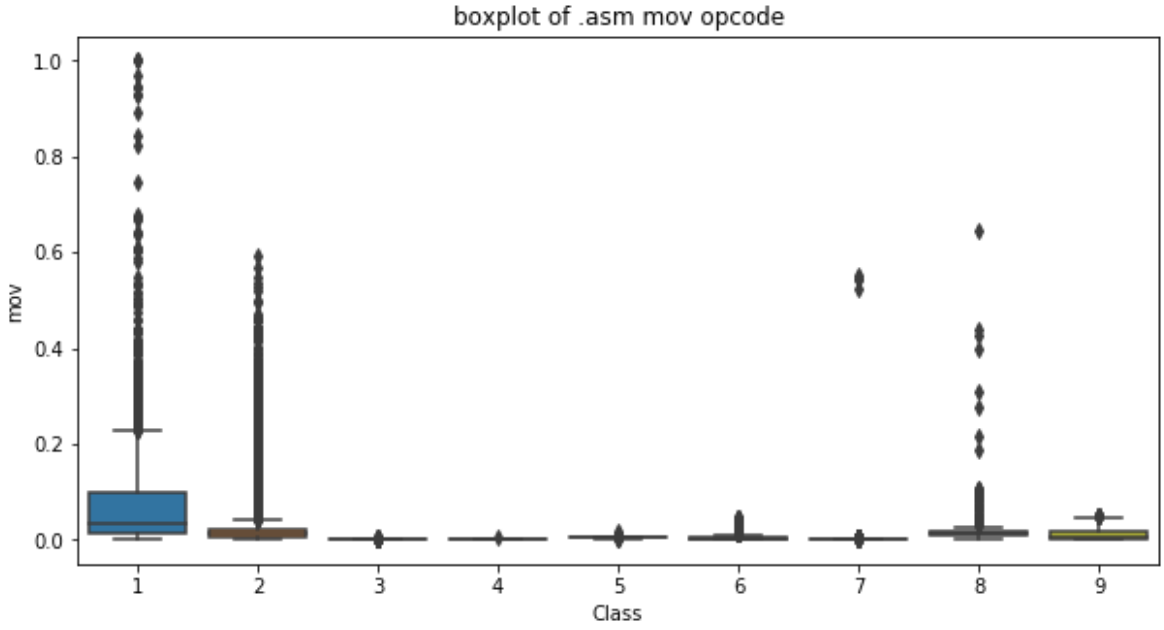
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



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

In [48]:

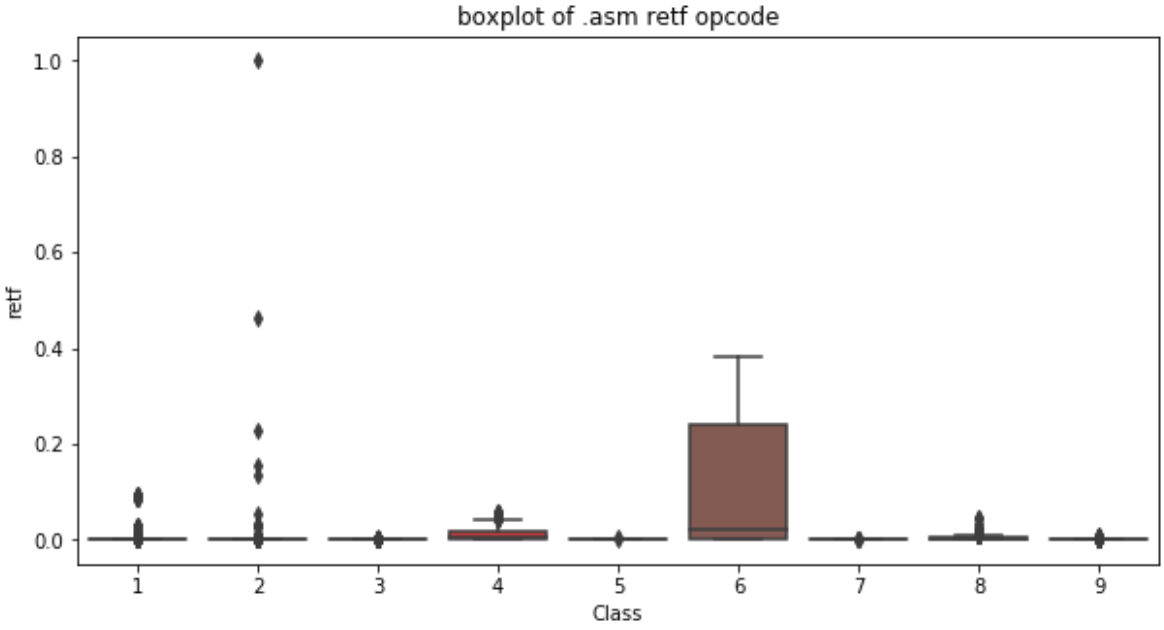
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



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

In [49]:

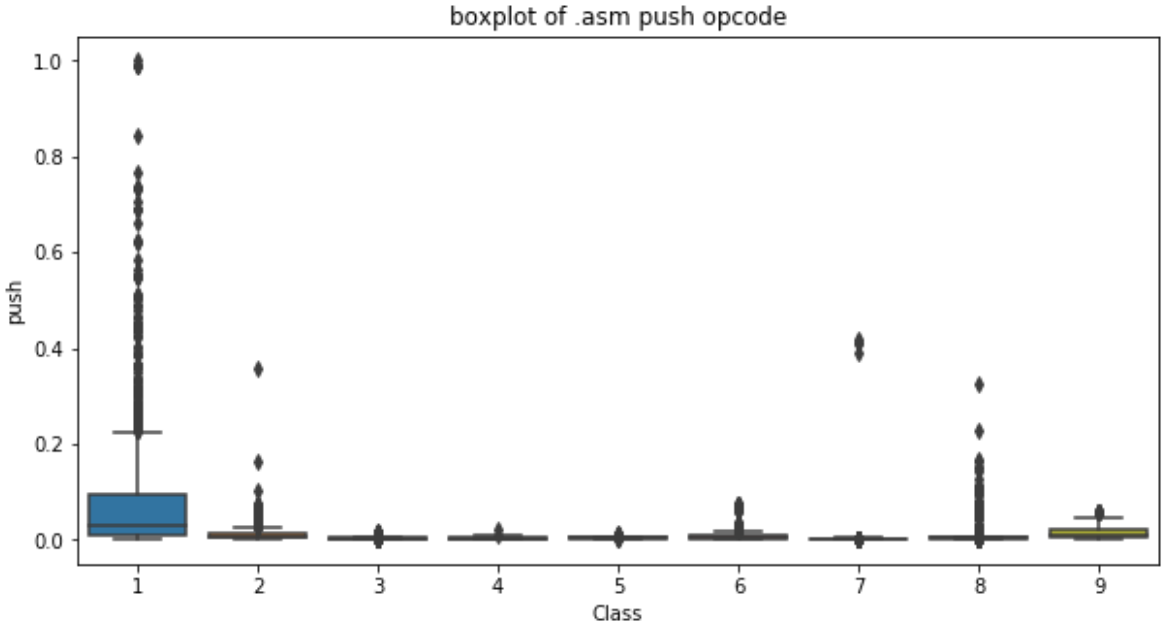
```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```



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

In [50]:

```
plt.figure(figsize = (10,5))
ax = sns.boxplot(x="Class", y="push", data=result_asm)
plt.title("boxplot of .asm push opcode")
plt.show()
```

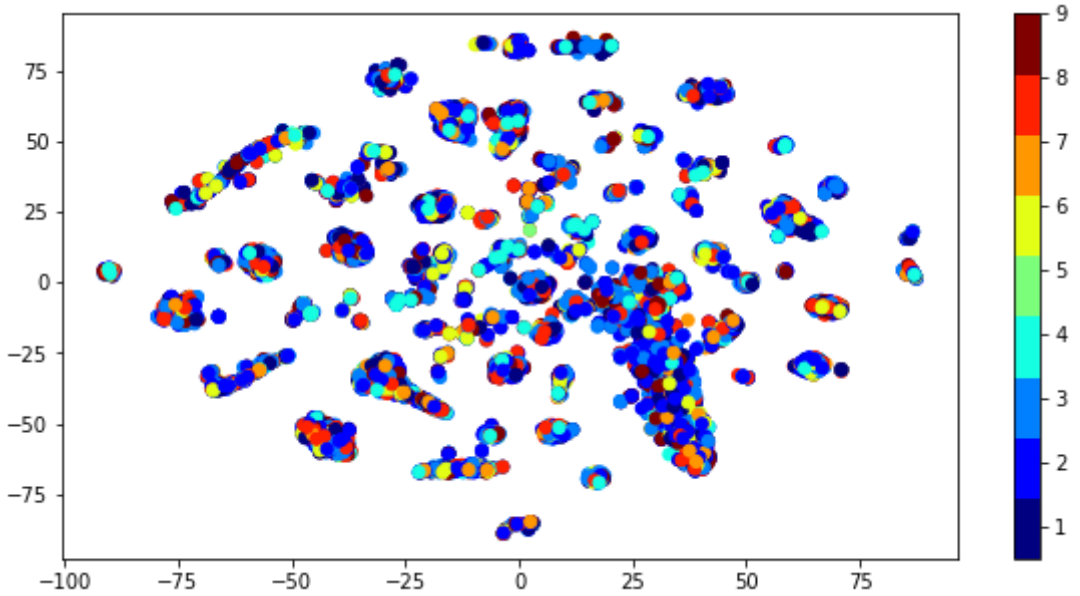


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

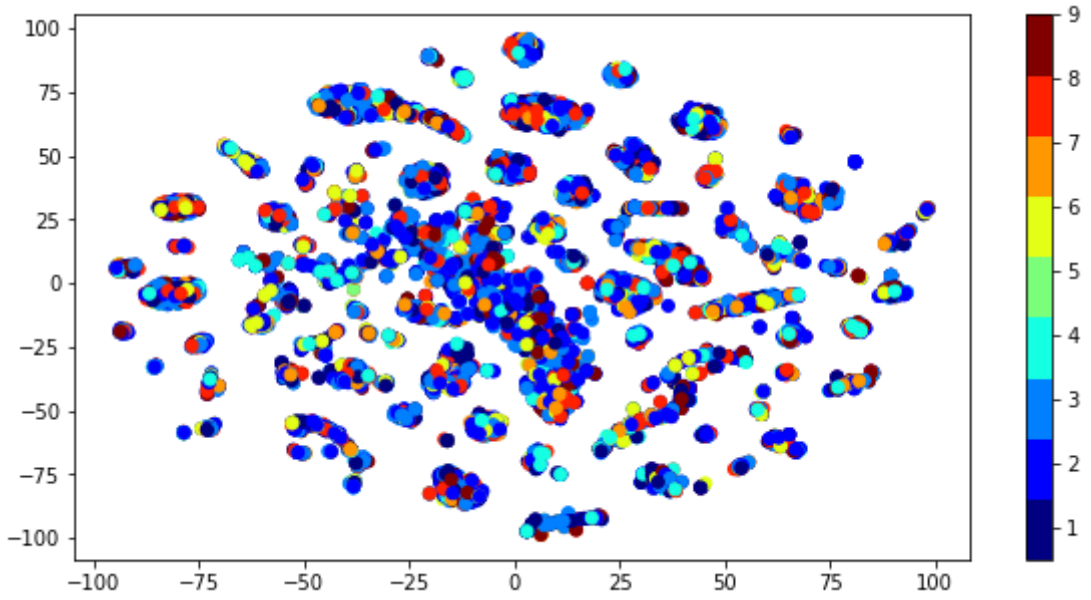
In [51]:

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

#multivariate analysis on asm files
#this is with perplexity 50
plt.figure(figsize = (10,5))
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_asm.drop(['ID','Class'], axis=1).fillna(0))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



In [52]:

