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

.bytes file

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- **Multi class log-loss**
- **Confusion matrix**

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

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

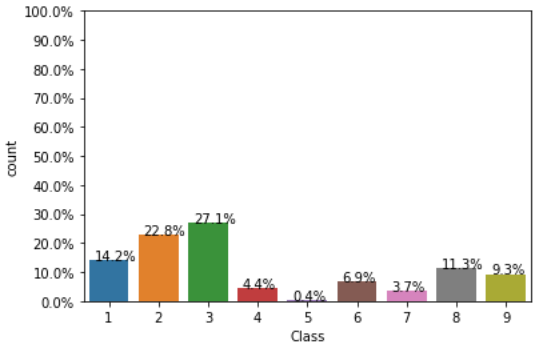
3. Exploratory Data Analysis

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

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

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

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

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

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

3.1. Distribution of malware classes in whole data set

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

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

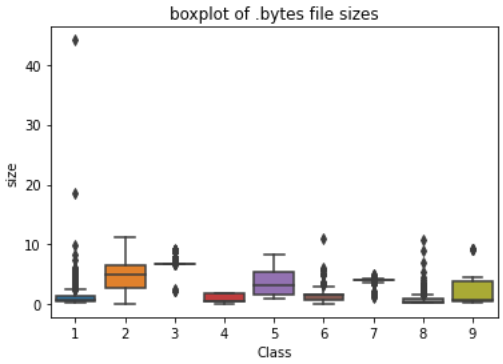
In [ ]:
```
files = os.listdir('byteFiles')
```

In [ ]:
```
#file sizes of byte files

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

```
                     ID      size  Class
0  Ee0OgkMFHZxyI5VwlcsC  0.609375      6
1  6xezkmBt9upOylqoX48E  1.113281      6
2  0NEsQlDGnUMg3Bew7R1A  0.363281      1
3  GKL3rD2YSf9xiXIPzeb6  3.820312      9
4  g8YBNnm4T162fPK7tqjL  6.714844      3
```

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

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



3.2.3 feature extraction from byte files

In [ ]:
```
from pathlib import Path
#https://stackoverflow.com/questions/10763298/how-can-i-read-a-binary-file-and-turn-the-data-into-an-image

# Define width and height
w, h = 50, 100

if not os.path.isdir('binary2_image'):
    os.makedirs('binary2_image')

if os.path.isdir('byteFiles'):
    data_files = os.listdir('byteFiles')
    for file in data_files:
        name=file.split('.')[0]

        my_file = Path('byteFiles/'+file)
        if my_file.is_file():
          # Read file using numpy "fromfile()"
          with open('byteFiles/'+file,'rb') as f:
            d = np.fromfile(f,dtype=np.uint8,count=w*h).reshape(h,w)

          # Make into PIL Image and save
          PILimage = Image.fromarray(d)

          PILimage = Image.fromarray(d)
          destiny = 'binary2_image/' +name + '.jpeg'
          PILimage.save(destiny)
          f.close()

if not os.path.isdir('asm_image'):
    os.makedirs('asm_image')

if os.path.isdir('asmFiles'):
    data_files = os.listdir('asmFiles')
    for file in data_files:
        name=file.split('.')[0]

        my_file = Path('asmFiles/'+file)
        if my_file.is_file():
          # Read file using numpy "fromfile()"
          with open('asmFiles/'+file,'rb') as f:
            d = np.fromfile(f,dtype=np.uint8,count=w*h).reshape(h,w)

          # Make into PIL Image and save
          PILimage = Image.fromarray(d)

          PILimage = Image.fromarray(d)
          destiny = 'asm_image/' +name + '.jpeg'
          PILimage.save(destiny)
          f.close()
```

In [ ]:
```
from PIL import Image
from pathlib import Path
binary_file_name = []
binary_arrays = []
if os.path.isdir('binary2_image'):
    data_files = os.listdir('binary2_image')
    for file in data_files:
        name=file.split('.')[0]
        binary_file_name.append(name)

        im = Image.open('binary2_image/'+file,'r')
        pix_val = list(im.getdata())
        binary_arrays.append(pix_val)

binary_arrays = np.array(binary_arrays)
column_name = ['pixel : ' + str(i) for i in range(len(binary_arrays[0]))]

dic = {column_name[i]:binary_arrays[:,i] for i in range(len(binary_arrays[0]))}
```

```
dic.update({'ID':binary_file_name})

binary_file = pd.DataFrame(dic)
binary_file.to_csv('binary_file.csv')
```

In [ ]:
```
binary_file = pd.read_csv('binary_file.csv')
```

In [ ]:
```
binary_file.head(5)
```

Out[ ]:

	Unnamed: 0	pixel : 0	pixel : 1	pixel : 2	pixel : 3	pixel : 4	pixel : 5	pixel : 6	pixel : 7	pixel : 8	pixel : 9	pixel : 10	pixel : 11	pixel : 12	pixel : 13	pixel : 14	pixel : 15	pixel : 16	pixel : 17	pixel : 18	pixel : 19	pixel : 20	pixel : 21	pi :
0	0	62	56	42	71	69	35	46	56	33	57	55	52	60	52	45	54	50	37	56	68	49	50	
1	1	51	65	22	52	69	37	60	61	31	58	68	28	47	66	35	65	57	47	58	41	29	64	
2	2	57	49	31	56	55	30	54	70	40	56	69	47	58	70	40	47	56	40	63	59	35	55	
3	3	69	55	31	64	71	34	45	68	30	44	43	47	65	59	42	46	76	26	51	51	23	44	
4	4	54	36	39	69	45	34	70	56	33	46	41	38	47	43	41	60	65	22	64	80	36	53	

5 rows × 5002 columns

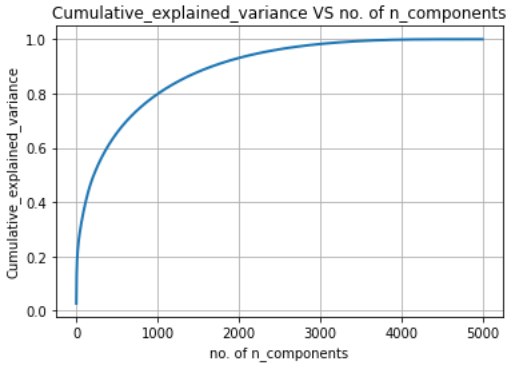
In [ ]:
```
binary_file.shape
```

Out[ ]:
```
(10868, 5001)
```

In [ ]:
```
binary_file = binary_file.drop(columns=['ID','Unnamed: 0'])
```

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
#Desired dimensionality of output data. Must be strictly less than the number of features
max_features = 5000 - 1
svd = TruncatedSVD(n_components=max_features)
svd_data = svd.fit_transform(binary_file)
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)
```

In [ ]:
```
%matplotlib inline
# Plot the TrunvatedSVD spectrum
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.grid()
plt.xlabel('no. of n_components')
plt.ylabel('Cumulative_explained_variance')
plt.title("Cumulative_explained_variance VS no. of n_components")
plt.show()
```



In [ ]:
```
Optimal_components=3500

svd_trunc = TruncatedSVD(n_components=Optimal_components)
svd_transform = svd_trunc.fit_transform(binary_file)

print("Shape of data : ",svd_transform.shape)
```

```
Shape of data :  (10868, 3500)
```

In [ ]:
```
type(svd_transform)
```

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
from PIL import Image
from pathlib import Path
binary_file_name = []
binary_arrays = []
if os.path.isdir('binary2_image'):
    data_files = os.listdir('binary2_image')
```

```
    for file in data_files:
        name=file.split('.')[0]
        binary_file_name.append(name)
```

In [ ]:
```
column_name = ['pixel : ' + str(i) for i in range(5000)]
```

In [ ]:
```
best_fearures = [column_name[i] for i in svd_trunc.components_[0].argsort()[::-1][0:3500]]
#best_fearures.append('ID')
#binary_data = pd.read_csv('binary_file.csv',names=best_fearures)
#binary_data = pd.read_csv('binary_file.csv',names=best_fearures)
```

In [ ]:
```
#binary_arrays = np.array(binary_arrays)
#column_name = ['pixel : ' + str(i) for i in range(len(binary_arrays[0]))]

dic = {best_fearures[i]:svd_transform[:,i] for i in range(3500)}
dic.update({'ID':binary_file_name})

binary_data = pd.DataFrame(dic)
#binary_file.to_csv('binary_file.csv')
```

In [ ]:
```
binary_data.head(5)
```

Out[ ]:

	pixel : 3459	pixel : 3075	pixel : 2259	pixel : 1059	pixel : 1875	pixel : 4659	pixel : 675	pixel : 2691	pixel : 3891	pixel : 4275	pixel : 291	pixel : 1491	pixel : 2
0	3462.344837	-47.646559	21.519436	-24.367828	3.938196	3.985415	-11.668452	-30.991851	1.193291	-15.801324	6.190879	-18.765335	-40.6991
1	3621.684600	-23.887582	-78.396846	68.302360	23.937345	-0.498847	-39.899394	21.715165	-4.045202	-25.957239	-0.717803	-1.212317	8.4995
2	3544.195427	-11.878047	-27.568569	3.327325	9.503637	-13.209312	-19.694874	-3.654795	-30.064650	28.288543	-7.148923	-2.531504	-19.4672
3	3584.104290	568.660672	61.307523	1.236295	-3.501591	1.359530	47.152679	12.799232	14.774650	-19.569575	19.587804	-3.320317	-17.2512
4	3450.769118	-26.434190	26.668526	-21.311390	-6.603748	1.627494	-12.477852	-44.563811	-12.688335	-1.810538	-8.971798	-23.562621	9.2183

5 rows × 3501 columns

In [ ]:
```
byte_features_with_size_binary = binary_data.merge(data_size_byte, on='ID')
byte_features_with_size_binary.to_csv("result_with_size_binary.csv")
byte_features_with_size_binary.head(2)
```

Out[ ]:

	pixel : 3459	pixel : 3075	pixel : 2259	pixel : 1059	pixel : 1875	pixel : 4659	pixel : 675	pixel : 2691	pixel : 3891	pixel : 4275	pixel : 291	pixel : 1491	pixel : 267
0	3462.344837	-47.646559	21.519436	-24.367828	3.938196	3.985415	-11.668452	-30.991851	1.193291	-15.801324	6.190879	-18.765335	-40.699164
1	3621.684600	-23.887582	-78.396846	68.302360	23.937345	-0.498847	-39.899394	21.715165	-4.045202	-25.957239	-0.717803	-1.212317	8.499539

2 rows × 3503 columns

In [ ]:
```
def n_gram_table(file_name_in_csv_format,n_gram_no,from_which_file):
  line = "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
  from itertools import permutations
  def n_gram(string,number):
    line = string.split(',')
    line1 = []
    for i in line:
      line1.append(str(i))
    line1 = list(set(line1))

    N_gram = []

    for i in range(1,number+1):
        perm = permutations(line1, i)

        for i in list(perm):
          N_gram.append((" ".join(list(i))))

    return N_gram

  column_name = n_gram(line,n_gram_no)

  from sklearn.feature_extraction.text import CountVectorizer
  from tqdm import tqdm
  vec  = CountVectorizer(vocabulary = column_name,ngram_range=(1, 2))

  files = os.listdir(from_which_file)
  filenames2=[]
  feature_matrix = np.zeros((len(files),len(column_name)),dtype=int)
  k=0


  #program to convert into bag of words of bytefiles
  #this is custom-built bag of words this is unigram bag of words
  byte_feature_file=open(file_name_in_csv_format,'w+')
  #byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27
```

```
  byte_feature_file.write("ID"+",")
  for i in vec.get_feature_names():
    byte_feature_file.write(str(i)+",")
  byte_feature_file.write("\n")  # we had to add +"," to above line and also byte_feature_file.write("\n") remaining all same.