```
# by univariate analysis on the .asm file features we are getting very negligible infor
# 'rtn', '.BSS:' '.CODE' features, so here we are trying multivariate analysis after re
# the plot looks very messy
plt.figure(figsize = (10,5))
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.CODE','si
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [53]:

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

In [54]:

```
asm_x.rename(columns = {'[':'bracket','-':'minussign','+':'plussign','*':'mulsign','?'
```

In [55]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,strat
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_a
```

In [56]:

```
1 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
bracket          False
minussign        False
plussign         False
mulsign          False
questionmark     False
atsign           False
```

```
size             False
filelines        False
.dataweighted    False
dtype: bool
```

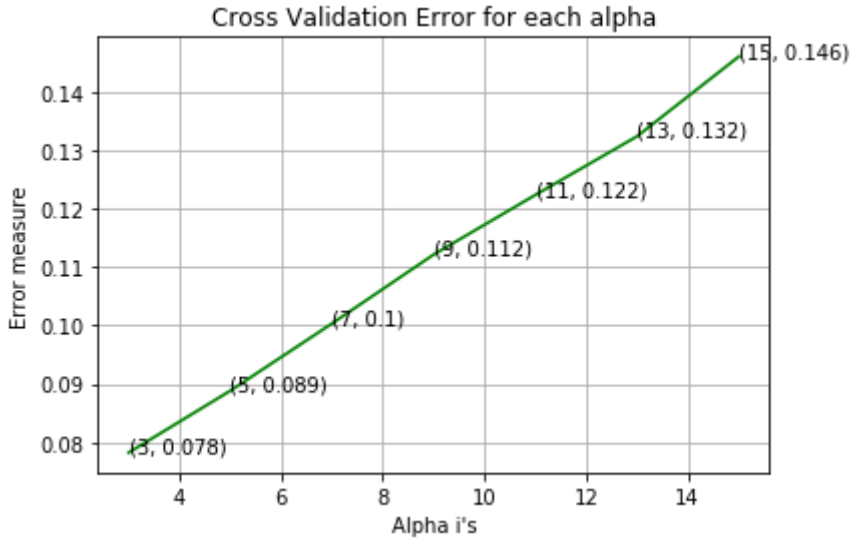
4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

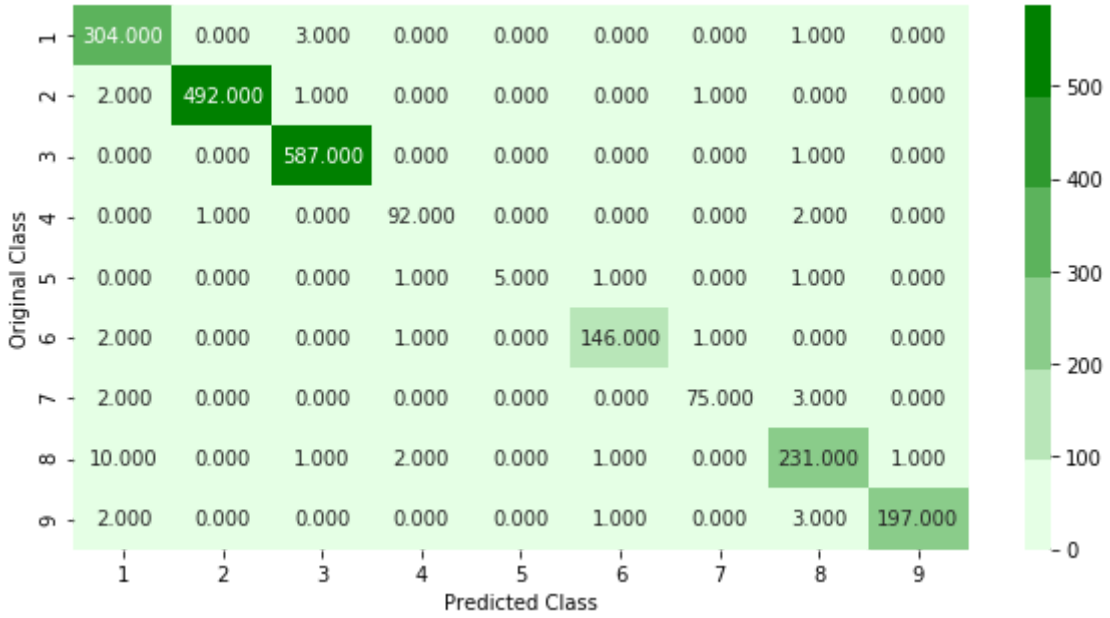
In [57]:

```
alpha = [x for x in [3,5,7,9,11,13,15]]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm,y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps

for i in zip(alpha,cv_log_error_array):
    print ('log_loss for k = ',i[0],'is',i[1])

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)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
plot_confusion_matrix(y_test_asm, sig_clf.predict(X_test_asm),'KNN(n = {})'.format(alp
```

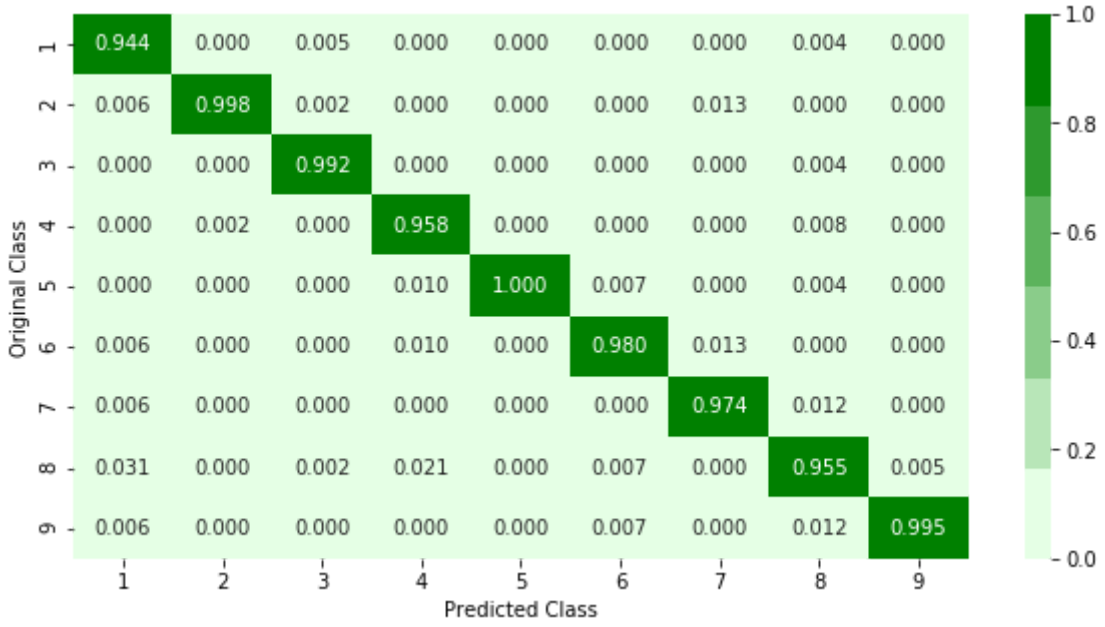
```
log_loss for k =  3 is 0.07822634324887906
log_loss for k =  5 is 0.08882553980508383
log_loss for k =  7 is 0.10034291193361768
log_loss for k =  9 is 0.1121531486151857
log_loss for k =  11 is 0.1223683462308549
log_loss for k =  13 is 0.1324030327413686
log_loss for k =  15 is 0.14605447214817008
```



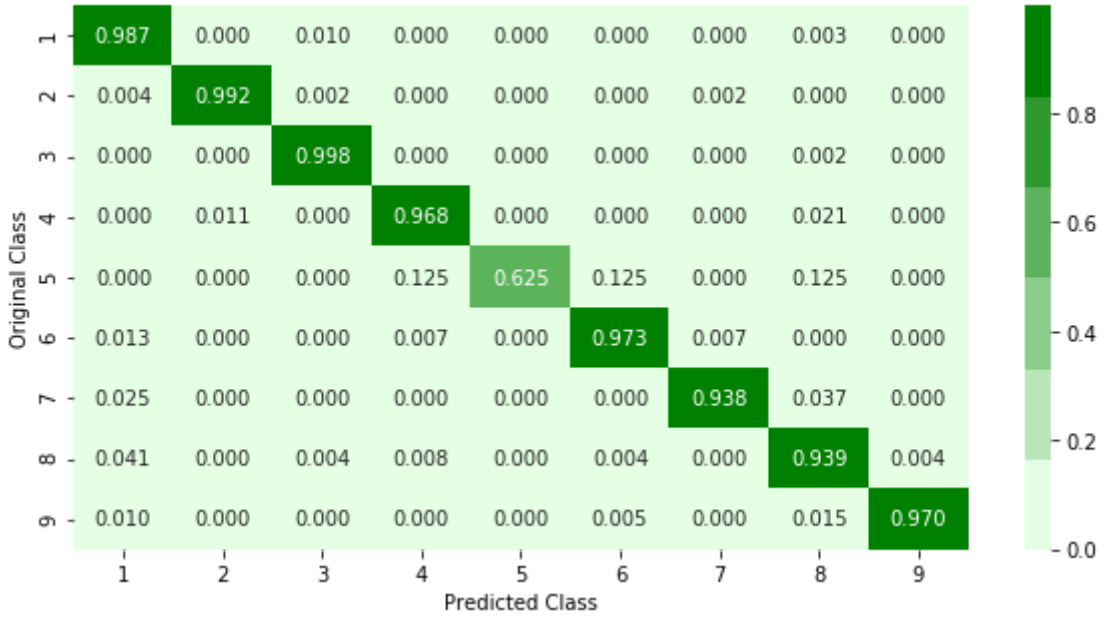

```
For values of best alpha =  3 The train log loss is: 0.051252139577566594
For values of best alpha =  3 The cross validation log loss is: 0.0782263432
4887906
For values of best alpha =  3 The test log loss is: 0.08829219350799569
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```




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




```
---------------------------------------------------- Recall matrix -----------
---------------------------------------
```