  for file in tqdm(files[0:10000]):
    id_name=file.split('.')[0]
    filenames2.append(file)
    byte_feature_file.write(id_name+",")
    if(file.endswith("txt")):
      dir_name = from_which_file+'/'
      f =  open(dir_name+file,"r")
      c = f.read().lower()
      d = [c]
      countvector =  vec.fit_transform(d)
      b = countvector.toarray()
      feature_matrix[k] = b[0]
    for i in feature_matrix[k]:
      byte_feature_file.write(str(i)+",")
    byte_feature_file.write("\n")
    k += 1

  byte_feature_file.close()

n_gram_table('result.csv',2,'byteFiles')
```

In [ ]:
```
byte_features_with_size_binary = pd.read_csv("result_with_size_binary.csv")
byte_features_with_size_binary =  byte_features_with_size_binary.drop(columns = ['Unnamed: 0'])
byte_features_with_size_binary.head(5)
```

Out[ ]:

	pixel : 3459	pixel : 3075	pixel : 2259	pixel : 1059	pixel : 1875	pixel : 4659	pixel : 675	pixel : 2691	pixel : 3891	pixel : 4275	pixel : 291	pixel : 1491	pixel : 2
0	3462.344837	-47.646559	21.519436	-24.367828	3.938196	3.985415	-11.668452	-30.991851	1.193291	-15.801324	6.190879	-18.765335	-40.6991
1	3621.684600	-23.887582	-78.396846	68.302360	23.937345	-0.498847	-39.899394	21.715165	-4.045202	-25.957239	-0.717803	-1.212317	8.4995
2	3544.195427	-11.878047	-27.568569	3.327325	9.503637	-13.209312	-19.694874	-3.654795	-30.064650	28.288543	-7.148923	-2.531504	-19.4672
3	3584.104290	568.660672	61.307523	1.236295	-3.501591	1.359530	47.152679	12.799232	14.774650	-19.569575	19.587804	-3.320317	-17.2512
4	3450.769118	-26.434190	26.668526	-21.311390	-6.603748	1.627494	-12.477852	-44.563811	-12.688335	-1.810538	-8.971798	-23.562621	9.2183

5 rows × 3503 columns

In [ ]:
```
#https://stackoverflow.com/questions/22258491/read-a-small-random-sample-from-a-big-csv-file-into-a-python-data-frame
import random

filename = "result.csv"
n = 10  # every 10th line = 1% of the lines
df = pd.read_csv(filename, header=0, skiprows=lambda i: i % n != 0)
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(1000, 66051)
```

In [ ]:
```
column_name = [col for col in df.columns]
```

In [ ]:
```
df.head(5)
```

Out[ ]:

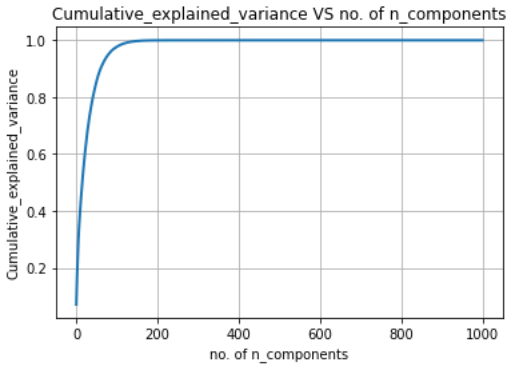
	ID	6	84	ba	d2	1e	d4	25	5	4f	e0	b9	63	b8	bc	2f	29	16	8f	6d	c0	9
0	fn2sqVuRcbvEH90ZK47x	0	860	369	632	411	468	534	0	436	1117	390	588	854	497	381	447	410	390	1684	1852	34
1	EJexdbnNPfwsgUp4GCQD	0	3249	3204	3243	3095	3191	3190	0	3260	3201	3275	3301	3241	3189	3319	3126	3201	3309	3220	3263	323
2	F6cqosn3GRDidyO0fYB5	0	8019	66471	67447	465	394	298	0	708	595	407	704	467	312	406	275	337	672	696	8615	70
3	2wIHgZMUrG4JBNf7ovxQ	0	3319	3227	3187	3106	3216	3226	0	3285	3248	3222	3407	3215	3240	3300	3190	3327	3239	3274	3243	319
4	9kjAqe6TgFsQZGRXcp2t	0	8304	442	467	176	389	154	0	391	482	576	525	683	392	255	130	287	632	733	11285	64

5 rows × 66051 columns

In [ ]:
```
df = df.drop(columns=['ID'])
df = df.drop(columns=['Unnamed: 66050'])
```

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
#Desired dimensionality of output data. Must be strictly less than the number of features
max_features = 5000
svd = TruncatedSVD(n_components=max_features)
svd_data = svd.fit_transform(df)
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)
```

In [ ]:
```
#for 5000
%matplotlib inline
# Plot the TrunvatedSVD spectrum
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.grid()
plt.xlabel('no. of n_components')
plt.ylabel('Cumulative_explained_variance')
plt.title("Cumulative_explained_variance VS no. of n_components")
plt.show()
```



In [ ]:
```
Optimal_components=300

svd_trunc = TruncatedSVD(n_components=Optimal_components)
svd_transform = svd_trunc.fit_transform(df)

print("Shape of data : ",svd_transform.shape)
```

```
Shape of data :  (1000, 300)
```

In [ ]:
```
best_fearures = [column_name[i] for i in svd_trunc.components_[0].argsort()[::-1][0:300]]
best_fearures.append('ID')
print(best_fearures)
```

```
['fd', 'b4', '6e', '78', 'de', '1a', '79', '0d', '72', 'a1', 'bd', 'cf', '95', 'b1', '2e', 'f5', 'f6', 'e7', '44', '7b', '55', 'e9', '9
a', '91', 'e2', '89', '5b', '49', '7f', 'ea', 'c3', '8b', '6f', 'ec', '61', '75', 'd7', 'aa', 'ce', 'e6', '34', '2f', 'b6', '8f', 'c0',
'35', 'f7', 'a7', '51', 'e8', '32', 'dc', 'f8', '92', 'd2', 'f4', 'fa', '17', '96', '1c', 'c2', '6', 'be', '33', 'ab', 'd9', 'fb', '14',
'c5', 'fe', '8e', 'c6', '8a', '85', 'ca', 'a6', '2d', 'ff', 'f0', '3a', '30', '93', 'b0', '3c', '3b', '80', '57', '18', '81', 'd4', 'f2',
'7e', 'db', '8c', '9f', '15', '16', 'ed', '22', '70', 'af', 'd8', 'c4', '94', 'd3', '8', 'a9', 'e3', '4', '82', '98', '4b', '74', '4a',
'b3', '7', '52', '19', '40', 'e0', '2b', 'fc', '99', '39', '53', 'c7', '87', 'b8', '7c', '77', '5a', 'b7', '88', '1b', '3f', '5c', 'bb',
'a5', '6d', 'dd', '10', '6a', '3', 'e4', '4e', 'd1', '3d', '59', 'e5', 'd0', '84', 'bc', 'cd', '62', '4c', '64', '0b', '1', 'b9', '??',
'd5', '7a', '0a', '25', 'ae', '41', '0', '6b', 'b2', '50', 'ef', '4d', '9b', 'ee', '69', '5d', 'd6', '47', '9d', 'b5', 'e1', '28', 'a8',
'29', '12', 'ba', '83', '60', 'f3', '24', '2c', 'ad', '65', '37', '2a', '86', '71', '9', '2', '4f', '6c', '7d', '46', '68', 'a3', '20',
'90', '1e', '8d', '66', '13', '67', '21', 'f1', '1d', '38', 'c9', '48', '31', '45', '9c', '56', '0c', 'f9', '63', 'df', '5', '0f', '26',
'ac', 'a0', '42', '23', '97', '11', '43', 'eb', '1f', 'c1', '36', '58', 'a2', 'cb', 'cc', '76', '5e', '6 df', '6 bb', '73', '6 4e', '6 d
b', '6 e9', '6 2f', '6 16', '6 6d', '6 ae', '9e', 'c8', '6 f2', '6 26', 'da', '6 8f', '6 fc', '6 8a', '6 6a', '6 9d', '27', '6 62', '6 4
0', '6 8e', '6 5', '6 97', '6 ec', '6 fb', '6 84', '54', '0e', 'bf', '6 89', '6 24', '6 1e', '6 b9', '6 29', '6 37', '6 c0', '6 63', '6 b
8', '6 a9', '6 4f', '6 9b', '6 25', '6 47', '6 d2', '6 e0', '3e', '5f', '6 d4', '6 bc', '6 ba', 'a4', 'ID']
```

In [ ]:
```
import time
start = time.time()
#read data in chunks of 1 million rows at a time
chunk = pd.read_csv('result.csv',usecols =best_fearures,chunksize=1000)
end = time.time()
print("Read csv with chunks: ",(end-start),"sec")
pd_df = pd.concat(chunk)
```

```
Read csv with chunks:  0.16666746139526367 sec
```

In [ ]:
```
pd_df.head(3)
```

Out[ ]:

	ID	6	84	ba	d2	1e	d4	25	5	4f	e0	b9	63	b8	bc	2f	29	16	8f	6d	c0	97
0	Ee0OgkMFHZxyI5VwIcsC	0	475	376	535	562	363	503	0	428	431	345	641	462	398	445	379	431	399	695	440	419
1	6xezkmBt9upOylqoX48E	0	1085	1013	1051	1134	1044	1176	0	1145	1099	1095	1161	1096	1090	1174	1128	1153	1144	1112	1187	1077
2	0NEsQlDGnUMg3Bew7R1A	0	349	297	451	310	322	379	0	341	365	334	422	384	280	275	369	271	301	331	540	297

3 rows × 301 columns

In [ ]:
```
file_name = []
for i , row in pd_df.iterrows():
  file_name.append(row['ID'])
```

In [ ]:
```
Optimal_components=300-1

svd_trunc = TruncatedSVD(n_components=Optimal_components)
svd_transform = svd_trunc.fit_transform(pd_df)

print("Shape of data : ",svd_transform.shape)
```

```
Shape of data :  (10000, 299)
```

In [ ]:
```
#binary_arrays = np.array(binary_arrays)
#column_name = ['pixel : ' + str(i) for i in range(len(binary_arrays[0]))]

dic = {best_fearures[i]:svd_transform[:,i] for i in range(299)}
dic.update({'ID':file_name})

binary_data = pd.DataFrame(dic)
#binary_file.to_csv('binary_file.csv')
```

In [ ]:
```
binary_data.head(5)
```

Out[ ]:

	fd	b4	6e	78	de	1a	79	0d	72	a1	
0	10880.180408	-3539.011331	-5832.709760	-1426.196109	-610.423116	632.071530	-110.159533	-946.093646	-225.576824	1158.049334	-919.7090
1	13712.224923	-4765.508970	-6261.112307	-1445.685677	-510.042412	1302.994928	-193.431623	-965.512687	-679.391146	156.127924	-772.7308
2	5822.427880	-1880.234529	-2980.255683	-637.534660	28.452545	205.241541	6.406612	-407.528024	-322.448954	321.904194	-270.4566
3	64623.062301	-23675.803164	-33993.291843	-9754.925639	2286.383663	4615.689102	-675.700318	-7683.173321	280.790414	1675.025164	-6272.8876
4	40082.387684	-14166.258905	-18878.757526	-4027.808321	-1670.673772	3601.234173	-626.198450	-2713.250390	-2123.813666	69.917703	-2150.0892

5 rows × 300 columns

In [ ]:
```
byte_features_with_size_binary_pixel = binary_data.merge(byte_features_with_size_binary, on='ID')
byte_features_with_size_binary_pixel.to_csv("result_with_size_binary_pixel.csv")
byte_features_with_size_binary_pixel.head(2)
```

Out[ ]:

	fd	b4	6e	78	de	1a	79	0d	72	a1	bd	
0	10880.180408	-3539.011331	-5832.709760	-1426.196109	-610.423116	632.071530	-110.159533	-946.093646	-225.576824	1158.049334	-919.709047	6
1	13712.224923	-4765.508970	-6261.112307	-1445.685677	-510.042412	1302.994928	-193.431623	-965.512687	-679.391146	156.127924	-772.730806	-7(

2 rows × 3802 columns

In [ ]:
```
byte_features_with_size_binary_pixel = pd.read_csv('result_with_size_binary_pixel.csv')

byte_features_with_size_binary_pixel = byte_features_with_size_binary_pixel.drop(columns = ['Unnamed: 0'])

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

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

In [ ]:
```
byte_features_with_size_binary_pixel.head(5)
```

Out[ ]:

	fd	b4	6e	78	de	1a	79	0d	72	a1	
0	10880.180408	-3539.011331	-5832.709760	-1426.196109	-610.423116	632.071530	-110.159533	-946.093646	-225.576824	1158.049334	-919.7090
1	13712.224923	-4765.508970	-6261.112307	-1445.685677	-510.042412	1302.994928	-193.431623	-965.512687	-679.391146	156.127924	-772.7308
2	5822.427880	-1880.234529	-2980.255683	-637.534660	28.452545	205.241541	6.406612	-407.528024	-322.448954	321.904194	-270.4566
3	64623.062301	-23675.803164	-33993.291843	-9754.925639	2286.383663	4615.689102	-675.700318	-7683.173321	280.790414	1675.025164	-6272.8876
4	40082.387684	-14166.258905	-18878.757526	-4027.808321	-1670.673772	3601.234173	-626.198450	-2713.250390	-2123.813666	69.917703	-2150.0892

5 rows × 3802 columns

In [ ]:
```
byte_features_with_size_binary_pixel = byte_features_with_size_binary_pixel.drop(columns = ['Unnamed: 0'])
```

In [ ]:
```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
```