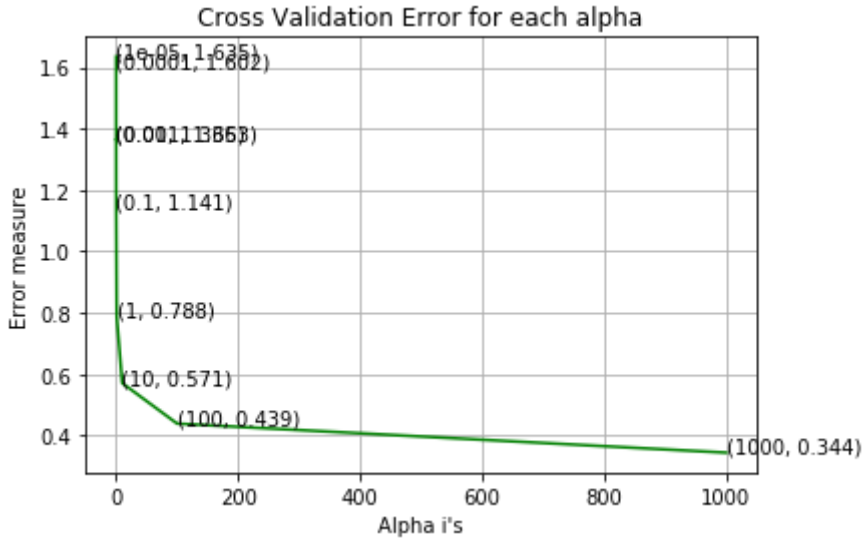

```
Percentage of misclassified points using KNN(n = 3) Model:  2.07%
```

4.4.2 Logistic Regression

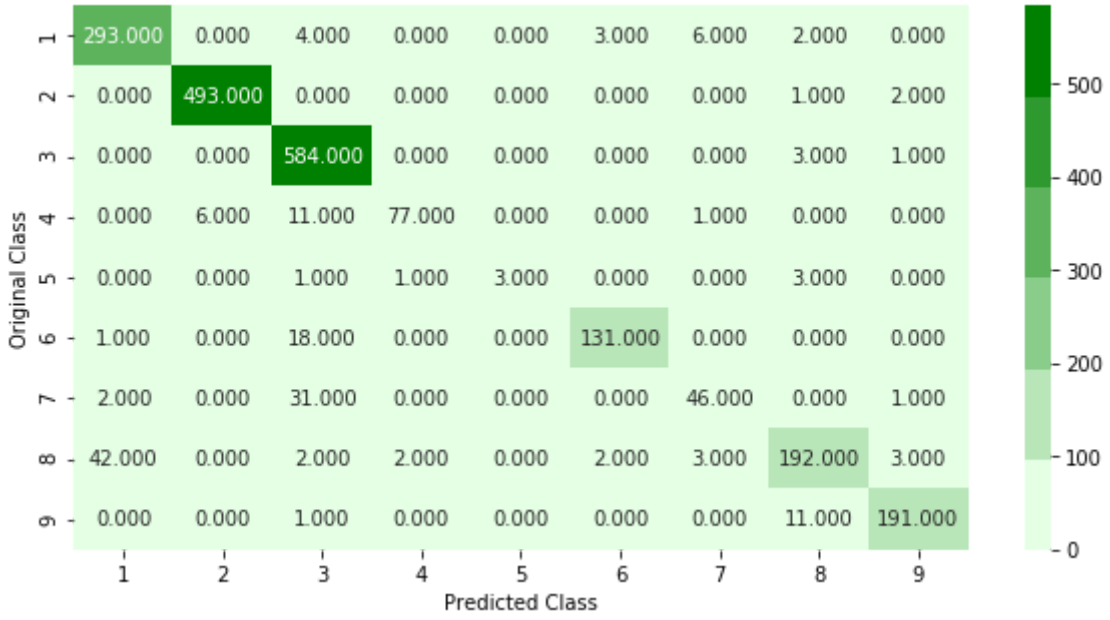
In [58]:

```
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.cla
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',log_loss(y_cv_asm, predict_y, labels=logisticR.classes_,
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',log_loss(y_test_asm, predict_y, labels=logisticR.class
plot_confusion_matrix(y_test_asm, sig_clf.predict(X_test_asm),'Logistic Regression (C =
```

```
log_loss for c =  1e-05 is 1.6353018273097308
log_loss for c =  0.0001 is 1.601693313639587
log_loss for c =  0.001 is 1.3634943675721063
log_loss for c =  0.01 is 1.3649332524062547
log_loss for c =  0.1 is 1.1408752185228466
log_loss for c =  1 is 0.7875378732127429
log_loss for c =  10 is 0.5707092238819212
log_loss for c =  100 is 0.43859802179195917
log_loss for c =  1000 is 0.3438629524532219
```

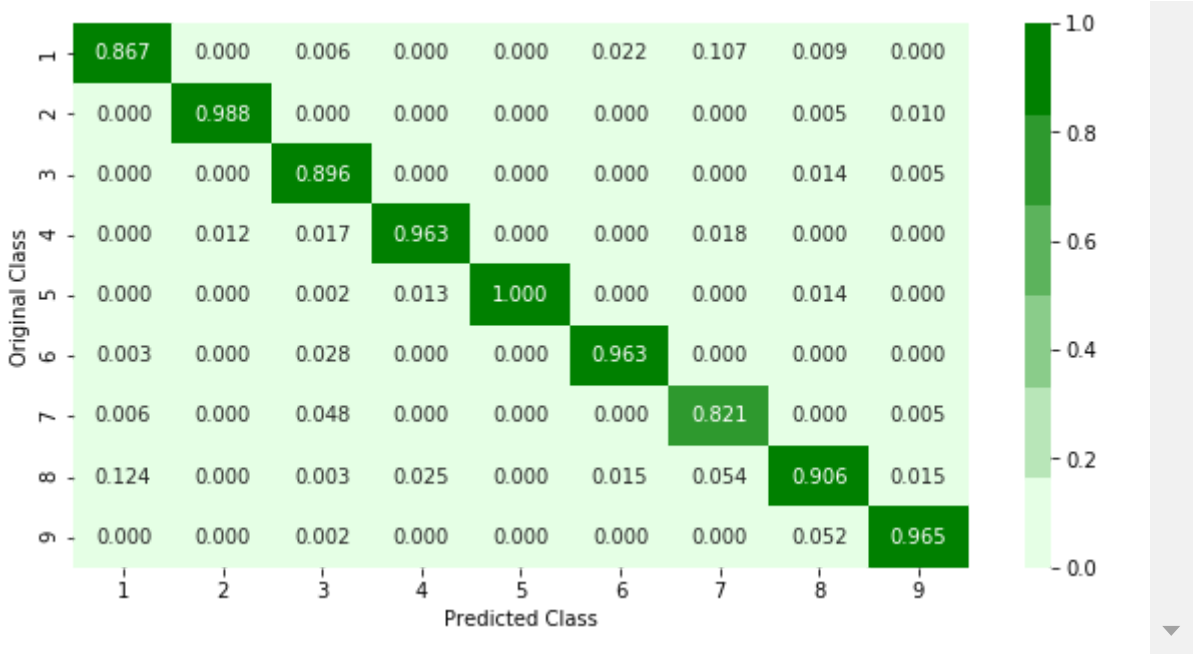



```
log loss for train data 0.3281465886689375
log loss for cv data 0.3438629524532219
log loss for test data 0.3478471628021207
---------------------------------------------------- Confusion matrix --------
-----------------------------------------
```

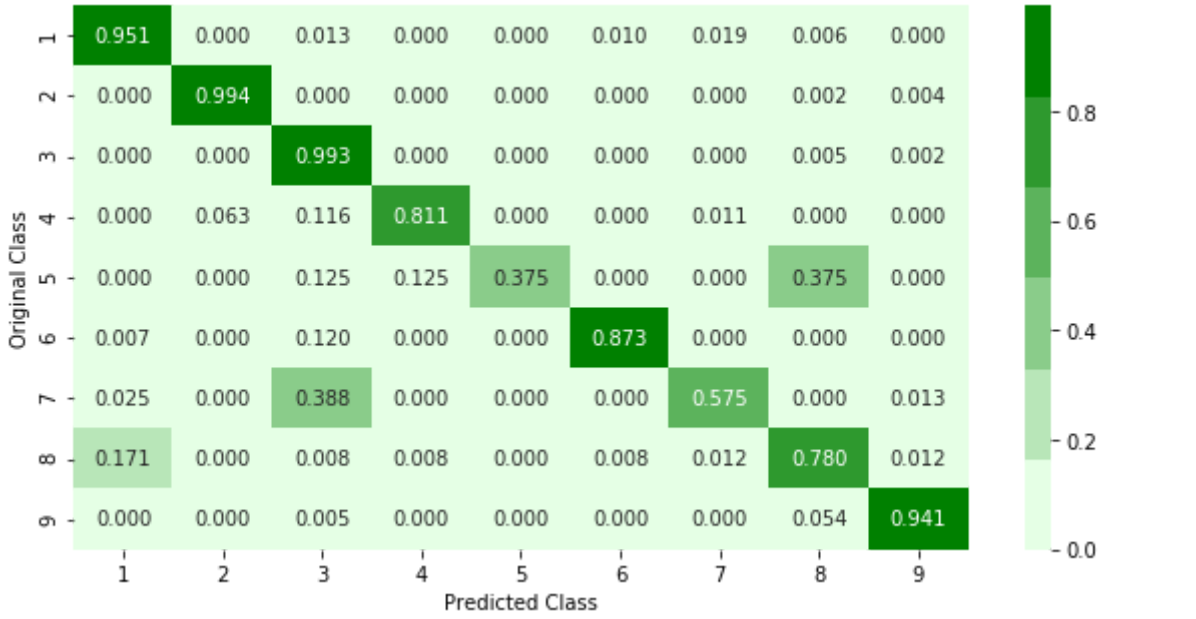


	1	2	3	4	5	6	7	8	9
1	293.000	0.000	4.000	0.000	0.000	3.000	6.000	2.000	0.000
2	0.000	493.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000
3	0.000	0.000	584.000	0.000	0.000	0.000	0.000	3.000	1.000
4	0.000	6.000	11.000	77.000	0.000	0.000	1.000	0.000	0.000
5	0.000	0.000	1.000	1.000	3.000	0.000	0.000	3.000	0.000
6	1.000	0.000	18.000	0.000	0.000	131.000	0.000	0.000	0.000
7	2.000	0.000	31.000	0.000	0.000	0.000	46.000	0.000	1.000
8	42.000	0.000	2.000	2.000	0.000	2.000	3.000	192.000	3.000
9	0.000	0.000	1.000	0.000	0.000	0.000	0.000	11.000	191.000