```
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
result = normalize(byte_features_with_size_binary_pixel)
```

In [ ]:
```
result.head(2)
```

Out[ ]:

	fd	b4	6e	78	de	1a	79	0d	72	a1	bd	cf	95	b1	2e
0	0.020686	0.241691	0.245214	0.07461	0.193234	0.401354	0.258731	0.246999	0.213273	0.437874	0.238340	0.216853	0.286667	0.328151	0.467180
1	0.026070	0.240115	0.244722	0.07460	0.193291	0.401774	0.258677	0.246983	0.212736	0.436829	0.238462	0.216140	0.285696	0.328129	0.467268

2 rows × 3802 columns

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

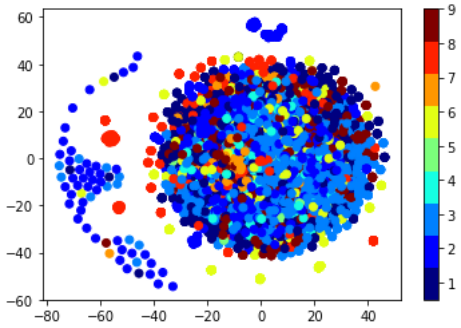
Out[ ]:

	fd	b4	6e	78	de	1a	79	0d	72	a1	bd	cf	95	b1	2e
0	0.020686	0.241691	0.245214	0.074610	0.193234	0.401354	0.258731	0.246999	0.213273	0.437874	0.238340	0.216853	0.286667	0.328151	0.467180
1	0.026070	0.240115	0.244722	0.074600	0.193291	0.401774	0.258677	0.246983	0.212736	0.436829	0.238462	0.216140	0.285696	0.328129	0.467268
2	0.011070	0.243824	0.248488	0.075041	0.193600	0.401087	0.258808	0.247447	0.213158	0.437002	0.238877	0.216545	0.286434	0.327896	0.467292
3	0.122864	0.215803	0.212897	0.070061	0.194897	0.403846	0.258361	0.241398	0.213872	0.438413	0.233919	0.214436	0.285582	0.331177	0.468386
4	0.076206	0.228028	0.230242	0.073189	0.192625	0.403211	0.258394	0.245530	0.211028	0.436740	0.237324	0.215298	0.284060	0.328597	0.466846

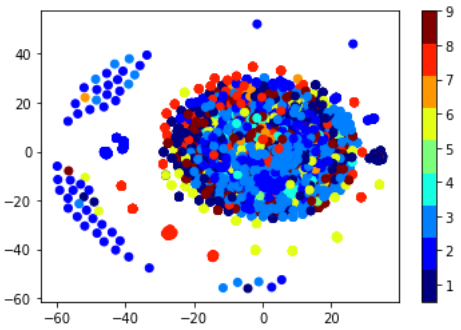
5 rows × 3802 columns

3.2.4 Multivariate Analysis

In [ ]:
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], 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 [ ]:
```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



Train Test split

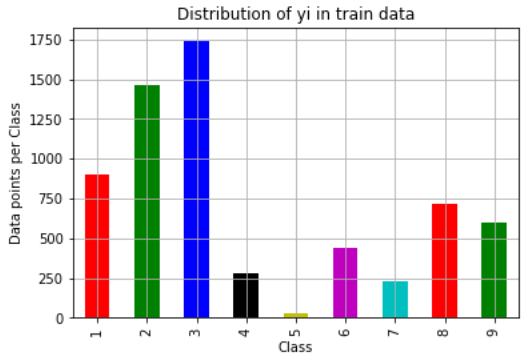
In [ ]:
```
byte_features_with_size_binary_pixel = pd.read_csv('result_with_size_binary_pixel.csv')

byte_features_with_size_binary_pixel = byte_features_with_size_binary_pixel.drop(columns = ['Unnamed: 0'])

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

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

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

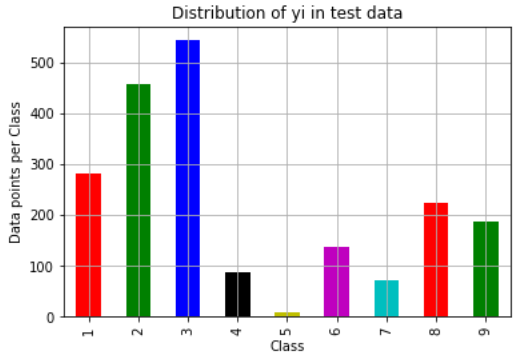
```
Number of data points in train data: 6400
Number of data points in test data: 2000
Number of data points in cross validation data: 1600
```

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

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

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


print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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

print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
```

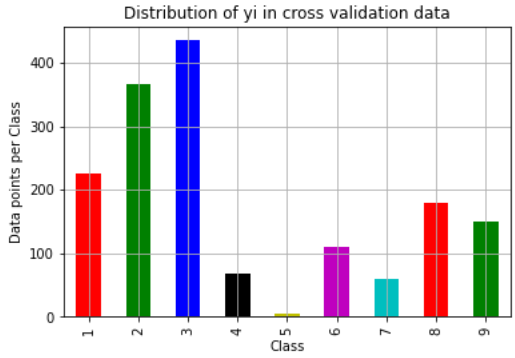
```
plt.show()

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




```
Number of data points in class 3 : 1738 ( 27.156 %)
Number of data points in class 2 : 1466 ( 22.906 %)
Number of data points in class 1 : 901 ( 14.078 %)
Number of data points in class 8 : 719 ( 11.234 %)
Number of data points in class 9 : 597 ( 9.328 %)
Number of data points in class 6 : 442 ( 6.906 %)
Number of data points in class 4 : 278 ( 4.344 %)
Number of data points in class 7 : 234 ( 3.656 %)
Number of data points in class 5 : 25 ( 0.391 %)
------------------------------------------------------------------------------
```




```
Number of data points in class 3 : 543 ( 27.15 %)
Number of data points in class 2 : 458 ( 22.9 %)
Number of data points in class 1 : 281 ( 14.05 %)
Number of data points in class 8 : 225 ( 11.25 %)
Number of data points in class 9 : 187 ( 9.35 %)
Number of data points in class 6 : 138 ( 6.9 %)
Number of data points in class 4 : 87 ( 4.35 %)
Number of data points in class 7 : 73 ( 3.65 %)
Number of data points in class 5 : 8 ( 0.4 %)
------------------------------------------------------------------------------
```




```
Number of data points in class 3 : 435 ( 27.187 %)
Number of data points in class 2 : 366 ( 22.875 %)
Number of data points in class 1 : 225 ( 14.062 %)
Number of data points in class 8 : 180 ( 11.25 %)
Number of data points in class 9 : 149 ( 9.312 %)
Number of data points in class 6 : 111 ( 6.938 %)
Number of data points in class 4 : 69 ( 4.312 %)
Number of data points in class 7 : 59 ( 3.688 %)
Number of data points in class 5 : 6 ( 0.375 %)
```

In [ ]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
```

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

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

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

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

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

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

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

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

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

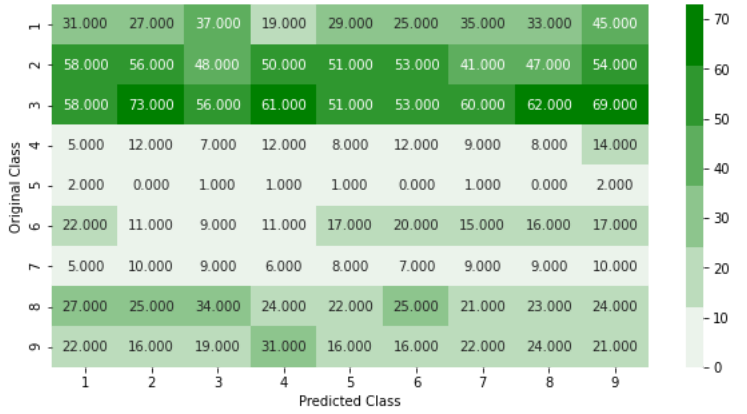
4.1.1. Random Model

In [ ]:

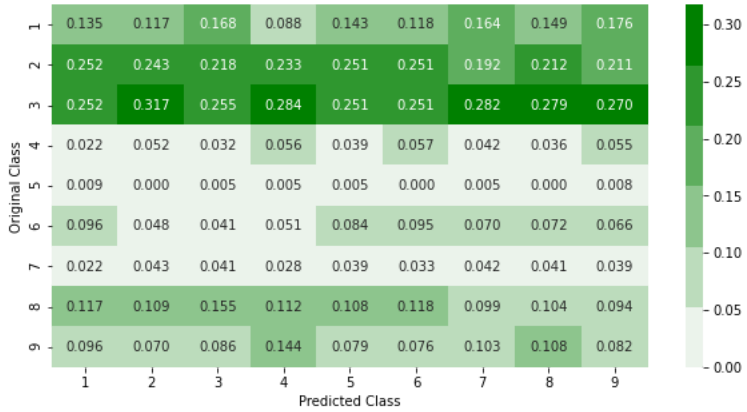
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.4714313387591496
Log loss on Test Data using Random Model 2.471818291241031
```

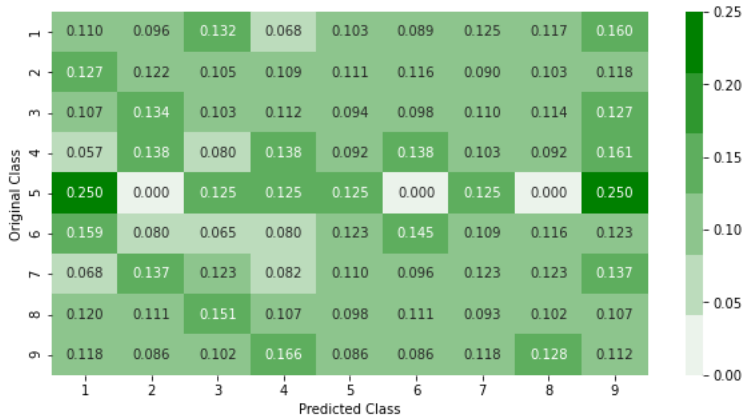
```
Number of misclassified points  88.55
---------- Confusion matrix ----------
```




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




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




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

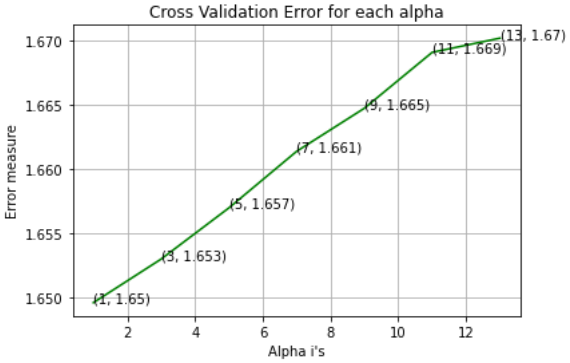
4.1.2. K Nearest Neighbour Classification

In [ ]:

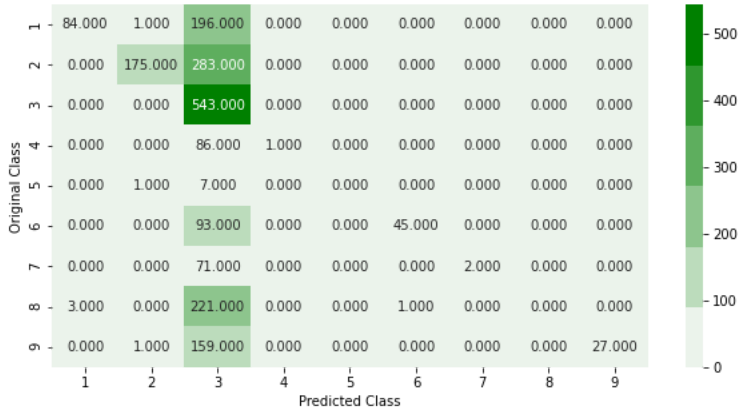
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
```

```
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

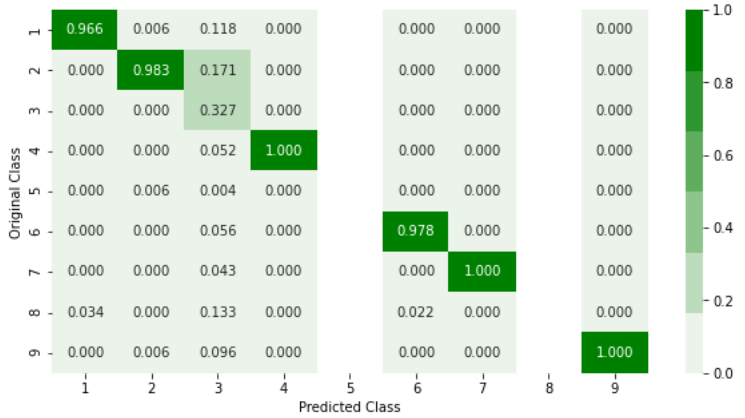
```
log_loss for k =  1 is 1.6496422817358967
log_loss for k =  3 is 1.6530242406892586
log_loss for k =  5 is 1.6569957622194078
log_loss for k =  7 is 1.6613873234106622
log_loss for k =  9 is 1.6647332078843369
log_loss for k =  11 is 1.6690691818835175
log_loss for k =  13 is 1.6701727644239441
```




```
For values of best alpha =  1 The train log loss is: 0.7789582785320465
For values of best alpha =  1 The cross validation log loss is: 1.6496422817358967
For values of best alpha =  1 The test log loss is: 1.6526347509472208
Number of misclassified points  56.15
---------------------------------------------------- Confusion matrix -----------------------------------------------------
```

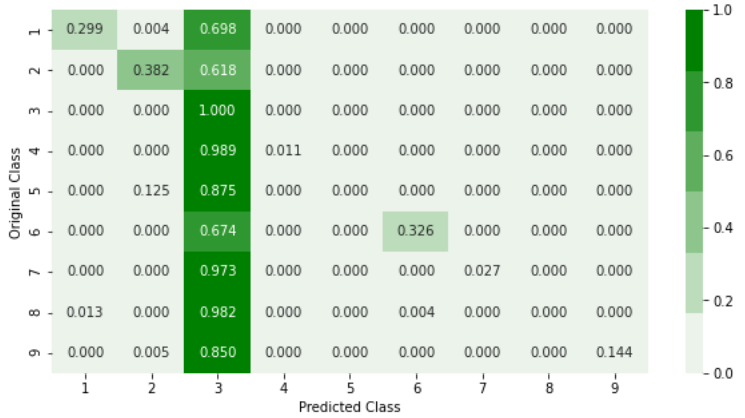


---------- Precision matrix ----------



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

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



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

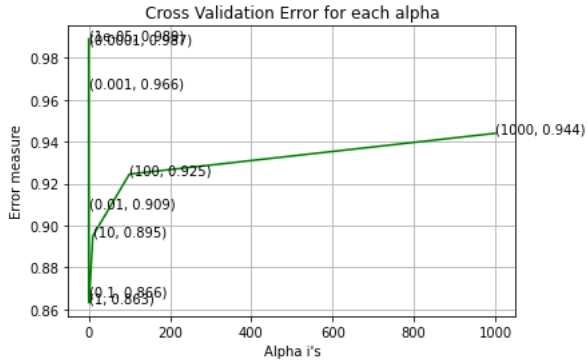
4.1.3. Logistic Regression

In [ ]:

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

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

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

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

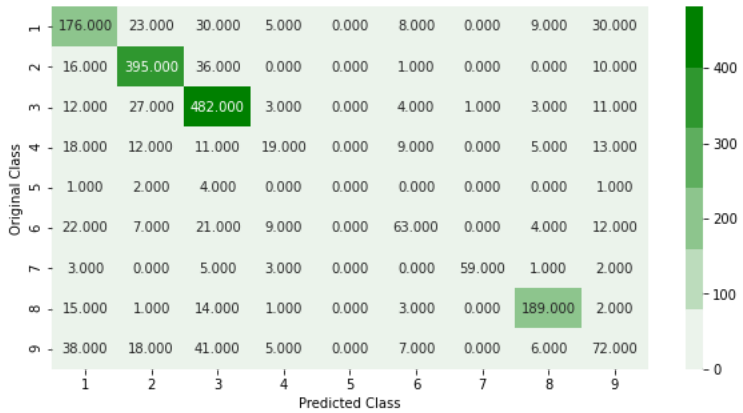
```
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  1e-05 is 0.989047687779439
log_loss for c =  0.0001 is 0.986846302204002
log_loss for c =  0.001 is 0.9655608634815152
log_loss for c =  0.01 is 0.9085205909775851
log_loss for c =  0.1 is 0.86640377360578
log_loss for c =  1 is 0.8631026082457715
log_loss for c =  10 is 0.8949906737037171
log_loss for c =  100 is 0.924536862630352
log_loss for c =  1000 is 0.9440231530670041
```

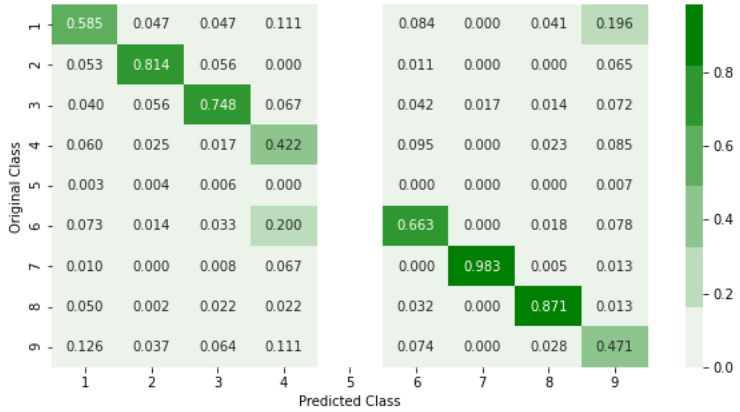



```
log loss for train data 0.2763699732810322
log loss for cv data 0.8631026082457715
log loss for test data 0.935104105498665
Number of misclassified points  27.250000000000004
```

---------- Confusion matrix ----------

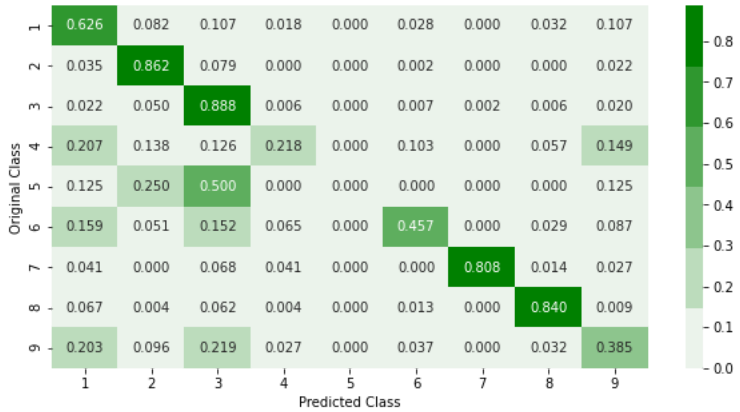


---------- Precision matrix ----------




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

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




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

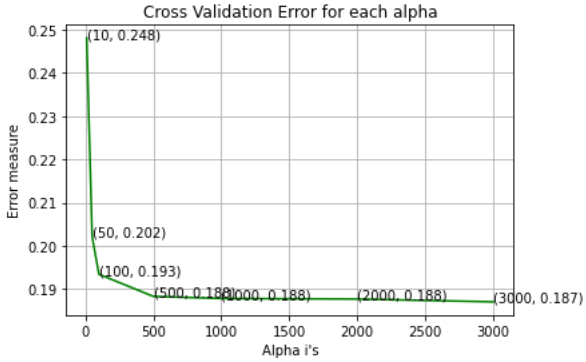
4.1.4. Random Forest Classifier

In [ ]:

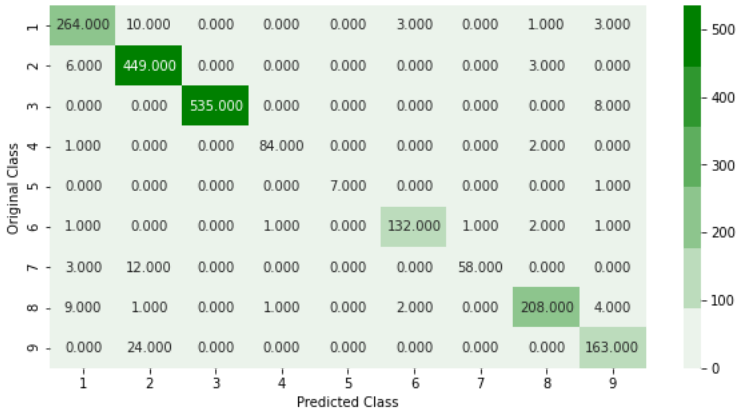
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

# -------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -------------------------------

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

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


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

```
plt.ylabel("Error measure")
plt.show()


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

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

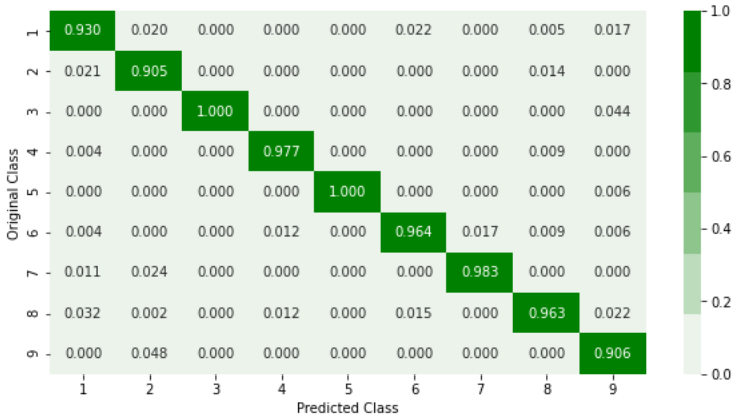
```
log_loss for c =  10 is 0.2480541852416666
log_loss for c =  50 is 0.20213568394293793
log_loss for c =  100 is 0.19343793920014185
log_loss for c =  500 is 0.18832516359302773
log_loss for c =  1000 is 0.1878361640074045
log_loss for c =  2000 is 0.18771180247110095
log_loss for c =  3000 is 0.18706701008453522
```



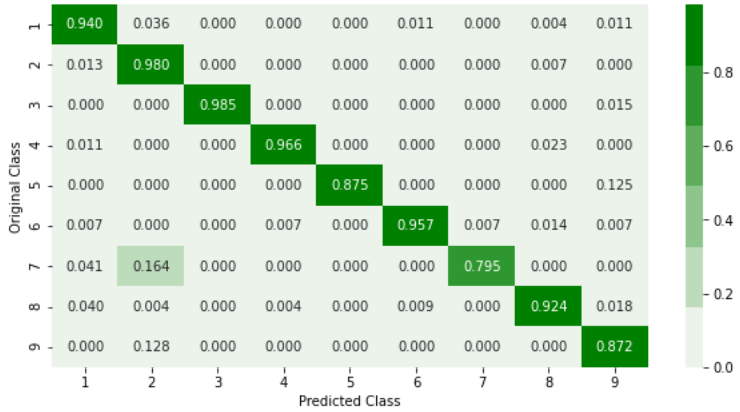

```
For values of best alpha =  3000 The train log loss is: 0.040431823312130695
For values of best alpha =  3000 The cross validation log loss is: 0.18706701008453522
For values of best alpha =  3000 The test log loss is: 0.18686918382077555
Number of misclassified points  5.0
---------- Confusion matrix ----------
```




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




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



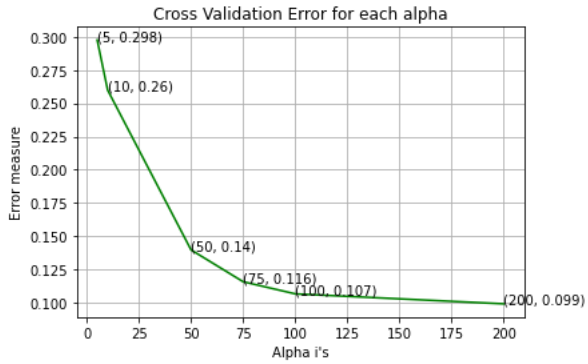

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.5. XgBoost Classification

In [ ]:

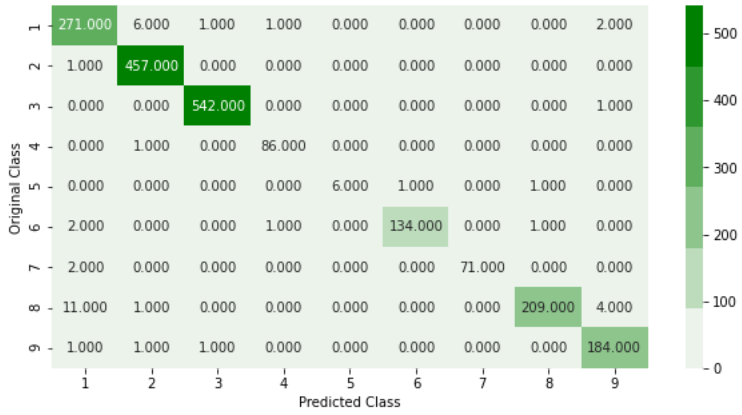
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------
alpha=[5,10,50, 75, 100, 200]
#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=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

%matplotlib inline
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])
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
log_loss for c =  5 is 0.29756897774831337
log_loss for c =  10 is 0.25992856480997256
log_loss for c =  50 is 0.13969844133355008
log_loss for c =  75 is 0.115724164814096
log_loss for c =  100 is 0.10659058736303624
log_loss for c =  200 is 0.09909347548706622
```

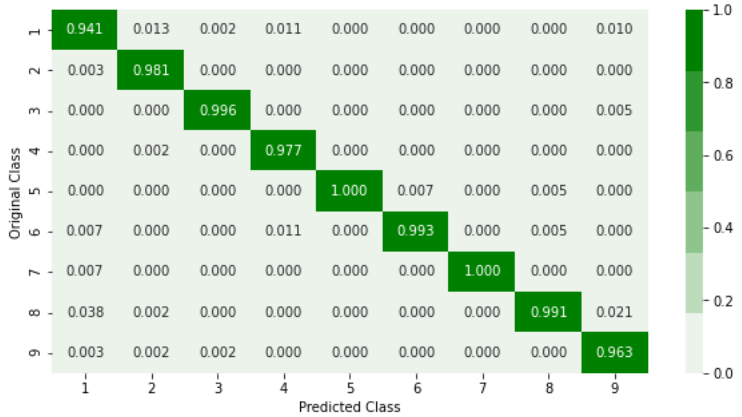