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



---------- Recall matrix ----------



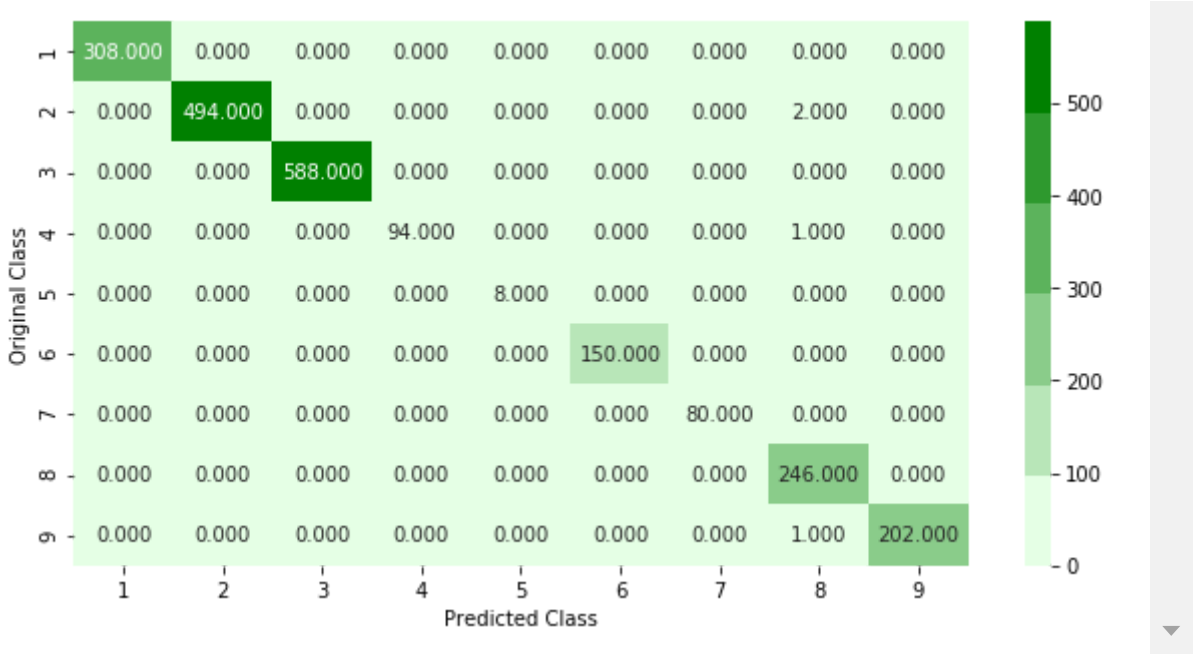
Percentage of misclassified points using Logistic Regression (C = 1000) Model: 7.54%

4.4.3 Random Forest Classifier

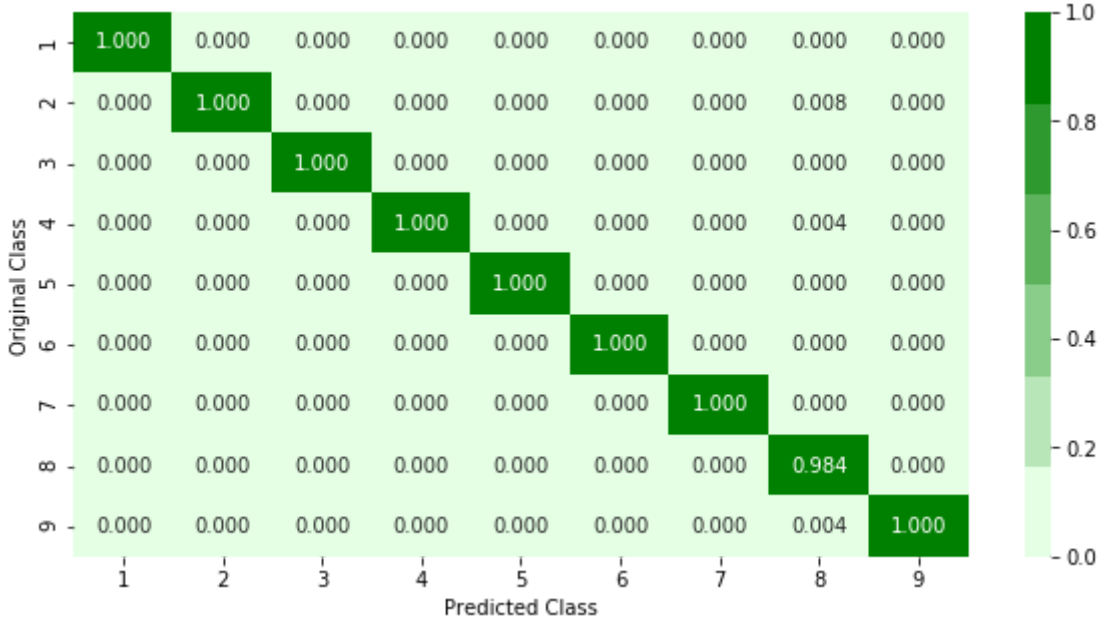
In [59]:

```
if not os.path.exists('RF_one_param_tuning_asm.joblib'):
    alpha=[10,50,100,500,1000,2000,3000]
    cv_log_error_array=[]
    train_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_,

    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)

    joblib.dump(sig_clf,'RF_one_param_tuning_asm.joblib')
else:
    sig_clf = joblib.load('RF_one_param_tuning_asm.joblib')

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

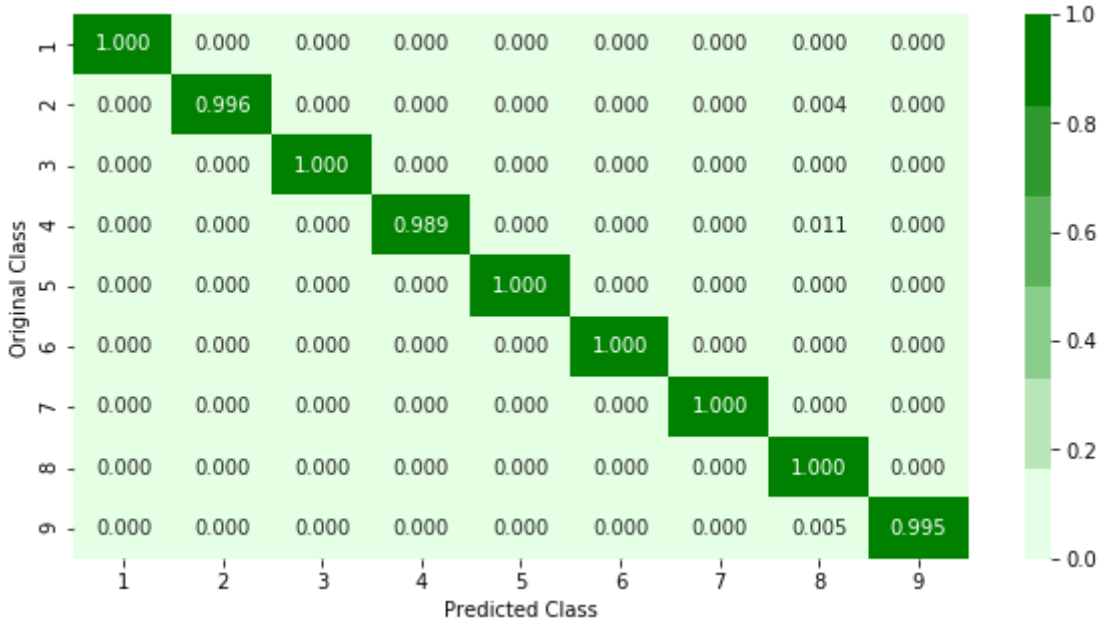
```
For values of best alpha =  1000 The train log loss is: 0.02130763882281661
For values of best alpha =  1000 The cross validation log loss is: 0.0217788
30232004408
For values of best alpha =  1000 The test log loss is: 0.023422488727851302
---------------------------------------------------- Confusion matrix --------
----------------------------------------
```



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



---------- Recall matrix ----------



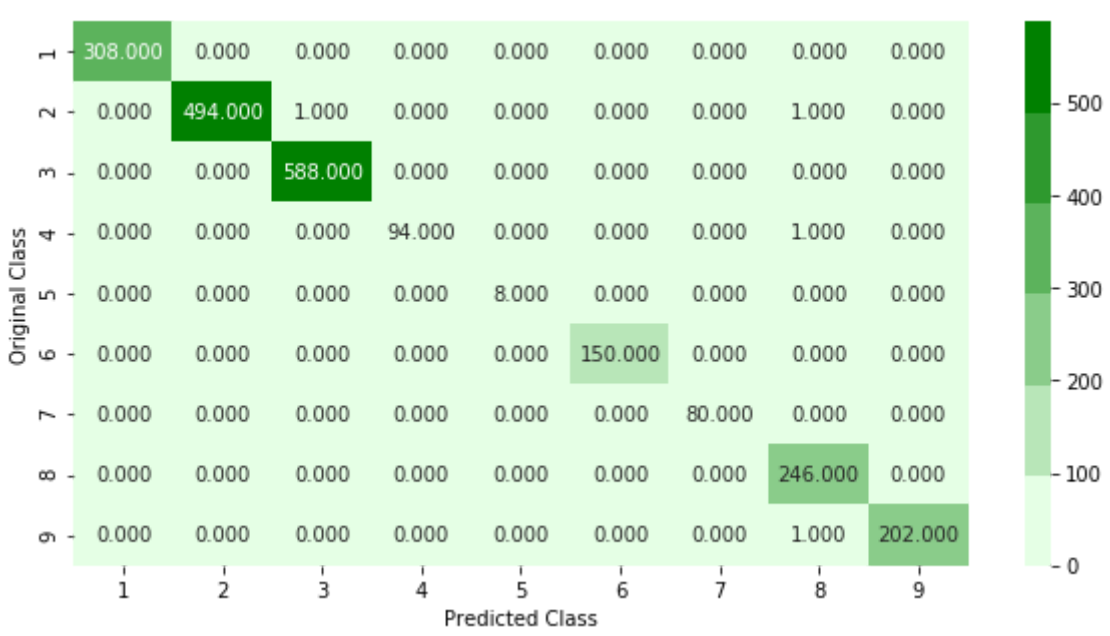

```
Percentage of misclassified points using Random Forest (n_estimators = 1000)
Model:  0.18%
```

4.4.4 XgBoost Classifier

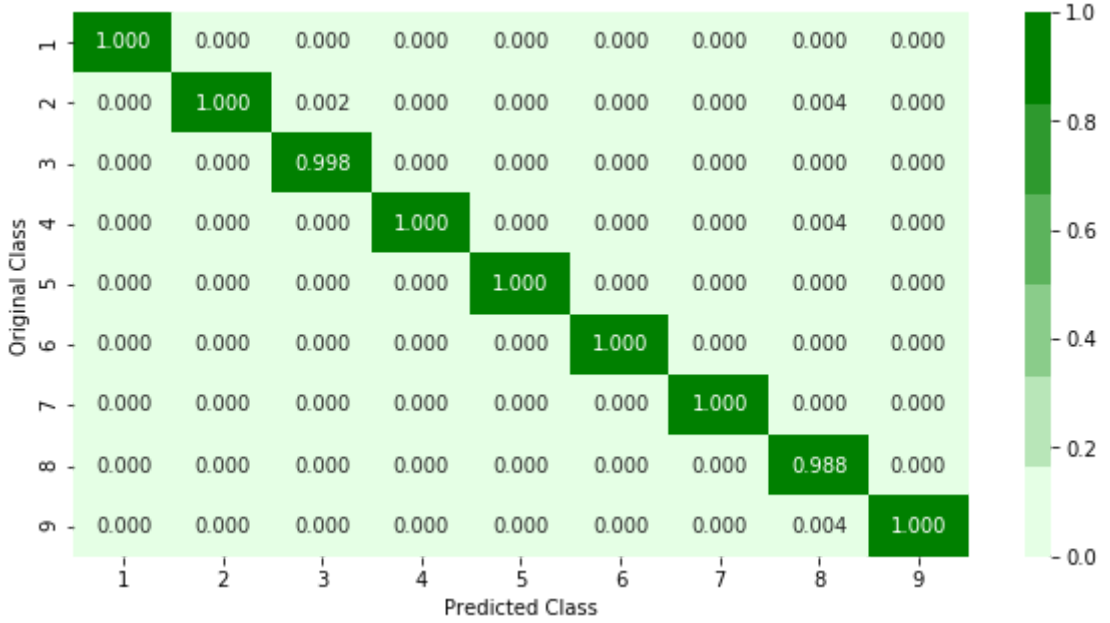
In [60]:

```
if not os.path.exists('XGB_one_param_tuning_asm.joblib'):
    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_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_,

    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)

    joblib.dump(sig_clf,'XGB_one_param_tuning_asm.joblib')
else:
    sig_clf = joblib.load('XGB_one_param_tuning_asm.joblib')

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

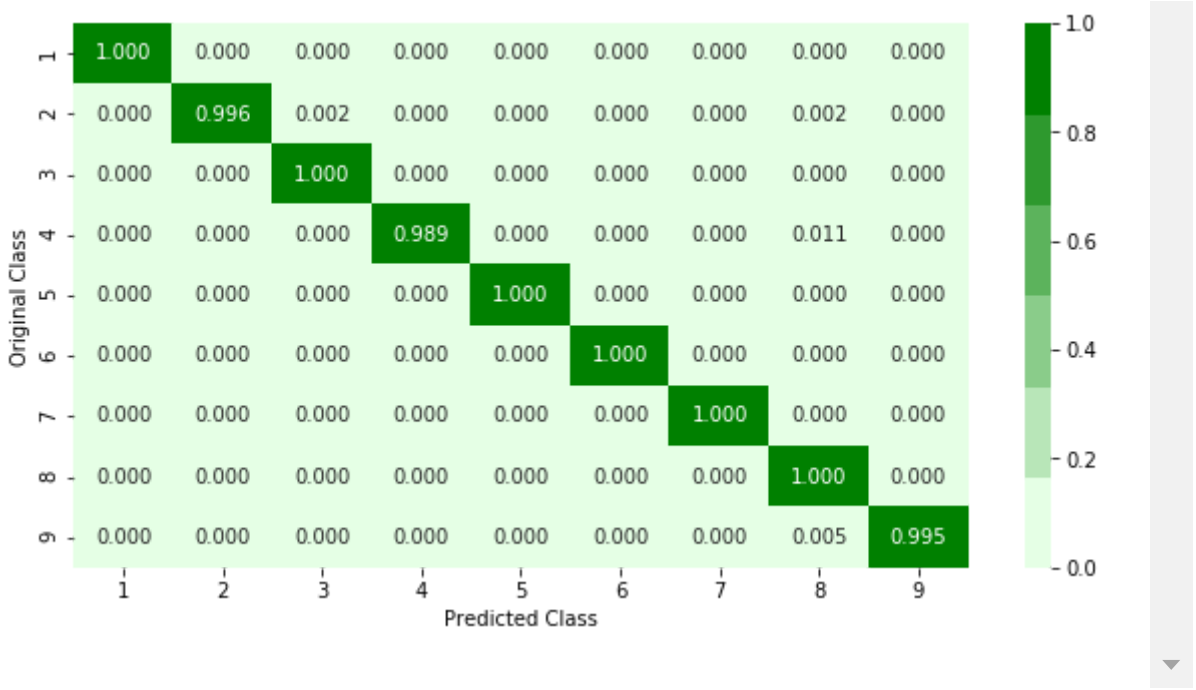
```
For values of best alpha =  1000 The train log loss is: 0.016417298234131933
For values of best alpha =  1000 The cross validation log loss is: 0.0165180
29039938188
For values of best alpha =  1000 The test log loss is: 0.01880318519615657
---------------------------------------------------- Confusion matrix --------
----------------------------------------
```



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



---------- Recall matrix ----------




```
Percentage of misclassified points using XGB Model (n_estimators = 1000) Mod
el:  0.18%
```

4.4.5 Xgboost Classifier with best hyperparameters

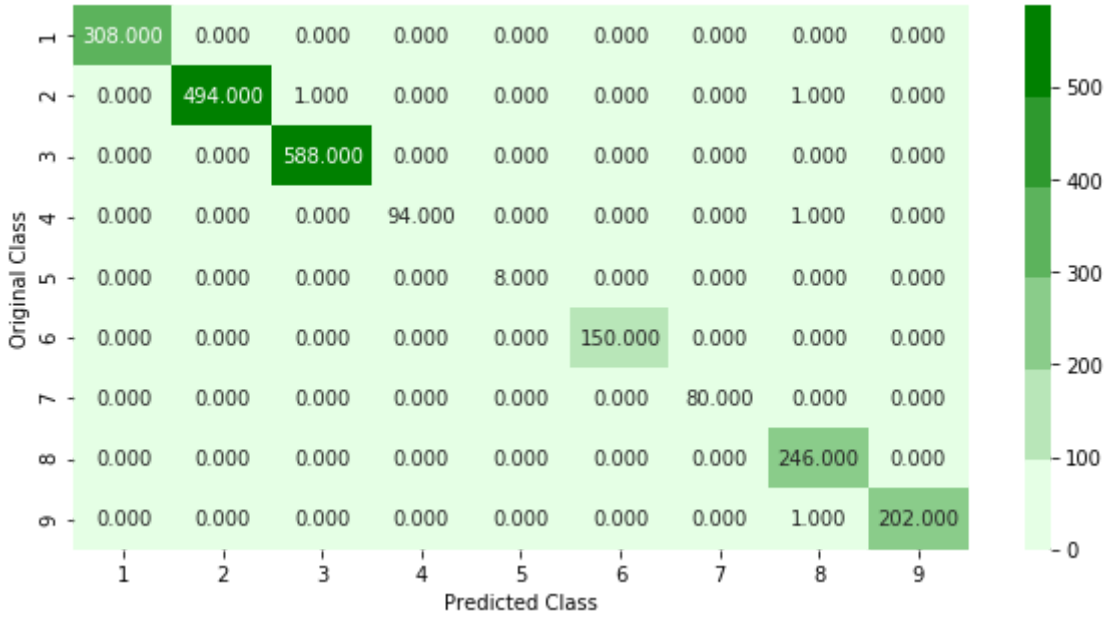
In [61]:

```
if not os.path.exists('XGB_multiple_param_tuning_asm.joblib'):
    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,n_iter=10,verbose=1(
    random_cfl1.fit(X_train_asm,y_train_asm)
    joblib.dump(random_cfl1, 'XGB_multiple_param_tuning_asm.joblib')
else:
    random_cfl1 = joblib.load('XGB_multiple_param_tuning_asm.joblib')
```

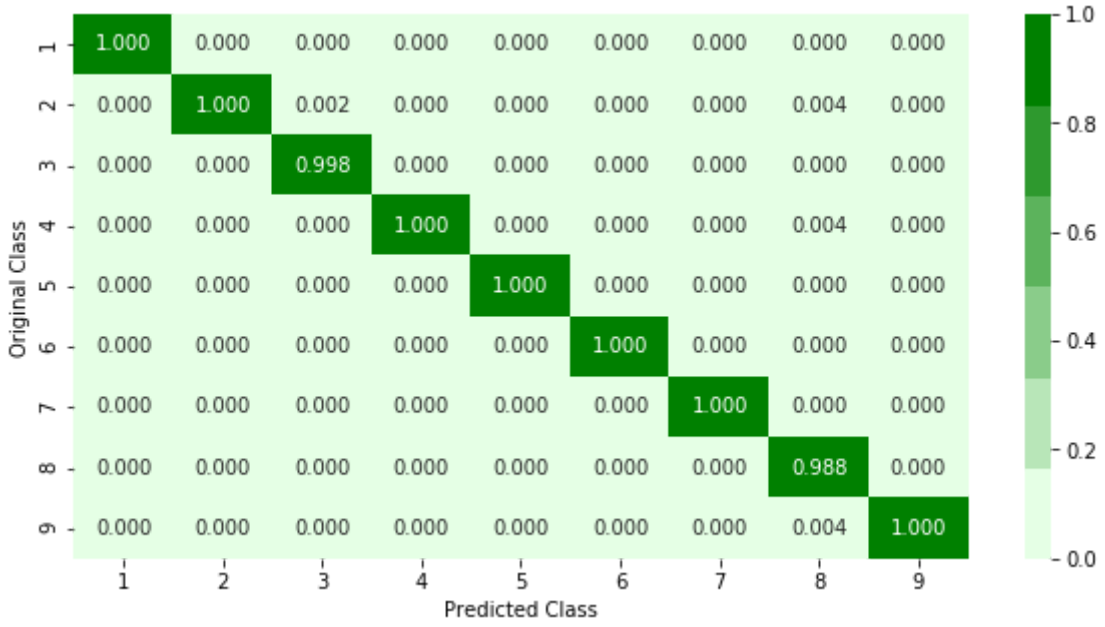
In [62]:

```
if not os.path.exists('XGB_multiple_param_tuning_Calibrated_asm.joblib'):
    c_cfl = CalibratedClassifierCV(random_cfl1.best_estimator_ ,method='sigmoid')
    c_cfl.fit(X_train_asm,y_train_asm)
    joblib.dump(c_cfl, 'XGB_multiple_param_tuning_Calibrated_asm.joblib')
else:
    c_cfl = joblib.load('XGB_multiple_param_tuning_Calibrated_asm.joblib')

predict_y = c_cfl.predict_proba(X_train_asm)
print ("The train log loss is:",log_loss(y_train_asm, predict_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print("The cross validation log loss is:",log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print("The test log loss is:",log_loss(y_test_asm, predict_y))
plot_confusion_matrix(y_test_asm, c_cfl.predict(X_test_asm),'XGB Model (with more para
```

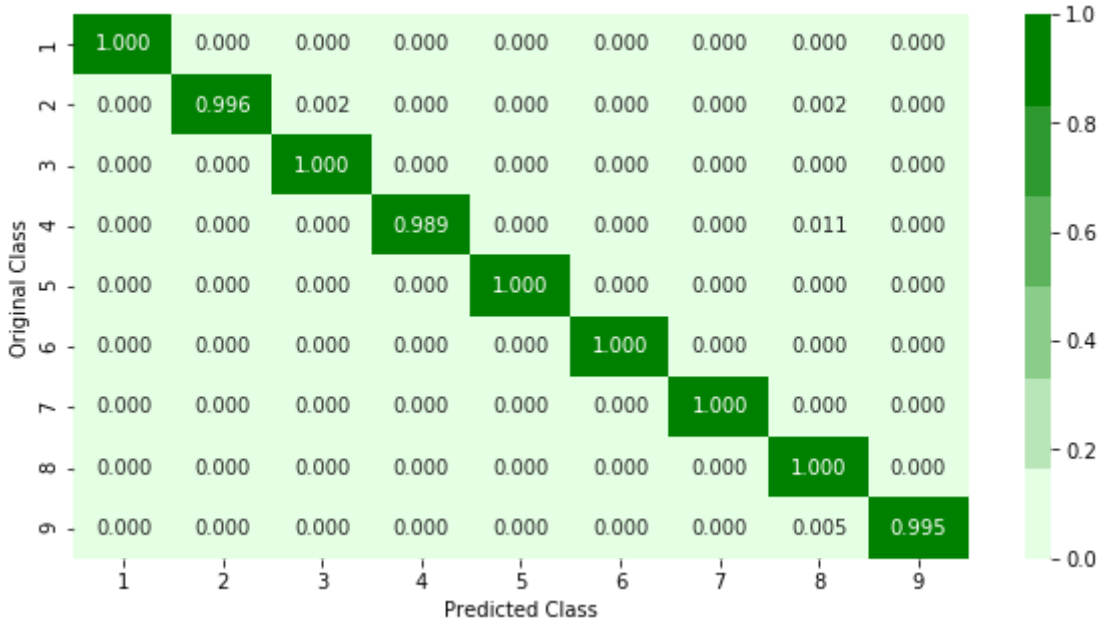
```
The train log loss is: 0.01717220458840472
The cross validation log loss is: 0.01829959740822653
The test log loss is: 0.019137559973987937
---------------------------------------------------- Confusion matrix --------
------------------------------------------
```