```
For values of best alpha =  200 The train log loss is: 0.03102634491817271
For values of best alpha =  200 The cross validation log loss is: 0.09909347548706622
For values of best alpha =  200 The test log loss is: 0.10286665430329858
Number of misclassified points  2.0
```

```
---------- Confusion matrix ----------
```




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




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


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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [ ]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
```

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[5,10,50, 75, 100, 200],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl1.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 52.6min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 105.9min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 121.9min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 158.3min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 180.0min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 217.4min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 244.1min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 268.1min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 277.3min finished
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [5, 10, 50, 75, 100,
                                                         200],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

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

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 0.1}
```

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

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=200, learning_rate=0.01, colsample_bytree=0.1, max_depth=5)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

```
train loss 0.07873788079852652
cv loss 0.16895940170210794
test loss 0.1727629156942633
```

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :

1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs

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

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

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

4.2.1 Feature extraction from asm files

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

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

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

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

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

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.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', 'ca
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
```

```
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
        with codecs.open('first/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                # https://www.tutorialspoint.com/python3/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


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

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
```

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


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

```
    file1.close()


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


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

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

In [ ]:

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

Out[ ]:

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

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	0	8	0	6	1
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	0	11	0	11	1

5 rows × 53 columns

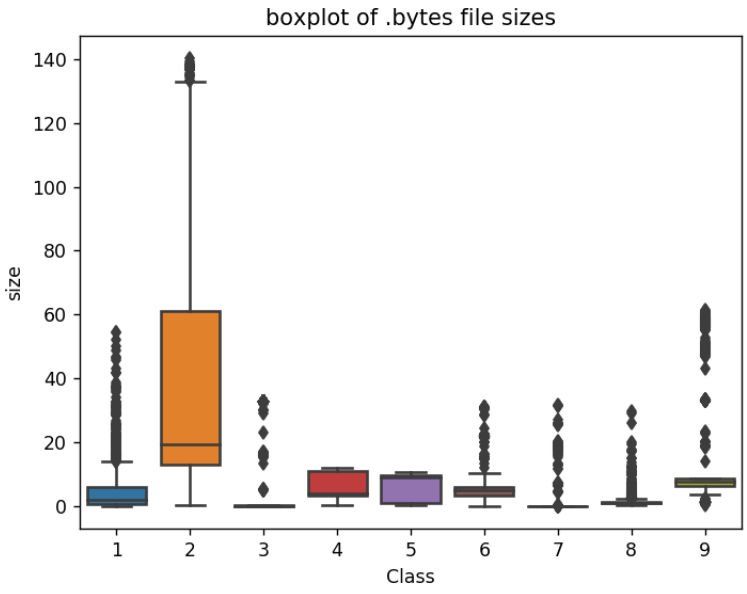
4.2.1.1 Files sizes of each .asm file

In [ ]:
```
#file sizes of byte files

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

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

4.2.1.2 Distribution of .asm file sizes

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



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

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

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	size

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	size
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	48	29	1	0.078190
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	0	20	1	0.063400
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	0	9	1	0.041695
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	0	6	1	0.018757
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	0	11	1	0.037567

5 rows × 54 columns

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

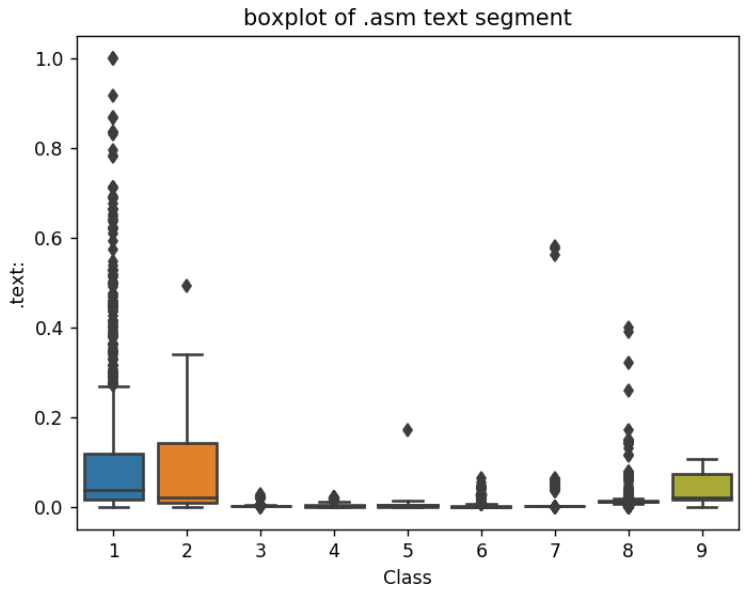
Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	ed
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.001057	0.(
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.000153	0.(
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.000178	0.(
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.000025	0.(
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000102	0.000362	0.000243	0.000064	0.(

5 rows × 54 columns

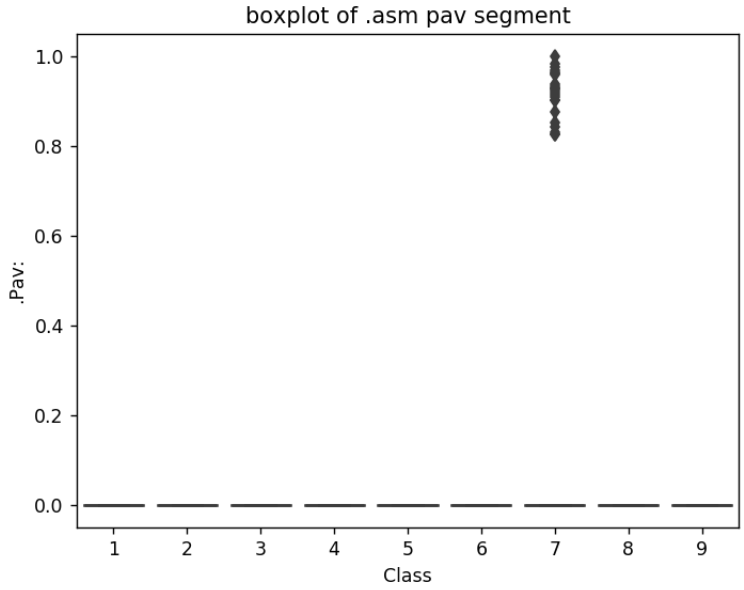
4.2.2 Univariate analysis on asm file features

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

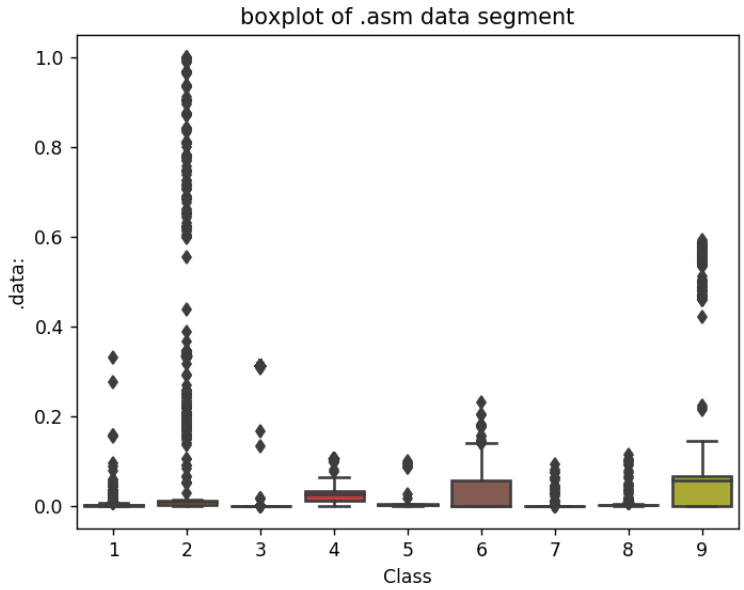


The plot is between Text and class
Class 1,2 and 9 can be easly separated

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

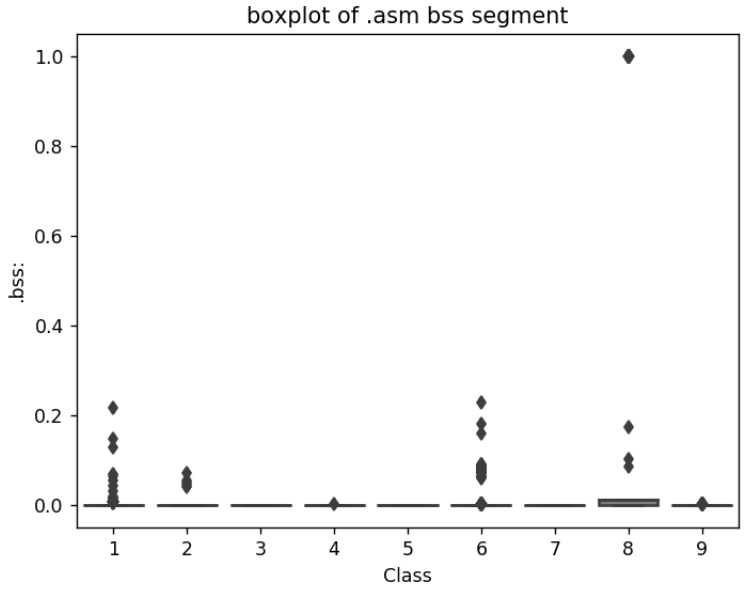


In [ ]:
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



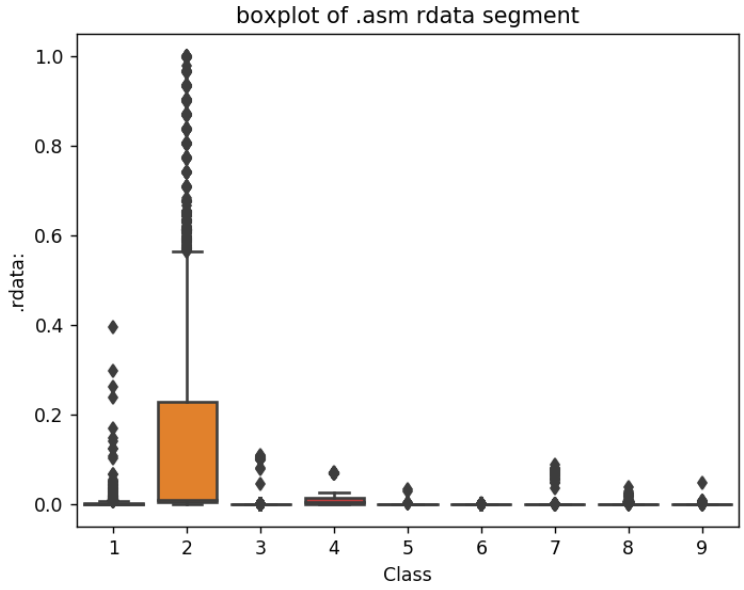
The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

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



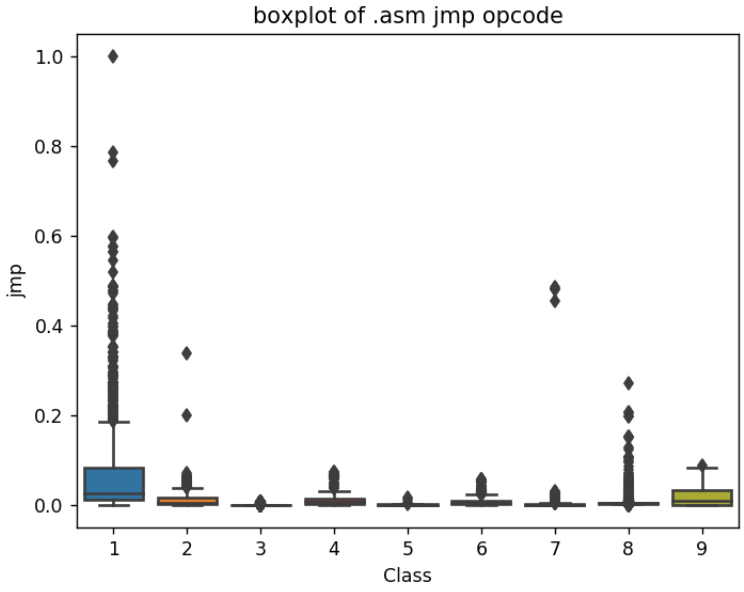
plot between bss segment and class label
very less number of files are having bss segment