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




```
---------------------------------------------------- Recall matrix -----------
---------------------------------------
```




```
Percentage of misclassified points using XGB Model (with more parameters tun
ed) Model:  0.18%
```

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

4.5.1. Merging both asm and byte file features

In [63]:

```
1 result.head()
```

Out[63]:

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

5 rows × 266 columns

In [64]:

```
1 result.rename(columns={'filelines':'bytefilelines'},inplace=True)
```

In [65]:

```
1 result_asm.head()
```

Out[65]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.ec
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	
1	02IOCvYEy8mjiuAQHax3	0.096045	0.001230	0.0	0.000246	0.030916	0.0	0.000000	
2	04EjIdbPV5e1XroFOpiN	0.107345	0.088779	0.0	0.002091	0.001493	0.0	0.000000	
3	05EeG39MTRrI6VY21DPd	0.096045	0.016323	0.0	0.001935	0.000416	0.0	0.000882	
4	05LHG8fR3iPn6agIo9z7	0.096045	0.000000	0.0	0.001066	0.000000	0.0	0.000133	

5 rows × 62 columns

In [66]:

```
1 result_asm.rename(columns = {'filelines':'asmfilelines','[':'bracket','-':'minussign',
```

In [67]:

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

```
(10868, 266)
(10868, 62)
```

In [68]:

```
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(['Class'],axis=1)
result_x.head()
```

Out[68]:

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

5 rows × 325 columns

Dchad more features

In [69]:

```
dchad2 = pd.DataFrame()
for file in os.listdir('dchad/dchad2'):
    if file[-3:] == 'csv':
        filelocation = 'dchad/dchad2/'+file
        dchad2 = pd.concat(objs=[dchad2,pd.read_csv(filelocation)],axis=1)
dchad2.rename(columns={'filename':'ID'},inplace=True)
dchad2.head()
```

Out[69]:

	ID	edx	esi	es	ds	ss	cs	ah	al	ax	...	ASM_964	ASM_972
0	01IsoiSMh5gxyDYTI4CB	750	496	3	0	0	0	8	224	49	...	32	49
1	01SuzwMJEIXsK7A8dQbl	1121	24	3	1	4	2	6	22	7	...	48	9
2	01azqd4InC7m9JpocGv5	1493	1900	0	0	0	0	1	398	0	...	48	9
3	01jsnpXSAlgw6aPeDxrU	525	4	0	0	0	0	0	0	0	...	48	9
4	01kcPWA9K2BOxQeS5Rju	23	35	0	0	0	0	0	3	0	...	48	89

5 rows × 1276 columns

In [70]:

```
dchad2_fix = pd.DataFrame()
for column in dchad2.iteritems():
    if column[0] in dchad2_fix.columns.values:
        continue
    else:
        dchad2_fix[column[0]] = column[1]
dchad2_fix.head()
```

Out[70]:

	ID	edx	esi	es	ds	ss	cs	ah	al	ax	...	train_byte_p1	train_
0	01IsoiSMh5gxyDYTI4CB	750	496	3	0	0	0	8	224	49	...	1.0	(
1	01SuzwMJEIXsK7A8dQbl	1121	24	3	1	4	2	6	22	7	...	1.0	(
2	01azqd4InC7m9JpocGv5	1493	1900	0	0	0	0	1	398	0	...	1.0	(
3	01jsnpXSAlgw6aPeDxrU	525	4	0	0	0	0	0	0	0	...	1.0	(
4	01kcPWA9K2BOxQeS5Rju	23	35	0	0	0	0	0	3	0	...	1.0	(

5 rows × 653 columns

In [71]:

```
1 dchad2_fix = normalize(dchad2_fix)
```

In [72]:

```
result_x = pd.merge(result_x,dchad2_fix,on='ID', how='left')
result_x = result_x.drop(labels=['ID'],axis=1)
result_x = result_x.loc[:,[
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0a', '0b', '0c',
       '0d', '0e', '0f', '10', '11', '12', '13', '14', '15', '16', '17',
       '18', '19', '1a', '1b', '1c', '1d', '1e', '1f', '20', '21', '22',
       '23', '24', '25', '26', '27', '28', '29', '2a', '2b', '2c', '2d',
       '2e', '2f', '30', '31', '32', '33', '34', '35', '36', '37', '38',
       '39', '3a', '3b', '3c', '3d', '3e', '3f', '40', '41', '42', '43',
       '44', '45', '46', '47', '48', '49', '4a', '4b', '4c', '4d', '4e',
       '4f', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59',
       '5a', '5b', '5c', '5d', '5e', '5f', '60', '61', '62', '63', '64',
       '65', '66', '67', '68', '69', '6a', '6b', '6c', '6d', '6e', '6f',
       '70', '71', '72', '73', '74', '75', '76', '77', '78', '79', '7a',
       '7b', '7c', '7d', '7e', '7f', '80', '81', '82', '83', '84', '85',
       '86', '87', '88', '89', '8a', '8b', '8c', '8d', '8e', '8f', '90',
       '91', '92', '93', '94', '95', '96', '97', '98', '99', '9a', '9b',
       '9c', '9d', '9e', '9f', 'a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6',
       'a7', 'a8', 'a9', 'aa', 'ab', 'ac', 'ad', 'ae', 'af', 'b0', 'b1',
       'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8', 'b9', 'ba', 'bb', 'bc',
       'bd', 'be', 'bf', 'c0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7',
       'c8', 'c9', 'ca', 'cb', 'cc', 'cd', 'ce', 'cf', 'd0', 'd1', 'd2',
       'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'da', 'dc',
       'de', 'df', 'e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7',
       'e8', 'e9', 'ea', 'eb', 'ec', 'ed', 'ee', 'ef', 'f0', 'f1', 'f2',
       'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'fa', 'fb', 'fc', 'fd',
        'fe', 'ff', '??',
       'bytefilelines', 'vertex_count',
       'delta_max', 'density', 'HEADER:', '.idata:', '.rsrc:', '.reloc:',
       'retf','.dll', 'eip', 'asmfilelines', '.dataweighted',
       'esi', 'es', 'ds', 'ss', 'cs', 'ah', 'al', 'ax', 'bh', 'bl', 'bx',
       'ch', 'cl', 'cx', 'dh', 'dx', 'cdq', 'cld', 'cli',
       'cmc', 'const', 'daa', 'faddp', 'fchs', 'fdiv', 'fdivp', 'fild', 'fistp', 'fwor
       'jge', 'jl', 'jno', 'jo',
       'mul', 'neg', 'not', 'out', 'popf', 'proc', 'rcl', 'rep', 'retn',
       'sal', 'sar', 'sbb', 'setnle', 'setnz', 'setz',
       'shld', 'stc', 'sti',
       'wait', 'wcslen', '__vbaUI1I2', 'send',
        'IsDBCSLeadByte', 'SetWindowLongW', 'time', 'GetWindowLongW',
        'ValidateRect', 'socket', 'memmove', 'connect', 'lstrcpyW',
        'wcscmp', 'lstrcmpiW', 'wcscpy', 'GetUserDefaultLangID', 'Escape', 'LoadLibrary
       'WriteFile', 'GetModuleFileNameA', 'CloseHandle', 'RegCloseKey',
       'VirtualFree', 'GetLastError', 'FreeLibrary',
       'MultiByteToWideChar', 'RegQueryValueExA', 'GetTickCount',
       'GetStartupInfoA', 'RegOpenKeyExA', 'GetCurrentProcess',
       'EnterCriticalSection', 'LeaveCriticalSection', 'DeleteCriticalSection',
       'InitializeCriticalSection', 'LocalAlloc', 'RtlUnwind', 'lstrlenA',
       'GetDC', 'UnhandledExceptionFilter', 'GetLocaleInfoA',
       'TlsSetValue', 'TlsGetValue', 'RaiseException', 'TerminateProcess',
       'CharNextA', 'HeapAlloc', 'LocalFree', 'GetCPInfo', 'GetFileType',
        'HeapFree', 'DispatchMessageA', 'ShowWindow', 'DestroyWindow',
       'GetSystemMetrics', 'InterlockedIncrement', 'InterlockedDecrement',
       'LoadStringA', 'GetThreadLocale', 'SendMessageA', 'DeleteObject',
       'GetClientRect', 'GetWindowRect', 'wsprintfA',
       'SetLastError', 'QueryPerformanceCounter', 'GlobalUnlock',
       'TranslateMessage', 'HeapReAlloc', 'GetDeviceCaps', 'SetFocus',
       'CoTaskMemFree', '_initterm', 'CreatePopupMenu',
```

```
        'GetForegroundWindow', 'HeapSize', 'exit', 'free',
        'GetModuleHandleW', '_except_handler3', 'malloc', 'GetMenu',
        'VariantClear', 'GetSystemTime', 'GetMessagePos',
        'SetRect', 'memset', 'lstrlenW', 'memcpy', 'IsWindow',
         'EnableWindow', 'SetWindowPos', 'lstrcpynA', 'GetOEMCP',
        'SysFreeString', 'GetStringTypeW', 'lstrcpyA', 'GetWindowLongA',
        'LCMapStringW', 'LCMapStringA', 'SetHandleCount', 'GetStringTypeA',
        'GetEnvironmentStringsW', 'GetSysColor', 'RegisterClassA',
        'InvalidateRect', 'InterlockedExchange',
        'TlsAlloc', 'lstrcatA', 'CompareStringA',
        'ReleaseDC', 'CoCreateInstance', 'GetWindowTextA', 'lstrcmpiA',
        'GetDlgItem', 'TlsFree', 'GetParent', 'GetWindow', 'SetMenu',
        '__p__fmode', 'CompareStringW', '__getmainargs', 'sprintf',
        '_XcptFilter', 'DrawEdge',
        'GetIconInfo', '_controlfp', 'strstr', 'IsValidCodePage',
        'VariantInit', 'strncpy', '__CxxFrameHandler', 'GetCursor',
         'ReleaseMutex', 'strlen', '__vbaGenerateBoundsError', 'strchr',
        'strcpy', 'Rectangle', 'fclose', 'rand', 'strcat',
        'GetMessageTime', 'SendMessageW', 'SendDlgItemMessageA',
        'LoadStringW', 'Virtual', 'Offset', 'Import', 'var', 'UINT',
        'LONG', 'BOOL', 'WORD', 'BYTES', 'large', 'short', 'dd.1',
        'dw.1', 'ptr', 'DATA', 'FUNCTION', 'byte', 'word', 'char',
         'stdcall', 'arg', 'asc', 'align', 'WinMain', 'unk', 'cookie',
         'off', 'nullsub', 'DllEntryPoint', 'dll', 'HMENU', 'void',
        'HDC', 'HWND', 'entry', 'Software', '__imp_', 'case', 'assume', 'entropy'
]]
```

In [73]:

```
for column in result_x.iteritems():
    if len(set(column[1].isnull())) == 1 and list(set(column[1].isnull()))[0] == True:
        result_x.drop(labels=[column[0]],axis=1,inplace=True)
```

In [74]:

```
result_x.shape
```

Out[74]:

```
(10868, 510)
```

In [75]:

```
# from sklearn.feature_selection import SelectKBest, chi2
# bestfeatures = SelectKBest(score_func=chi2,k=100)
# result_x = bestfeatures.fit_transform(result_x,result_y)
# result_x = pd.DataFrame(result_x)
```

In [76]:

```
1 pp.ProfileReport(result_x)
```

Out[76]:

Overview

Dataset info

Number of variables	510
Number of observations	10868
Total Missing (%)	0.0%
Total size in memory	42.4 MiB
Average record size in memory	4.0 KiB

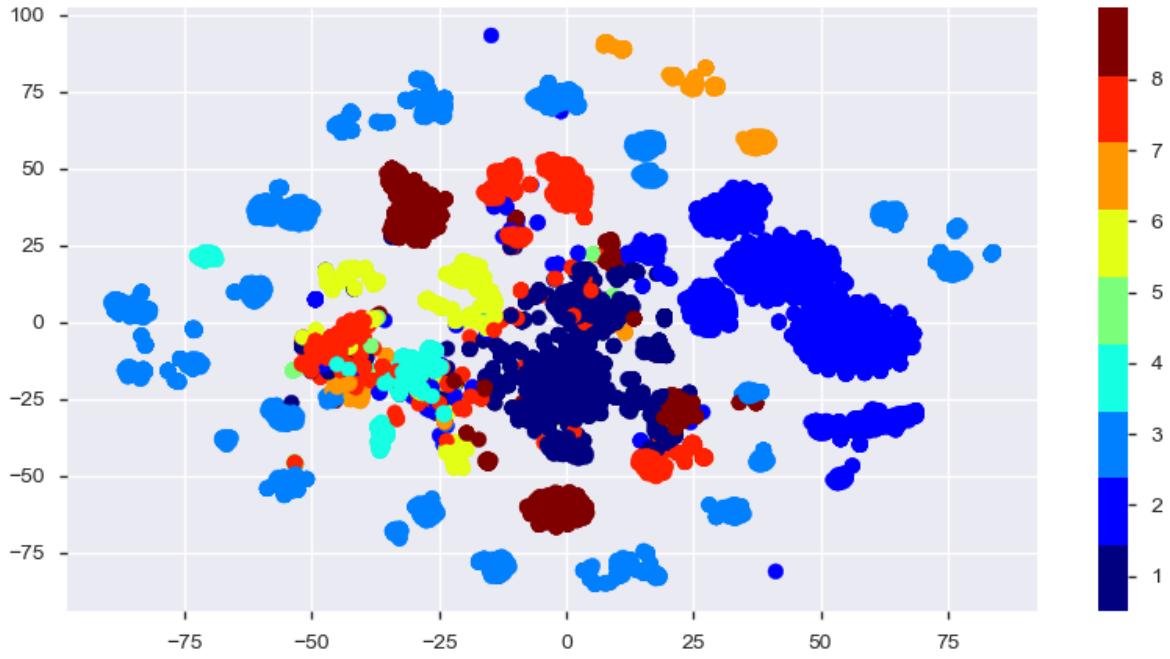
Variables types

Numeric	509
Categorical	0

4.5.2. Multivariate Analysis on final fearures

In [77]:

```
plt.figure(figsize = (10,5))
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```



4.5.3. Train and Test split

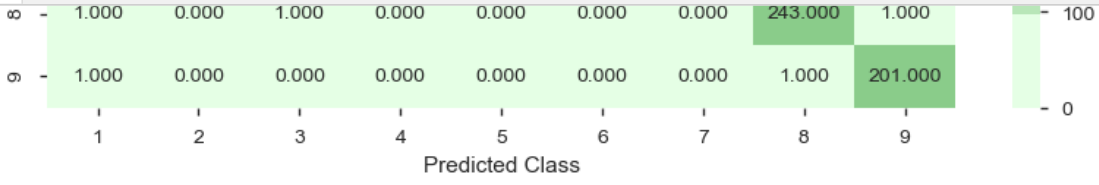
In [78]:

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,str
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_tra
```

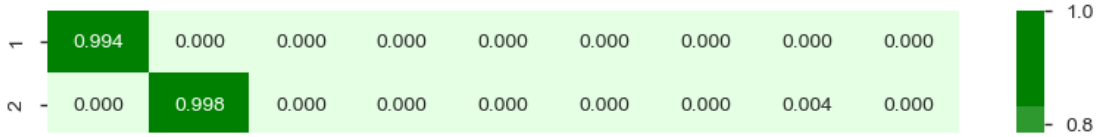
4.5.4. Random Forest Classifier on final features

In [79]:

```
if not os.path.exists('RF_one_param_tuning_merge.joblib'):
    alpha=[500,1000,5000,7000]
    cv_log_error_array=[]
    train_log_error_array=[]
    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)

    joblib.dump(sig_clf,'RF_one_param_tuning_merge.joblib')
else:
    sig_clf = joblib.load('RF_one_param_tuning_merge.joblib')

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



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



4.5.5. XgBoost Classifier on final features

In [80]:

```
if not os.path.exists('XGB_one_param_tuning_merge.joblib'):
    alpha=[500,1000,5000]
    cv_log_error_array=[]
    for i in alpha:
        x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
        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=alpha[best_alpha],nthread=-1)
    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)

    joblib.dump(sig_clf,'XGB_one_param_tuning_merge.joblib')
else:
    sig_clf = joblib.load('XGB_one_param_tuning_merge.joblib')

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

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

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

In [81]:

```
if not os.path.exists('XGB_multiple_param_tuning_merge.joblib'):
    x_cfl=XGBClassifier()

    prams={
         'learning_rate':[0.001,0.01,0.1],
         'n_estimators':[500,1000,2000,5000,7000],
         'max_depth':[3,5,7,9]
    }
    random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,n_iter=10,verbose=1
    random_cfl1.fit(X_train_merge,y_train_merge)
    joblib.dump(random_cfl1, 'XGB_multiple_param_tuning_merge.joblib')
else:
    random_cfl1 = joblib.load('XGB_multiple_param_tuning_merge.joblib')
```

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

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed: 31.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 106.1min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed: 176.5min remaining: 10
2.2min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 204.7min remaining: 6
2.3min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 209.1min remaining: 2
3.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 286.7min finished
```

In [82]:

```
random_cfl1.best_params_
```

Out[82]:

```
{'n_estimators': 7000, 'max_depth': 3, 'learning_rate': 0.1}
```

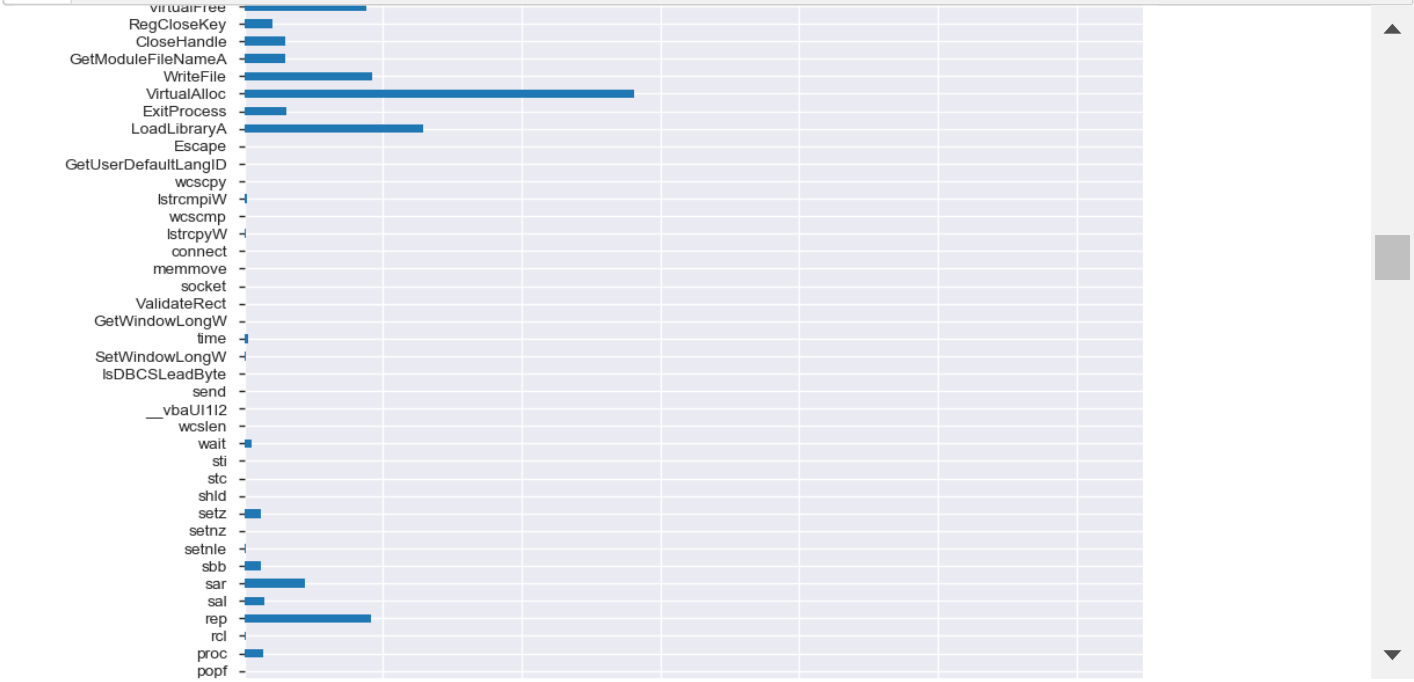
In [83]:

```
if not os.path.exists('XGB_multiple_param_tuning_Calibrated_merge.joblib'):
    c_cfl = CalibratedClassifierCV(random_cfl1.best_estimator_ ,method='sigmoid')
    c_cfl.fit(X_train_merge,y_train_merge)
    joblib.dump(c_cfl, 'XGB_multiple_param_tuning_Calibrated_merge.joblib')
else:
    c_cfl = joblib.load('XGB_multiple_param_tuning_Calibrated_merge.joblib')

predict_y = c_cfl.predict_proba(X_train_merge)
print ("The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = c_cfl.predict_proba(X_cv_merge)
print("The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = c_cfl.predict_proba(X_test_merge)
print("The test log loss is:",log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_merge, c_cfl.predict(X_test_merge),'XGB Model (with more
```

The feauture importances achieved by the XGBoost model plotted

In [84]:

```
feature_importance = pd.DataFrame(dict(zip(list(X_train_merge.columns.values),list(ran
feature_importance = feature_importance.transpose()
feature_importance.rename(columns={1:'score'},inplace=True)
feature_importance.plot.barh(figsize=(10,100));
```



5. MLP Softmax

1. Add bi-grams and n-gram features on byte files and improve the log-loss
2. Using the 'dchad' github account (https://github.com/dchad/malware-detection), decrease the logloss to <=0.01
3. Watch the video (https://www.youtube.com/watch?v=VLQTRILGz5Y) that was in reference section and implement the image features to improve the logloss

For this I read through the documentation provided and tried out a few different things.
(1) I fixed up all the code and reran everything from file processing to models (run time took a couple of days)
(2) I tried the n-gram features, but struggle with memory (so I omitted this approach)
(3) I used additional features from the document here: https://arxiv.org/pdf/1511.04317.pdf (https://arxiv.org/pdf/1511.04317.pdf) and added them in (['[','-','+','*','?','@',']']) during the processing of the asm files
(4) I used the feature importances from the XGBoost model and weighted the two most important features more (please see bar chart above)
(5) I performed hyperparamter tuning
(6) I built a MLP softmax running GPU's with adam optimizer to see if it could possibly reach a more global optimum
(7) I summarized the three top models in a pretty table

Conclusion:
I could not get the log loss below 0.01.
Since the heuristic is random I could only get the log loss between 0.01 - 0.04

In [85]:

```
# just something I tried for ngrams but I had memory issues:
if not os.path.exists('ngrambytesfile.csv'):
    files = os.listdir('byteFiles')
    bytefiledict = dict()
    counter = 0
    for file in files:
        filename = file.split('.')[0]
        filetext = ''
        with open(os.path.join('byteFiles',file),'r') as content:
            content = content.readlines()
        for line in content:
            filetext = filetext + ' '.join(line)
        if bytefiledict.get(filename) == None:
            bytefiledict[filename] = filetext.replace('\n','').strip()
            f = pd.DataFrame(bytefiledict,index=bytefiledict.keys()).reset_index()
            f.columns = ['ID','text']
        f.to_csv('ngrambytesfile.csv',index=False,header=False,mode='a')
        bytefiledict = dict()
        counter += 1
        if counter%2000 == 0:
            print(str(counter),' documents processed')
    print(str(counter),' documents processed')
    print('DONE!')
```

One hot encoding on the response variable

In [86]:

```
from sklearn.preprocessing import LabelBinarizer #MultilabelBinarizer
onehotencoding = LabelBinarizer()
y_train_onehot = onehotencoding.fit_transform(y_train_merge)
y_cv_onehot = onehotencoding.fit_transform(y_cv_merge)
y_test_onehot = onehotencoding.fit_transform(y_test_merge)
```

In [87]:

```
1 X_train_merge.head()
```

Out[87]:

	0	1	2	3	4	5	6	7	
8505	0.006351	0.012047	0.003451	0.003336	0.003887	0.003366	0.003716	0.005765	0.00531
4179	0.038119	0.001487	0.000328	0.000391	0.000386	0.000481	0.000273	0.000403	0.00048
10631	0.030318	0.026681	0.009145	0.008028	0.009156	0.008561	0.008355	0.013154	0.01458
5787	0.270462	0.005530	0.001695	0.001919	0.001777	0.001734	0.001890	0.002615	0.00308
9627	0.270160	0.005959	0.001927	0.001617	0.002290	0.002226	0.001608	0.002538	0.00315

5 rows × 510 columns

Multilayer Perceptron with softmax layer
L1 - 317 relu activation units
L2 - 300 relu activation units
L3 - 9 softmax activation units

Performed batch normalization and dropout in between the layers
batch size = 2000 and 150 epochs

In [89]:

```
import keras

# Create model:
model = keras.models.Sequential()

# Add layers:
model.add(keras.layers.Dense(510,activation='relu',kernel_initializer=keras.initialize
model.add(keras.layers.BatchNormalization()) #L2
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(300,activation='relu',kernel_initializer=keras.initialize
model.add(keras.layers.BatchNormalization()) #L4
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(9,activation='softmax')) #L5

model.summary()

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
history = model.fit(X_train_merge,y_train_onehot,batch_size=1000,epochs=150,verbose=1,
model_output = model.evaluate(X_test_merge,y_test_onehot)
```

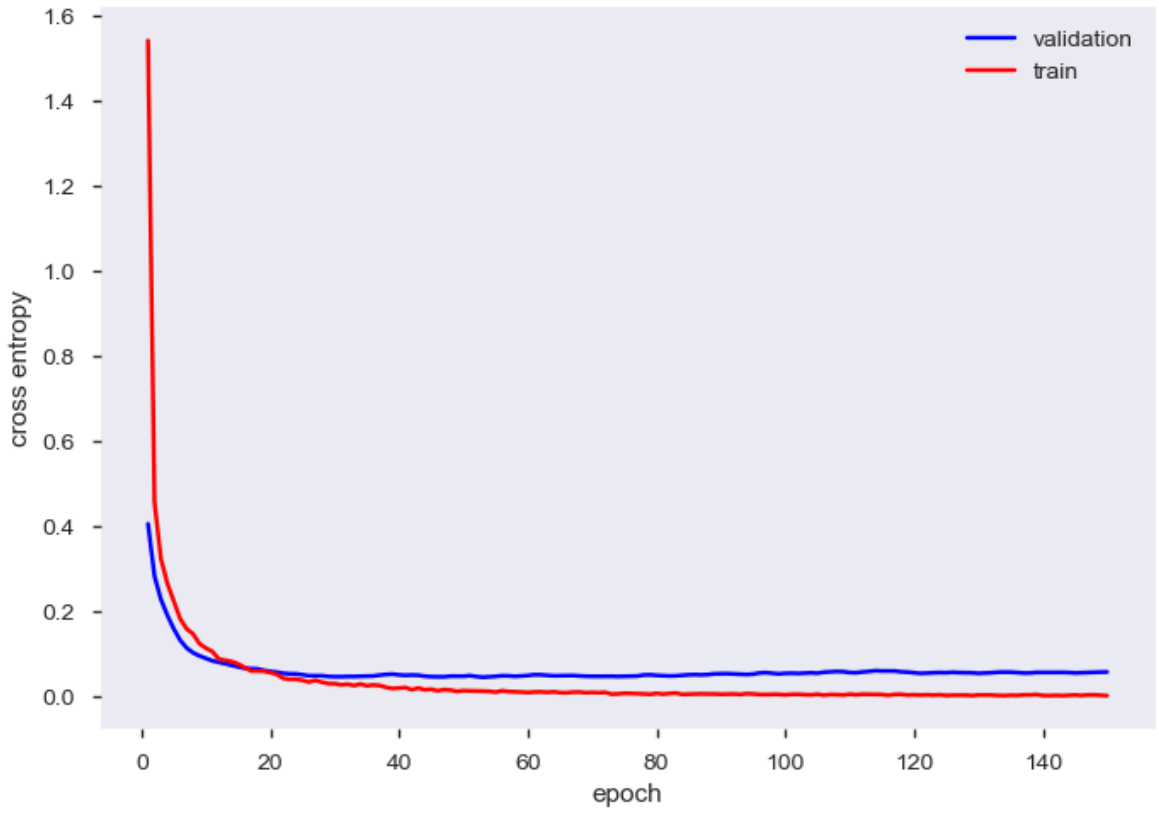
```
6955/6955 [==============================] - ETA: 0s - loss: 0.0016 - acc:
1.000 - ETA: 0s - loss: 0.0026 - acc: 0.999 - ETA: 0s - loss: 0.0034 - ac
c: 0.998 - ETA: 0s - loss: 0.0060 - acc: 0.998 - ETA: 0s - loss: 0.0050 -
acc: 0.998 - ETA: 0s - loss: 0.0046 - acc: 0.998 - 1s 162us/step - loss:
0.0048 - acc: 0.9986 - val_loss: 0.0585 - val_acc: 0.9908
Epoch 149/150
6955/6955 [==============================] - ETA: 0s - loss: 0.0050 - acc:
0.996 - ETA: 0s - loss: 0.0057 - acc: 0.996 - ETA: 0s - loss: 0.0046 - ac
c: 0.997 - ETA: 0s - loss: 0.0048 - acc: 0.997 - ETA: 0s - loss: 0.0042 -
acc: 0.997 - ETA: 0s - loss: 0.0041 - acc: 0.997 - 1s 146us/step - loss:
0.0037 - acc: 0.9980 - val_loss: 0.0584 - val_acc: 0.9902
Epoch 150/150
6955/6955 [==============================] - ETA: 0s - loss: 0.0028 - acc:
0.999 - ETA: 0s - loss: 0.0032 - acc: 0.999 - ETA: 0s - loss: 0.0038 - ac
c: 0.998 - ETA: 0s - loss: 0.0036 - acc: 0.998 - ETA: 0s - loss: 0.0036 -
acc: 0.998 - ETA: 0s - loss: 0.0032 - acc: 0.999 - 1s 151us/step - loss:
0.0029 - acc: 0.9991 - val_loss: 0.0588 - val_acc: 0.9908
2174/2174 [==============================] - ETA:  - ETA:  - ETA:  - ETA:
- ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 1s 239us/step
```

Model training results

In [90]:

```
print("Cross entropy: ",model_output[0])
print("Accuracy: ",model_output[1])
def plt_dynamic(x,vy,ty,ax,colors=['b']):
    ax.plot(x,vy,'b',label='validation')
    ax.plot(x,ty,'r',label='train')
    plt.legend()
    plt.grid()
    fig.canvas.draw()

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('cross entropy')
x=list(range(1,150+1))
vy=history.history['val_loss']
ty=history.history['loss']
plt_dynamic(x,vy,ty,ax)
```

```
Cross entropy:  0.04784118044415784
Accuracy:  0.9917203311867525
```



In [91]:

```
table = PrettyTable(['Model','Test Log Loss','Validation Log Loss',"Train Log Loss", '
```

In [92]:

```
table.add_row(['Random Forest','0.02','0.03','0.01','99.1%'])
table.add_row(['XGBoost','0.01','0.01','0.01','99.5%'])
table.add_row(['MLP (Softmax)','0.04','0.01','0.01','98.8%'])
```

In [93]:

```
print(table)
```

```
+---------------+--------------+--------------------+---------------+----
------+
|     Model     | Test Log Loss | Validation Log Loss | Train Log Loss | Acc
uracy |
+---------------+--------------+--------------------+---------------+----
------+
| Random Forest |      0.02     |         0.03        |      0.01      |  9
9.1%   |
|    XGBoost    |      0.01     |         0.01        |      0.01      |  9
9.5%   |
| MLP (Softmax) |      0.04     |         0.01        |      0.01      |  9
8.8%   |
+---------------+--------------+--------------------+---------------+----
------+
```