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



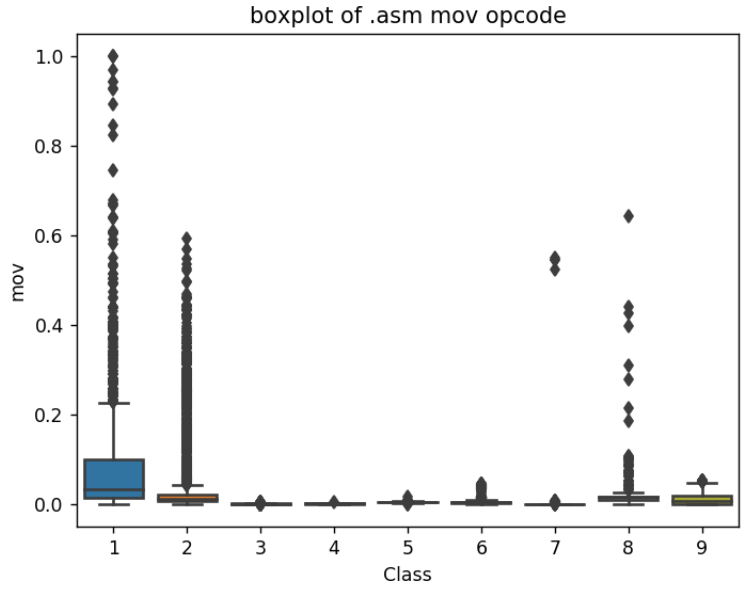
Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

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



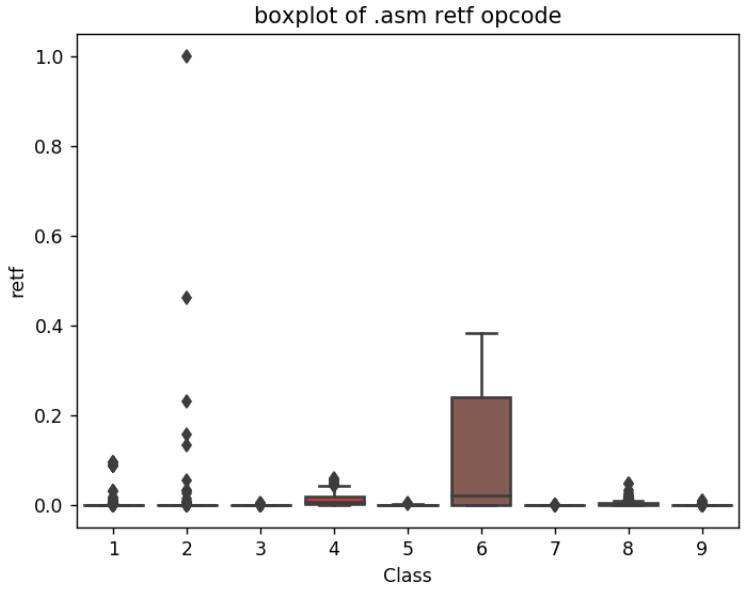
plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

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



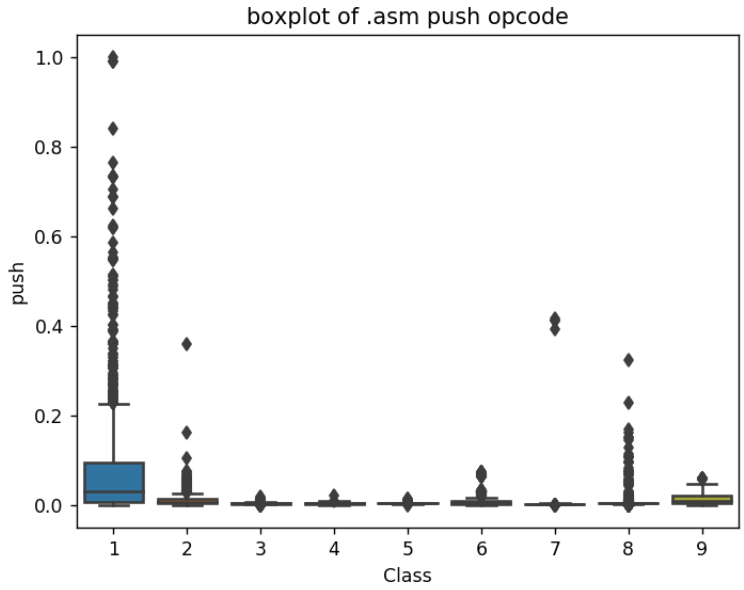
plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

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

In [ ]:
```
from PIL import Image
from pathlib import Path
asm_file_name = []
asm_arrays = []
if os.path.isdir('asm_image'):
    data_files = os.listdir('asm_image')
    for file in data_files:
        name=file.split('.')[0]
        asm_file_name.append(name)

        im = Image.open('asm_image/'+file,'r')
        pix_val = list(im.getdata())
        asm_arrays.append(pix_val)

asm_arrays = np.array(asm_arrays)
```

```
column_name = ['pixel : ' + str(i) for i in range(len(asm_arrays[0]))]

dic = {column_name[i]:asm_arrays[:,i] for i in range(len(asm_arrays[0]))}
dic.update({'ID':asm_file_name})

asm_file = pd.DataFrame(dic)
asm_file.to_csv('asm_file.csv')
```

In [ ]:
```
dfasm=pd.read_csv("asm_file.csv")
dfasm.shape
```

Out[ ]: (10861, 5002)

In [ ]:
```
dfasm.head(2)
```

Out[ ]:

	Unnamed: 0	pixel : 0	pixel : 1	pixel : 2	pixel : 3	pixel : 4	pixel : 5	pixel : 6	pixel : 7	pixel : 8	pixel : 9	pixel : 10	pixel : 11	pixel : 12	pixel : 13	pixel : 14	pixel : 15	pixel : 16	pixel : 17	pixel : 18	pixel : 19	pixel : 20	pixel : 21	pi :
0	0	68	71	69	66	65	88	60	39	51	54	41	48	43	49	45	16	13	5	10	15	0	25	
1	1	68	71	69	66	65	88	60	39	51	54	41	48	43	49	45	16	13	5	10	15	0	25	

2 rows × 5002 columns

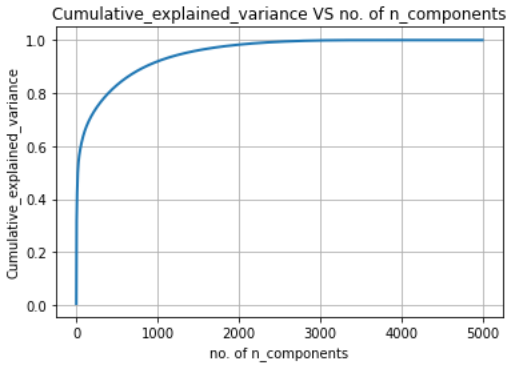
In [ ]:
```
dfasm = dfasm.drop(columns=['ID','Unnamed: 0'])
```

In [ ]:
```
dfasm.shape
```

Out[ ]: (10861, 5000)

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
#Desired dimensionality of output data. Must be strictly less than the number of features
max_features = 5000 - 1
svd = TruncatedSVD(n_components=max_features)
svd_data = svd.fit_transform(dfasm)
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)
```

In [ ]:
```
%matplotlib inline
# Plot the TrunvatedSVD spectrum
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.grid()
plt.xlabel('no. of n_components')
plt.ylabel('Cumulative_explained_variance')
plt.title("Cumulative_explained_variance VS no. of n_components")
plt.show()
```



In [ ]:
```
Optimal_components=3000

svd_trunc = TruncatedSVD(n_components=Optimal_components)
svd_transform = svd_trunc.fit_transform(dfasm)

print("Shape of data : ",svd_transform.shape)
```

```
Shape of data :  (10861, 3000)
```

In [ ]:
```
from PIL import Image
from pathlib import Path
asm_file_name = []
asm_arrays = []
if os.path.isdir('asm_image'):
    data_files = os.listdir('asm_image')
    for file in data_files:
        name=file.split('.')[0]
        asm_file_name.append(name)
```

In [ ]:
```
column_name = ['pixel : ' + str(i) for i in range(5000)]
best_fearures = [column_name[i] for i in svd_trunc.components_[0].argsort()[::-1][0:3000]]

dic = {best_fearures[i]:svd_transform[:,i] for i in range(3000)}
dic.update({'ID':asm_file_name})

asm_data = pd.DataFrame(dic)
```

In [ ]:
```
asm_data.head(5)
```

Out[ ]:

	pixel : 280	pixel : 192	pixel : 213	pixel : 284	pixel : 184	pixel : 308	pixel : 410	pixel : 170	pixel : 194	pixel : 171	pixel : 303	pixel :
0	4028.073683	-621.378059	-825.289935	44.564605	36.028959	5.400591	16.103856	-195.450080	-54.094948	-94.628160	-55.304150	34.171
1	4043.842574	-596.088299	-819.683412	53.186070	12.052960	-40.500301	-40.560276	-185.352045	-39.563901	-98.096166	-32.447496	34.683
2	3978.625730	-197.593367	620.467642	823.947573	-1038.939364	-868.822069	-356.841591	-173.415540	-45.200470	10.421191	-23.285332	-88.797
3	4076.798737	1803.476298	-488.815765	-375.889754	-470.498151	316.888703	-505.371024	-18.417842	35.587903	-6.421402	101.559309	59.293
4	3926.348769	-213.216141	328.024635	-57.563091	-13.597400	-67.282867	9.561821	311.221334	-144.700815	-194.759059	5.391663	-77.451

5 rows × 3001 columns

In [ ]:
```
asm_size_data = pd.read_csv('asm_with_size.csv')
asm_size_data = asm_size_data.drop(columns=['Unnamed: 0'])
asm_size_data.head(5)
```

Out[ ]:

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

In [ ]:
```
byte_features_with_size_asm = asm_data.merge(asm_size_data, on='ID')
byte_features_with_size_asm.to_csv("result_with_size_asm.csv")
byte_features_with_size_asm.head(2)
```

Out[ ]:

	pixel : 280	pixel : 192	pixel : 213	pixel : 284	pixel : 184	pixel : 308	pixel : 410	pixel : 170	pixel : 194	pixel : 171	pixel : 303	pixel : 210	pixel
0	4028.073683	-621.378059	-825.289935	44.564605	36.028959	5.400591	16.103856	-195.450080	-54.094948	-94.628160	-55.304150	34.171393	-50.24
1	4043.842574	-596.088299	-819.683412	53.186070	12.052960	-40.500301	-40.560276	-185.352045	-39.563901	-98.096166	-32.447496	34.683291	-61.71

2 rows × 3003 columns

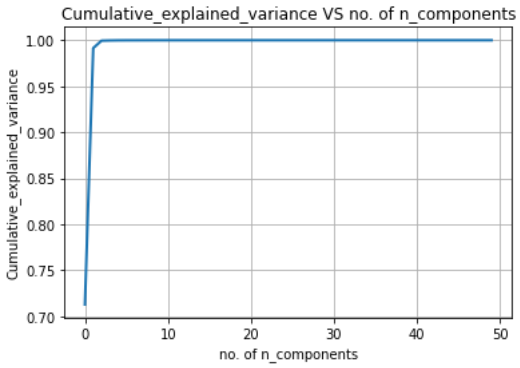
In [ ]:
```
dfasm=pd.read_csv("asmoutputfile.csv")
dfasm.shape
```

Out[ ]:
```
(10868, 52)
```

In [ ]:
```
dfasm = dfasm.drop(columns=['ID'])
```

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
#Desired dimensionality of output data. Must be strictly less than the number of features
max_features = 51-1
svd = TruncatedSVD(n_components=max_features)
svd_data = svd.fit_transform(dfasm)
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)
```

In [ ]:
```
%matplotlib inline
# Plot the TrunvatedSVD spectrum
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.grid()
plt.xlabel('no. of n_components')
plt.ylabel('Cumulative_explained_variance')
plt.title("Cumulative_explained_variance VS no. of n_components")
plt.show()
```



In [ ]:
```
Optimal_components=20

svd_trunc = TruncatedSVD(n_components=Optimal_components)
svd_transform = svd_trunc.fit_transform(dfasm)

print("Shape of data : ",svd_transform.shape)
```

```
Shape of data :  (10868, 20)
```

In [ ]:
```
file_names = []
for i , row in dfasm['ID'].iteritems():
  file_names.append(row)
```

In [ ]:
```
column_name = [col for col in dfasm.columns]
```

In [ ]:
```
dic = {column_name[i]:svd_transform[:,i] for i in range(20)}
dic.update({'ID':file_names})

asm_data = pd.DataFrame(dic)
asm_data.head(2)
```

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	.reloc:
0	01kcPWA9K2BOxQeS5Rju	60.084337	770.513000	-48.622595	55.684512	30.350897	50.106603	4.286030	1.818266	123.882334	3.243021	9.756179
1	1E93CpP60RHFNiT5Qfvn	59.386191	859.761233	-78.242932	31.445927	38.780822	-33.725334	-12.136069	1.790523	97.816607	16.830043	-9.452131

In [ ]:
```
byte_features_with_size_asm_pixel = asm_data.merge(byte_features_with_size_asm, on='ID')
byte_features_with_size_asm_pixel.to_csv("result_with_size_asm_pixel.csv")
byte_features_with_size_asm_pixel.head(2)
```

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	.reloc:
0	01kcPWA9K2BOxQeS5Rju	60.084337	770.513000	-48.622595	55.684512	30.350897	50.106603	4.286030	1.818266	123.882334	3.243021	9.756179
1	1E93CpP60RHFNiT5Qfvn	59.386191	859.761233	-78.242932	31.445927	38.780822	-33.725334	-12.136069	1.790523	97.816607	16.830043	-9.452131

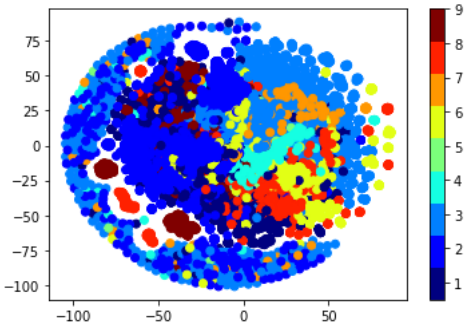
2 rows × 3022 columns

In [5]:
```
byte_features_with_size_asm_pixel = pd.read_csv('result_with_size_asm_pixel.csv')

byte_features_with_size_asm_pixel = byte_features_with_size_asm_pixel.drop(columns = ['Unnamed: 0'])

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

asm_x = result.drop(['ID','Class','.BSS:','.CODE'], axis=1)
asm_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y_train_asm,test_size=0.20)
```

4.2.2 Multivariate Analysis on .asm file features

In [ ]:
```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic-neighbourhood-embeddingt-sne-part-1/

#multivariate analysis on byte files
```

```
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.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()
```



4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [ ]:
```
asm_x = result.drop(['ID','Class','.BSS:','.CODE'], axis=1)
asm_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y_train_asm,test_size=0.20)
```

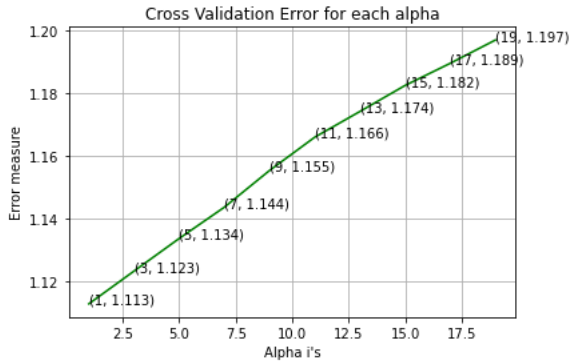
4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

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

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


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

alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-15))
```

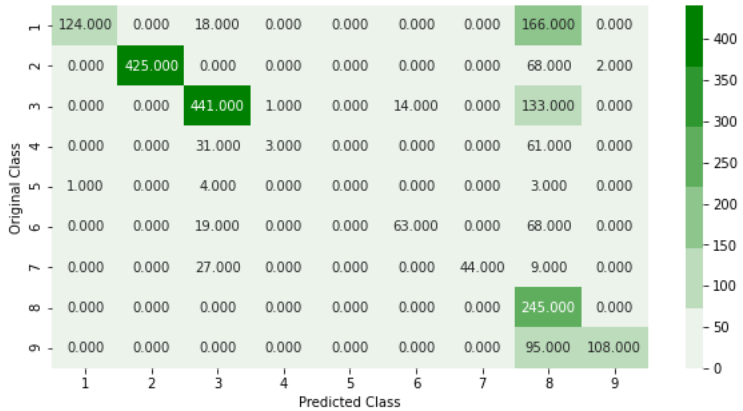
```
for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

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

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


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

```
log_loss for k =  1 is 1.1128050516867647
log_loss for k =  3 is 1.1232378584493727
log_loss for k =  5 is 1.1336249187601248
log_loss for k =  7 is 1.1435836592725297
log_loss for k =  9 is 1.155274917468895
log_loss for k =  11 is 1.1658704657096088
log_loss for k =  13 is 1.1740674465229672
log_loss for k =  15 is 1.1823571369783807
log_loss for k =  17 is 1.1894153441882414
log_loss for k =  19 is 1.1968287262284452
```

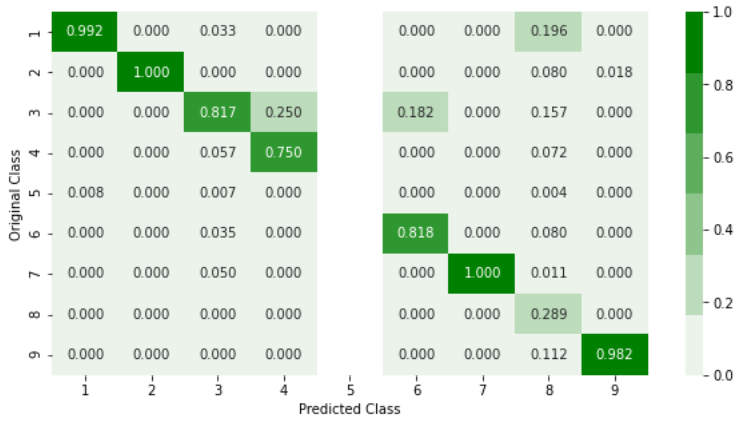



```
log loss for train data 0.44461102614404535
log loss for cv data 1.1128050516867647
log loss for test data 1.1129272182257988
Number of misclassified points  33.13391624482283
```

---------- Confusion matrix ----------

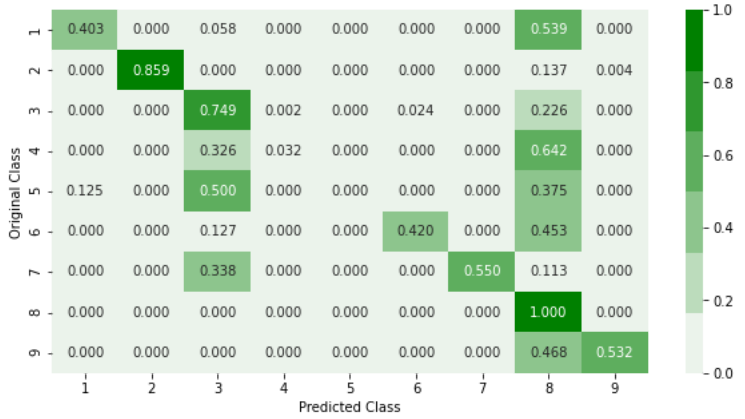


---------- Precision matrix ----------




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

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




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

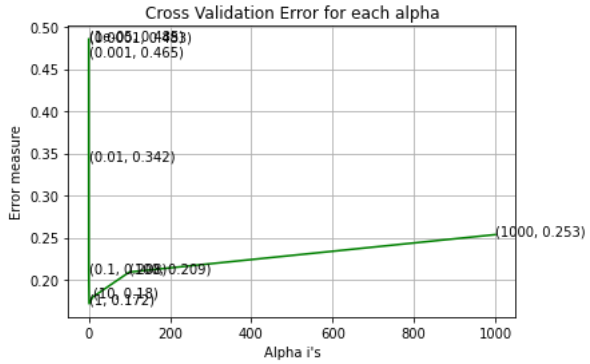
4.4.2 Logistic Regression

In [ ]:

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

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

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


alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

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

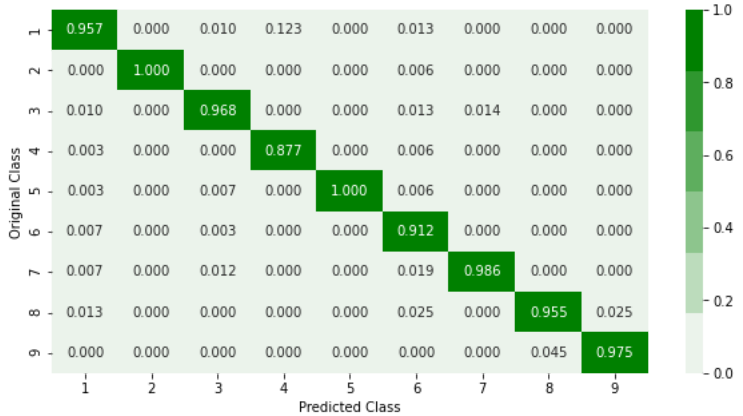
```
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

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

```
log_loss for c =  1e-05 is 0.4853680393101723
log_loss for c =  0.0001 is 0.48344049804905337
log_loss for c =  0.001 is 0.4651545258753147
log_loss for c =  0.01 is 0.34222683358826195
log_loss for c =  0.1 is 0.20818604916128208
log_loss for c =  1 is 0.17162789410407123
log_loss for c =  10 is 0.17950977632732232
log_loss for c =  100 is 0.20912620537955076
log_loss for c =  1000 is 0.25349538717486286
```

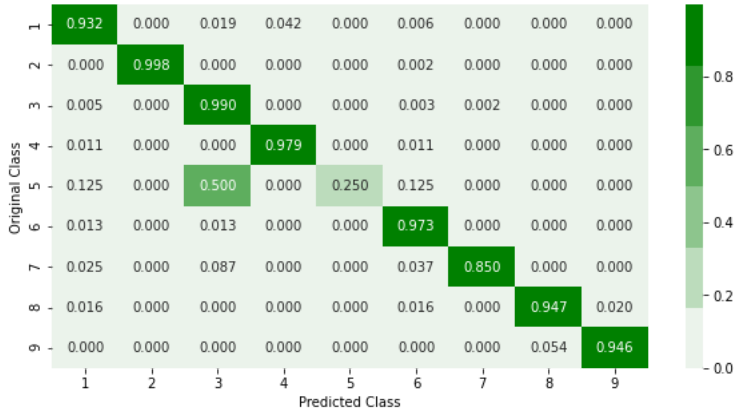



```
log loss for train data 0.04906307619690729
log loss for cv data 0.17162789410407123
log loss for test data 0.15652387661296083
Number of misclassified points  3.4974689369535206
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```


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




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



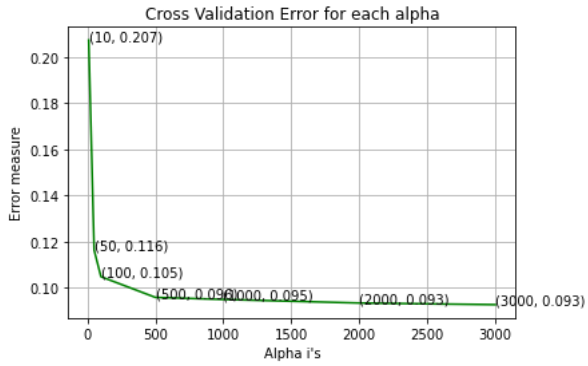

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.3 Random Forest Classifier

In [ ]:

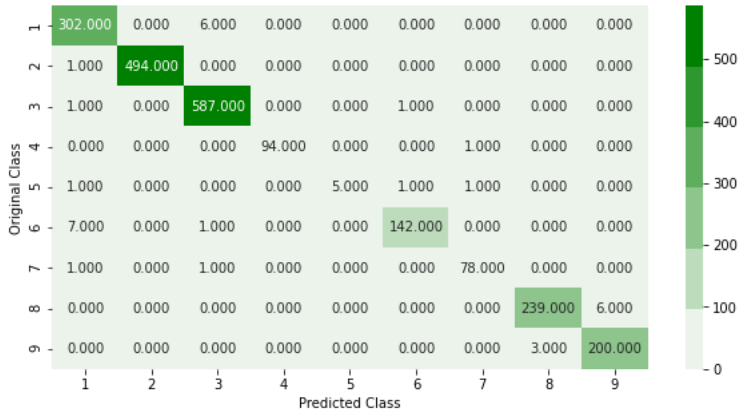
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# ------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

%matplotlib inline

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

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.2072670657560012
log_loss for c =  50 is 0.11630862401435027
log_loss for c =  100 is 0.10480428610204123
log_loss for c =  500 is 0.09583768172782696
log_loss for c =  1000 is 0.09494190235279833
log_loss for c =  2000 is 0.09339530054969708
log_loss for c =  3000 is 0.092680566294761
```

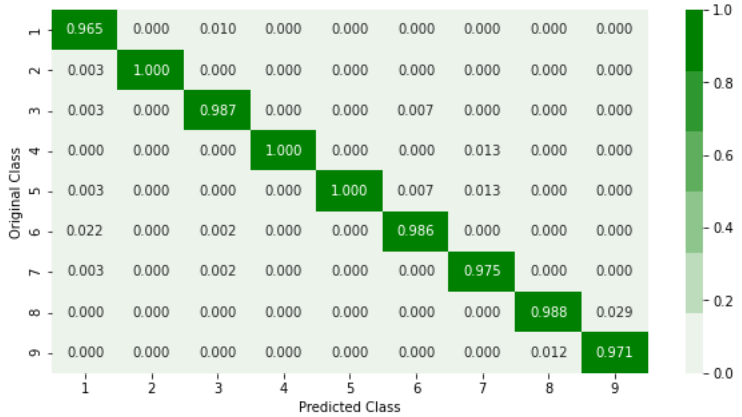



```
log loss for train data 0.01817532012902186
log loss for cv data 0.092680566294761
log loss for test data 0.08206900335960962
Number of misclassified points  1.4726184997699034
```

```
---------- Confusion matrix ----------
```




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




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

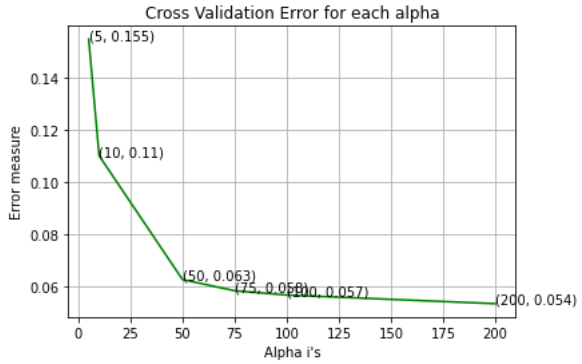

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.4 XgBoost Classifier

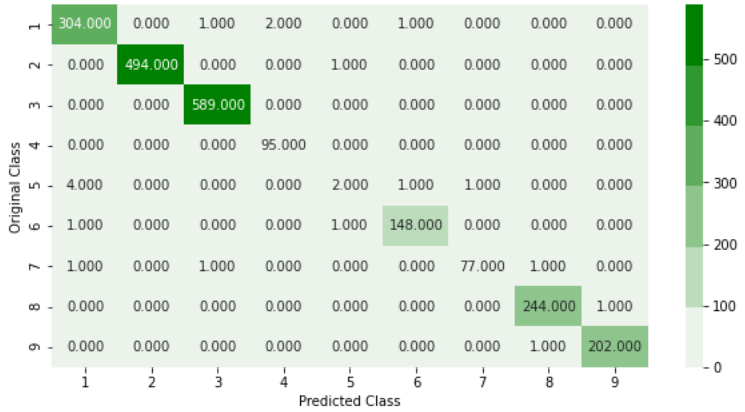
In [7]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data
```

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[5,10,50, 75, 100, 200]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

%matplotlib inline
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])
x_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)

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

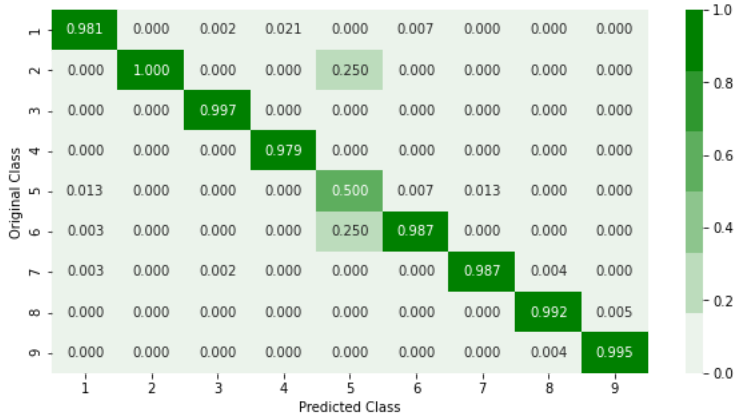
```
log_loss for c =  5 is 0.1549142982666909
log_loss for c =  10 is 0.11021986195076798
log_loss for c =  50 is 0.06275195691776098
log_loss for c =  75 is 0.058473674968804526
log_loss for c =  100 is 0.056817066861299954
log_loss for c =  200 is 0.05351652976313305
```




```
For values of best alpha =  200 The train log loss is: 0.0168573742206137
For values of best alpha =  200 The cross validation log loss is: 0.05351652976313305
For values of best alpha =  200 The test log loss is: 0.045533178785614095
Number of misclassified points  0.8283479061205707
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```

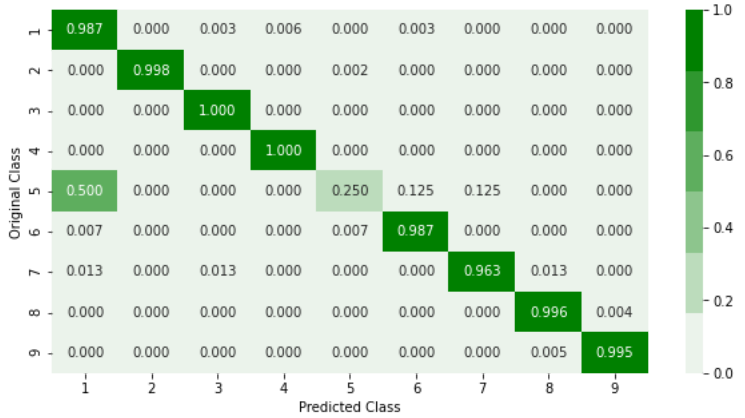


---------- Precision matrix ----------



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

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



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

4.4.5 Xgboost Classifier with best hyperparameters

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

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[5,10,50, 75, 100, 200],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_asm,y_train_asm)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  9.5min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 19.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 32.4min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 37.4min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 58.4min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 63.9min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 75.9min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 100.8min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 105.2min finished
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score=nan,
```

```
                         estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                                 colsample_bylevel=1,
                                                 colsample_bynode=1,
                                                 colsample_bytree=1, gamma=0,
                                                 learning_rate=0.1, max_delta_step=0,
                                                 max_depth=3, min_child_weight=1,
                                                 missing=None, n_estimators=100,
                                                 n_jobs=1, nthread=None,
                                                 objective='binary:logistic',
                                                 random_state=0, reg_alpha=0,
                                                 reg_lambda=1...
                                                 seed=None, silent=None, subsample=1,
                                                 verbosity=1),
                         iid='deprecated', n_iter=10, n_jobs=-1,
                         param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                              'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                                0.15, 0.2],
                                              'max_depth': [3, 5, 10],
                                              'n_estimators': [5, 10, 50, 75, 100,
                                                               200],
                                              'subsample': [0.1, 0.3, 0.5, 1]},
                         pre_dispatch='2*n_jobs', random_state=None, refit=True,
                         return_train_score=False, scoring=None, verbose=10)
```

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

```
{'subsample': 0.3, 'n_estimators': 200, 'max_depth': 10, 'learning_rate': 0.2, 'colsample_bytree': 0.3}
```

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

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=200,subsample=0.3,learning_rate=0.2,colsample_bytree=0.3,max_depth=10)
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

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

```
train loss 0.01767014929396549
cv loss 0.035097405720627785
test loss 0.04937553694684861
```

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

4.5.1. Merging both asm and byte file features

In [5]:
```
byte_features_with_size_asm_pixel = pd.read_csv('result_with_size_asm_pixel.csv')

byte_features_with_size_asm_pixel = byte_features_with_size_asm_pixel.drop(columns = ['Unnamed: 0'])

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

In [7]:
```
byte_features_with_size_binary_pixel = pd.read_csv('result_with_size_binary_pixel.csv')

byte_features_with_size_binary_pixel = byte_features_with_size_binary_pixel.drop(columns = ['Unnamed: 0'])

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

```
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
result_binary = normalize(byte_features_with_size_binary_pixel)
```

In [8]:
```
print(result.shape)
print(result_binary.shape)
```

```
(10861, 3022)
(10000, 3802)
```

In [9]:
```
result_x = pd.merge(result_binary,result.drop(['Class'], axis=1),on='ID')
result_y = result_x['Class']
result_x = result_x.drop(['ID','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

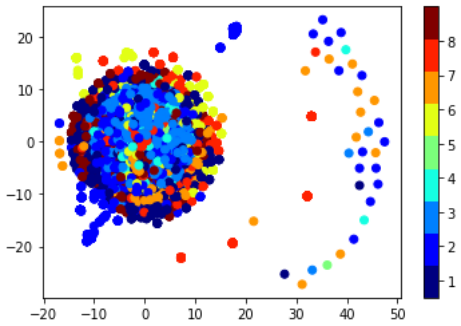
Out[9]:

	fd	b4	6e	78	de	1a	79	0d	72	a1	bd	cf	95	b1	2e
0	0.020686	0.241691	0.245214	0.074610	0.193234	0.401354	0.258731	0.246999	0.213273	0.437874	0.238340	0.216853	0.286667	0.328151	0.467180
1	0.026070	0.240115	0.244722	0.074600	0.193291	0.401774	0.258677	0.246983	0.212736	0.436829	0.238462	0.216140	0.285696	0.328129	0.467268
2	0.011070	0.243824	0.248488	0.075041	0.193600	0.401087	0.258808	0.247447	0.213158	0.437002	0.238877	0.216545	0.286434	0.327896	0.467292
3	0.122864	0.215803	0.212897	0.070061	0.194897	0.403846	0.258361	0.241398	0.213872	0.438413	0.233919	0.214436	0.285582	0.331177	0.468386
4	0.076206	0.228028	0.230242	0.073189	0.192625	0.403211	0.258394	0.245530	0.211028	0.436740	0.237324	0.215298	0.284060	0.328597	0.466846

5 rows × 6818 columns

4.5.2. Multivariate Analysis on final fearures

In [12]:
```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x)
vis_x = results[:, 0]
vis_y = results[:, 1]
%matplotlib inline
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

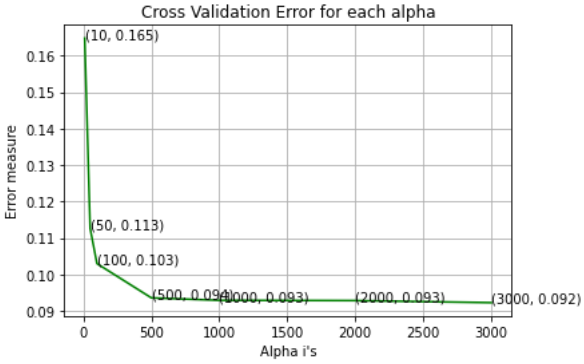
In [10]:
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

4.5.4. Random Forest Classifier on final features

In [12]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# --------------------------------
%matplotlib inline
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
```

```
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=r_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))

plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

```
log_loss for c =  10 is 0.1648228157139891
log_loss for c =  50 is 0.11264535062238791
log_loss for c =  100 is 0.10296612343298973
log_loss for c =  500 is 0.09356457153200276
log_loss for c =  1000 is 0.09288354041113073
log_loss for c =  2000 is 0.0928126102300245
log_loss for c =  3000 is 0.09224195781784245
```




```
For values of best alpha =  3000 The train log loss is: 0.02407627367677853
For values of best alpha =  3000 The cross validation log loss is: 0.09224195781784245
For values of best alpha =  3000 The test log loss is: 0.08953597147086867
```

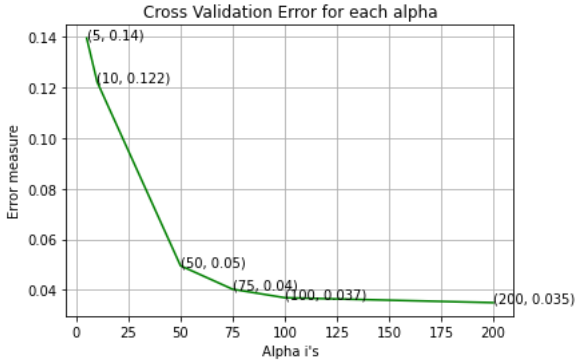
4.5.5. XgBoost Classifier on final features

In [ ]:

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

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------
%matplotlib inline
alpha=[5,10,50, 75, 100, 200]
cv_log_error_array=[]
for i in alpha:
```

```
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)
```

```
log_loss for c =  5 is 0.13956465761568346
log_loss for c =  10 is 0.12227040025832969
log_loss for c =  50 is 0.04955315218213867
log_loss for c =  75 is 0.040258576342046666
log_loss for c =  100 is 0.03690728920223634
log_loss for c =  200 is 0.03494765927579046
```



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

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

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[5,10,50, 75, 100, 200],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 45.4min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 91.3min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 113.6min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 135.7min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 206.5min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 333.8min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 405.9min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 467.5min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 469.6min finished
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=-1,
```

```
                    param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                         'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                           0.15, 0.2],
                                         'max_depth': [3, 5, 10],
                                         'n_estimators': [5, 10, 50, 75, 100,
                                                          200],
                                         'subsample': [0.1, 0.3, 0.5, 1]},
                    pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score=False, scoring=None, verbose=10)
```

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

```
{'subsample': 1, 'n_estimators': 50, 'max_depth': 3, 'learning_rate': 0.2, 'colsample_bytree': 0.3}
```

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------
x_cfl=XGBClassifier(n_estimators=50,max_depth=3,learning_rate=0.2,colsample_bytree=0.3,subsample=1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)


predict_y = sig_clf.predict_proba(X_train_merge)
print("The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print("The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print("The test log loss is:",log_loss(y_test_merge, predict_y))
```

```
The train log loss is: 0.013679148302896717
The cross validation log loss is: 0.036970956585008
The test log loss is: 0.027328810784065394
```